

# **Stauerkennung auf Autobahnwebcam-Bildern mit geringer Ressourcenverwendung**

## **Studienarbeit**

für die Prüfung zum  
Bachelor of Science

**Studiengang Angewandte Informatik**  
Duale Hochschule Baden-Württemberg Karlsruhe

von  
Maurice Heumann & Mirko Müller

Abgabedatum:	17. September 2019
Matrikelnummer:	3970752 & 8255829
Kurs:	TINF16B4
Ausbildungsfirma:	CAS Software AG
Betreuer der Ausbildungsfirma:	Dr. Christian Bomhardt

# Eidesstattliche Erklärung

Erklärung gemäß § 5 (3) der "Studien- und Prüfungsordnung DHBW Technik" vom 22. September 2011.

Ich versichere hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Karlsruhe, den 15. Mai 2019

---

MAURICE HEUMANN & MIRKO MÜLLER

---

## Sperrvermerk

Die Ergebnisse der Arbeit stehen ausschließlich dem auf dem Deckblatt aufgeführten Ausbildungsbetrieb zur Verfügung.

## Copyrightvermerk

Dieses Werk einschließlich seiner Teile ist **urheberrechtlich geschützt**. Jede Verwertung außerhalb der engen Grenzen des Urheberrechtsgesetzes ist ohne Zustimmung des Autors unzulässig und strafbar. Das gilt insbesondere für Vervielfältigungen, Übersetzungen, Mikroverfilmungen sowie die Einspeicherung und Verarbeitung in elektronischen Systemen.

© 2018

# **Abstract**

Frequent traffic jams on German highways often remain undetected by public navigation services. This paper demonstrates a method to solve this problem by processing and evaluating publicly available images of traffic cameras. Pictures are analyzed using image processing algorithms like background subtraction. The method differs from previous implementations due to the selection of algorithms being focused on resource efficiency while staying as accurate as possible. The showcased method is implemented as an android application.

# Inhaltsverzeichnis

<b>Eidesstattliche Erklärung</b>	<b>I</b>
<b>Abkürzungsverzeichnis</b>	<b>VI</b>
<b>Abbildungsverzeichnis</b>	<b>VII</b>
<b>Tabellenverzeichnis</b>	<b>IX</b>
<b>1 Einleitung</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Ziel der Arbeit . . . . .	1
<b>2 Grundlagen</b>	<b>3</b>
2.1 Straßenverkehrszentrale Baden-Württemberg . . . . .	3
2.2 Histogramm . . . . .	3
2.3 Faltungskerne . . . . .	4
2.3.1 Gauß-Filter . . . . .	5
2.3.2 Laplace-Filter . . . . .	5
2.3.3 Sobel-Operator . . . . .	6
2.4 Otsu . . . . .	7
2.5 Haar Kaskaden . . . . .	8
2.6 Morphologische Operatoren . . . . .	9
2.6.1 Erosion . . . . .	9
2.6.2 Dilatation . . . . .	10
2.6.3 Opening . . . . .	10
2.6.4 Closing . . . . .	11
2.7 OpenCV . . . . .	12
<b>3 Analyse</b>	<b>13</b>
3.1 Stand der Technik . . . . .	13
3.1.1 Statische Verfahren . . . . .	13
3.1.1.1 Pixelorientierte Bildanalyse . . . . .	13
3.1.1.2 Kantenerkennung . . . . .	16
3.1.2 Dynamische Verfahren . . . . .	18
3.1.2.1 Neuronale Netze . . . . .	18

3.1.2.2	Background-Subtraction . . . . .	20
3.2	Bestimmung des Testdatensatzes . . . . .	22
3.2.1	Verkehrskameras laden . . . . .	22
3.2.2	Verkehrsbilder laden . . . . .	23
3.2.3	Vergleichsdatensätze generieren . . . . .	23
3.3	Auswahl des Verfahrens . . . . .	24
3.3.1	Helligkeit . . . . .	24
3.3.2	Haar-Features . . . . .	24
3.3.3	Edge-Detection . . . . .	25
3.3.4	Background-Subtraction . . . . .	26
3.3.5	Auswahl . . . . .	26
<b>4</b>	<b>Implementierung</b>	<b>29</b>
4.1	Backend . . . . .	29
4.1.1	Infrastruktur . . . . .	29
4.1.2	Datenerhebung . . . . .	32
4.1.3	Datenpersistenz . . . . .	33
4.1.4	Datenbereitstellung . . . . .	35
4.2	Client . . . . .	36
4.2.1	OpenCV . . . . .	36
4.2.2	Backend Kommunikation . . . . .	37
4.2.3	Geolokalisierung . . . . .	38
4.2.4	Richtungsfeststellung . . . . .	38
4.2.5	Verkehrskameras laden . . . . .	39
4.2.6	Bilder laden . . . . .	40
4.2.7	Bilder verarbeiten . . . . .	41
4.2.8	Bluetooth . . . . .	46
4.2.9	Benutzeroberfläche . . . . .	46
4.2.10	Stauansage . . . . .	48
<b>5</b>	<b>Bewertung</b>	<b>51</b>
5.1	Abweichungen der Systeme . . . . .	51
5.2	Kennzahlen . . . . .	52
5.3	Datenerhebung . . . . .	53
5.4	Auswertung . . . . .	54
5.4.1	Backend . . . . .	54
5.4.2	Frontend . . . . .	55
<b>6</b>	<b>Zusammenfassung</b>	<b>57</b>
6.1	Fazit . . . . .	57
6.2	Ausblick . . . . .	58



# Abkürzungsverzeichnis

API	Application Programming Interface
DHBW	Duale Hochschule Baden-Württemberg
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
JPEG	Joint Photographic Experts Group
JSON	JavaScript Object Notation
PNG	Portable Network Graphics
SQL	Structured Query Language
SSL	Secure Sockets Layer
SVZ	Straßenverkehrszentrale
TLS	Transport Layer Security
UI	User Interface

# Abbildungsverzeichnis

2.1	Histogramm über die Helligkeitswerte eines Bildes . . . . .	4
2.2	Anwendung eines Weichzeichnungsfilters . . . . .	5
2.3	Anwendung eines Laplace-Filters . . . . .	6
2.4	Anwendung eines Sobel-Operators . . . . .	7
2.5	Segmentierung eines Bildes mit dem Otsu Verfahren . . . . .	8
2.6	Haar Features . . . . .	8
2.7	Anwendung der Erosion auf ein Bild . . . . .	10
2.8	Anwendung der Dilatation auf ein Bild . . . . .	11
2.9	Anwendung von Opening auf ein Bild . . . . .	11
2.10	Anwendung von Closing auf ein Bild . . . . .	12
3.1	Detektor im Einsatz . . . . .	15
3.2	Kantenerkennung . . . . .	17
3.3	Ergebnis des Verfahrens . . . . .	17
3.4	Visualisierung der Schichten eines konvolutionalen neuronalen Netzwerks (unten Eingang - oben Ausgabe) . . . . .	19
3.5	Background-Subtraction im Einsatz . . . . .	21
4.1	Server Architektur . . . . .	31
4.2	Datenbank Schema . . . . .	33
4.3	Zu verarbeitendes Beispielbild . . . . .	42
4.4	Ergebnis nach der Vorverarbeitung . . . . .	43
4.5	Ergebnis nach der Verarbeitung . . . . .	43
4.6	Ergebnis nach der Nachbearbeitung . . . . .	44
4.7	Konturen erkennen und schließen . . . . .	45
4.8	Vollständig verarbeitetes Bild . . . . .	45
4.9	Kartenansicht der Anwendung . . . . .	47
4.10	Detailansicht der Anwendung . . . . .	48
4.11	Einstellungen der Anwendung . . . . .	49
5.1	Ressourcenverbrauch alter Server in htop . . . . .	54
5.2	Ressourcenverbrauch neuer Server in htop . . . . .	54
5.3	Ressourcenverbrauch alter Client . . . . .	55

5.4 Ressourcenverbrauch neuer Client . . . . .	55
--	----

# Tabellenverzeichnis

3.1	Tabellenstruktur der Kamerliste . . . . .	22
3.2	Ergebnisse des Vergleichsalgorithmus . . . . .	26
3.3	Ergebnisse des Helligkeitsvergleichs . . . . .	27
3.4	Ergebnisse der Haar-Features . . . . .	27
3.5	Ergebnisse der Kantenerkennung . . . . .	28
3.6	Ergebnisse der Background-Subtraction . . . . .	28

# **1 Einleitung**

## **1.1 Motivation**

Öffentliche Stauinformationen sind oftmals nicht akkurat oder aktuell genug, damit Autofahrer auf die Verkehrslage reagieren können. Deshalb befasst sich diese Arbeit mit der Auswertung von Verkehrsüberwachungskameras der Straßenverkehrszentrale Baden-Württemberg. Mit diesem Ansatz kann Autofahrern zeitnah vorgeschlagen werden, eine Alternativroute zu befahren und somit Zeit einzusparen. Eine einfache Möglichkeit, die Daten an den Nutzer zu übermitteln, ohne dass dieser während des Autofahrens beeinträchtigt wird, ist diese akustisch wiederzugeben. Mögliche Schnittstellen bieten hierbei das Autoradio oder das Smartphone des Fahrers. Das dynamisch reagierende System präsentierte von M. Herglotz, S. Knab und J. Kümmerlin in [5], löst das Problem, ist für den produktiven Einsatz jedoch zu ressourcenintensiv.

## **1.2 Ziel der Arbeit**

In der vorliegenden Arbeit soll eine alternative Methode zur Erkennung von stark stckendem oder zum Stillstand gekommenem Verkehr auf Autobahnen entwickelt werden. Als Methode wird hierbei ein Algorithmus bezeichnet, der mit Hilfe von Bildverarbeitungsverfahren die Fahrdynamik auf Verkehrsüberwachungskameras evaluiert. Bilder der Überwachungskameras werden über die Website der Straßenverkehrszentrale Baden-Württemberg minütlich zur Verfügung gestellt. Ausgewertete Ergebnisse sollen dem Nutzer über eine Android Anwendung übermittelt werden. Eine Schnittstelle zum Autoradio des Nutzers wird hierbei über das Übertragungsprotokoll Bluetooth ermöglicht. Falls die Schnittstelle nicht angesprochen werden kann, wird die Standard-Audio-Ausgabe des jeweiligen Android-Gerätes verwendet. Die Android Anwendung muss ihre Daten nicht direkt von der Straßenverkehrszentrale beziehen, sondern kann

diese auch indirekt über einen bereitgestellten Server abrufen. Dieser agiert dabei als Zwischenspeicher für Kamerabilder und bietet eine Historie. Die Implementierung der Arbeit soll, besonders bezüglich Kennzahlen der Ressourceneffizienz, optimiert werden. Die Kennzahlen umfassen hierbei:

- Energiebedarf
- Hauptspeicherbedarf
- Sekundärspeicherbedarf
- Laufzeit

Die Lösung soll nicht genauer als vorherige Ansätze sein, sondern ein ausgewogenes Maß zwischen Genauigkeit und Ressourceneffizienz finden.

# **2 Grundlagen**

Dieses Kapitel beschreibt Grundlagen, die für das weitere Verständnis der Arbeit benötigt werden.

## **2.1 Straßenverkehrszentrale Baden-Württemberg**

Aktuelle Verkehrsinformationen auf den Autobahnen und Bundesstraßen in Baden-Württemberg und der näheren Umgebung lassen sich über das Straßenverkehrszentrale Baden-Württemberg abrufen. Die Straßenverkehrszentrale nimmt sich dabei unter anderem zum Ziel, Stau auf den Straßen zu reduzieren und Autofahrer über aktuelle Verkehrsbehinderungen zu informieren. Durch verschiedene Verkehrskameras und Sensoren wird der Verkehr auf Straßen analysiert. Über flexible Tempolimits und einer Freigabe von Seitenstreifen wird die Last der Straßen reduziert und der Verkehrsfluss verbessert [2].

## **2.2 Histogramm**

Ein Histogramm visualisiert die Häufigkeitsverteilung von Grauwertklassen innerhalb eines Grauwertbildes. Hierzu wird der Grauwert jedes Bildpunktes ausgewertet und gezählt, wie oft welcher Grauwert innerhalb des Bildes vorkommt (Abbildung 2.1).

Anhand des Histogramms lässt sich beispielsweise der Kontrast eines Bildes erkennen. Dieser beschreibt die Spannweite zwischen dem dunkelsten und dem hellsten Punktes in einem Bild, bzw. der Klassen in einem Histogramm. Dadurch lässt sich auch der Kontrast erhöhen, indem eine Spreizung des Histogramms durchgeführt wird: Der Wertebereich der belegten Grauwertklassen wird auf den gesamten Farbbereich abgebildet

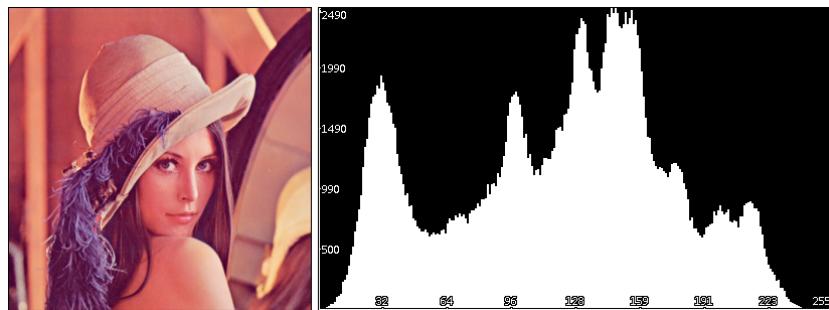


Abbildung 2.1: Histogramm über die Helligkeitswerte eines Bildes

Quelle: <http://opensource.graphics/tag/histogram/> (13.3.2019)

## 2.3 Faltungskerne

Faltungskerne beschreiben Filter in der Bildverarbeitung, die über die diskrete Faltung im 2-dimensionalen Raum auf ein Bild angewendet werden.

Grundsätzlich ist die 1-dimensionale Faltung im kontinuierlichen Raum durch die Integration zweier Funktionen  $g$  und  $f$  an einem Punkt  $t$  definiert, wobei die Funktion  $g$  gespiegelt wird, also auf  $f$  gefaltet wird:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t - \tau)d\tau$$

Für die Bildverarbeitung ist jedoch die Faltung im diskreten 2-dimensionalen Raum relevant. Hierfür wird statt dem Integral, die Doppelsumme über alle Werte  $n$  gebildet (in  $x$ - und  $y$ -Richtung) und der Filter  $k$  mit dem Bild  $I$  an einem Punkt  $(x, y)$  gefaltet:

$$I^*(x, y) = \sum_{i=1}^n \sum_{j=1}^n I(x - i, y - j)k(i, j)$$

Hiermit wird nun eine Faltungsmatrix, bzw. ein Faltungskern auf jeden Pixel im Bild angewendet:

$$k = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

Faltungskerne sind lokale Operatoren, die die Neuberechnung eines Pixels mittels eines Teilbereichs des Bildes durchführen. Mit solchen Filtern lassen sich Bil-

der beispielsweise Schärfen, Glätten. Es lassen sich jedoch auch Kanten finden oder Rauschanteile reduzieren.

### 2.3.1 Gauß-Filter

Der Gauß-Filter ist ein Faltungskern, der über eine gaußsche Glockenkurve gebildet wird. Mit solch einem Filter lassen sich Bilder glätten und somit Rauschanteile reduzieren. Bilder wirken dadurch weicher, bzw. verwaschen.

Ein möglicher Faltungskern sähe so aus:

$$\frac{1}{16} \begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}$$

Wendet man diesen Filter auf ein Bild mit der diskreten Faltung an, erhält man dieses Ergebnis:



Abbildung 2.2: Anwendung eines Weichzeichnungsfilters

Quelle: [https://de.wikipedia.org/wiki/Datei:Halftone,\\_Gaussian\\_Blr.jpg](https://de.wikipedia.org/wiki/Datei:Halftone,_Gaussian_Blr.jpg) (10.3.2019)

### 2.3.2 Laplace-Filter

Ein Laplace-Filter ist ein Faltungskern zur Kantendetektion innerhalb eines Bildes. Unter einer Kante versteht man eine rasche Veränderung der Helligkeitswerte entlang einer Richtung.

Um Kanten zu finden wird ein Operator auf das Bild angewendet, der die zweite Ableitung bildet (Laplace-Operator). Abrupte Schwankungen der Intensitätswerte werden dadurch als Nulldurchgänge sichtbar.

Über die Faltung des Operators der Vorwärtsdifferenz (1 -1) mit sich selbst, lässt sich ein 1-dimensionaler Faltungskern der zweiten Ableitung bilden: (1 -2 1). Dieser Kern lässt sich transponieren, um ein Bild nicht nur in x-, sondern auch in y-Richtung abzuleiten. Beide Kerne kombiniert ergeben den Laplace-Filter:

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

Wendet man diesen Filter auf ein Bild an, erhält man folgendes Ergebnis:

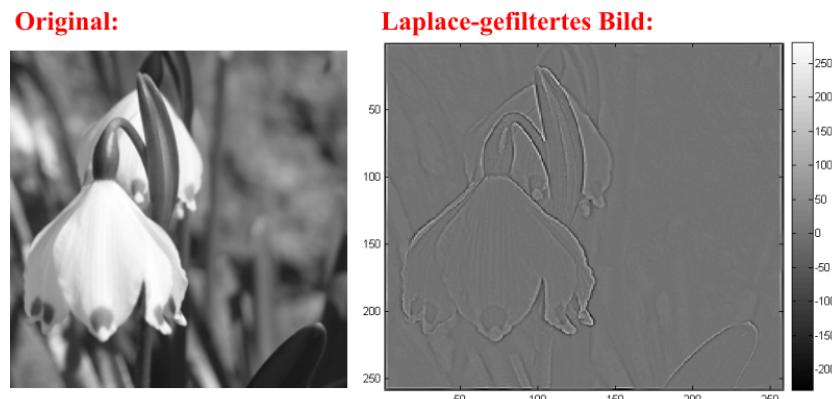


Abbildung 2.3: Anwendung eines Laplace-Filters

Quelle: [https://de.wikipedia.org/wiki/Datei:Laplace\\_beispiel.png](https://de.wikipedia.org/wiki/Datei:Laplace_beispiel.png) (10.3.2019)

Aufgrund eines hohen Rauschanteils in natürlichen Bildern liefert dieser Filter jedoch nicht immer gute Resultate.

### 2.3.3 Sobel-Operator

Um auf natürlichen Bildern Kanten zuverlässig zu erkennen, kombiniert der Sobel-Operator die Ideen des Gauß- und Laplace-Filters. Das Bild wird in eine Richtung über die zentrale Differenz (1 0 -1) abgeleitet, in die andere Richtung jedoch über einen Gauß-Filter (1 2 1) geglättet, um Rauschanteile zu reduzieren. Kombiniert man beide Filter miteinander, erhält man den Sobel-Operator.

$$\begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$$

Dadurch werden Kanten jedoch nur in eine Richtung erkannt. Um Kanten in die jeweils andere Richtung zu erkennen, kann die Faltungsmatrix transponiert werden und ebenfalls auf das Ausgangsbild angewendet werden.

Die beiden erhaltenen Ergebnisse können nun vereint werden, um alle Kanten innerhalb eines Bildes zu erhalten:

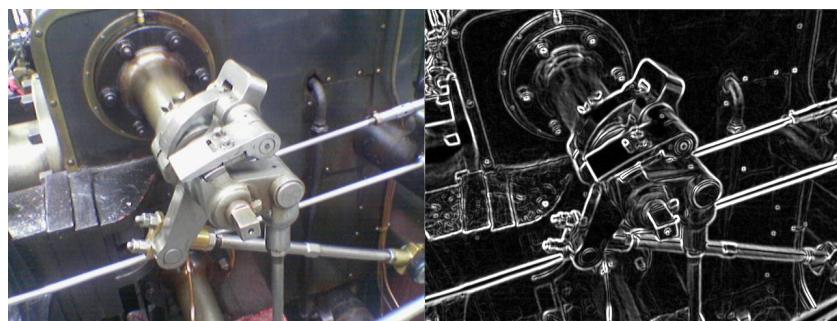


Abbildung 2.4: Anwendung eines Sobel-Operators

Quelle: [https://en.wikipedia.org/wiki/File:Valve\\_sobel\\_\(3\).PNG](https://en.wikipedia.org/wiki/File:Valve_sobel_(3).PNG) (10.3.2019)

## 2.4 Otsu

Das Verfahren von Otsu dient zur Bildsegmentierung. Durch die Findung eines geeigneten Schwellwertes, sollen Grauwerte eines Bildes über das Histogramm in zwei Klassen eingeteilt werden. Die beiden Klassen werden so gewählt, dass die Varianz der Grauwerte innerhalb der Klassen möglichst gering ist, jedoch zwischen den Klassen möglichst groß.

Der Grauwert, der beide Klassen voneinander trennt, ist gleichzeitig der Schwellwert zur Segmentierung des Bildes. Ist der Grauwert eines Pixels unter dem Schwellwert, gehört er zur 1. Klasse, ansonsten zur 2. Klasse. Abhängig davon wie das Bild geartet ist, lässt sich eine Klasse als Hintergrund interpretieren und die jeweils andere beschreibt den Vordergrund, bzw. das gewünschte Objekt.



Abbildung 2.5: Segmentierung eines Bildes mit dem Otsu Verfahren

Quelle: [https://de.wikipedia.org/wiki/Schwellenwertverfahren#Verfahren\\_von\\_Otsu](https://de.wikipedia.org/wiki/Schwellenwertverfahren#Verfahren_von_Otsu)  
(29.3.2019)

Dadurch, dass ein globaler Schwellwert für das gesamte Bild verwendet wird, haben globale Helligkeitseinflüsse wie Licht und Schatten starke Auswirkungen auf das Ergebnis des Verfahrens, wie in Abbildung 2.5 zu erkennen ist. Um solche Störfaktoren zu umgehen, kann, statt eines globalen Schwellwertes auch ein lokaler Schwellwert für jeden Pixel anhand seiner Nachbarschaft individuell bestimmt werden. So kommt es unter Umständen zu genaueren Ergebnissen. Für weitere Literatur, siehe [7].

## 2.5 Haar Kaskaden

Mittels Haar Kaskaden lassen sich relativ schnell Objekte innerhalb eines Bildes identifizieren.

Die Grundlage hierfür bilden *Haar Features* 2.6. Das sind Features, die einen Pixelverlauf als Kante, Linie oder Rechteck einstufen.

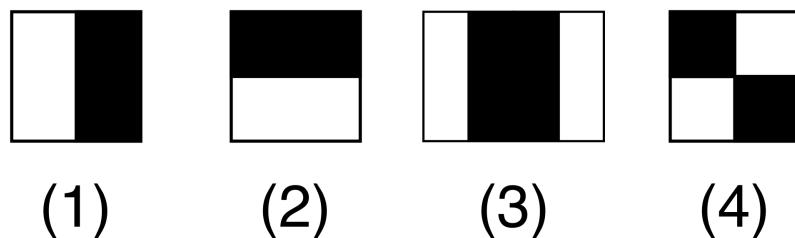


Abbildung 2.6: Haar Features

Quelle: [https://commons.wikimedia.org/wiki/File:VJ\\_featureTypes.svg](https://commons.wikimedia.org/wiki/File:VJ_featureTypes.svg) (30.3.2019)

Diese können ebenfalls als Faltungsmatrix dargestellt werden. Eine bestimmte Menge und Kombination dieser Features beschreibt anschließend das gesuchte Objekt. Sie müssen jedoch im Bild zuerst erkannt werden, was selbst bei einer geringen Auflösung sehr aufwendig werden kann.

Da diese Features auch nur einen geringen Teil des Bildes ausmachen, wird die Erkennung dieser in eine Kaskade unterteilt: Das Bild wird zunächst grob nach den Features durchsucht, und anschließend in den Bereichen die Suche verfeinert, in denen die gewünschten Features vorkommen könnten, sodass eine Kaskade von Suchen durchlaufen wird, bis die Features entweder gefunden oder nicht gefunden wurden und das Bild entsprechend eingestuft werden kann. Für weitere Informationen, siehe [11].

## 2.6 Morphologische Operatoren

Morphologische Operatoren werden in der Bildverarbeitung eingesetzt, um die Form von Strukturen innerhalb eines Bildes zu verändern. Sie können sowohl auf Binär-, als auch auf Grauwertbildern angewendet werden, wobei hier lediglich auf die Anwendung bei Binärbildern eingegangen wird.

Zunächst gibt es einfache Basisoperationen, wie Erosion und Dilatation, die die Basis für weitere, komplexere morphologische Operatoren, wie beispielsweise Opening und Closing, bilden.

### 2.6.1 Erosion

Bei der Erosion werden Strukturen, wie der Name vermuten lässt, erodiert bzw. abgetragen. Man definiert hierfür eine Maske, üblicherweise ein Raster von 3x3 Pixeln, wobei der Kern der Maske auch der Mittelpunkt des Rasters ist.

Für jeden gesetzten Pixel (mit Wert 1) innerhalb des Binärbilds wird nun die Maske aufgelegt. Sind dabei nicht alle anderen Pixel innerhalb der Maske ebenfalls gesetzt, so wird das Pixel im resultierenden Bild auf den Wert 0 gesetzt. Haben jedoch alle Nachbarn eines Pixels innerhalb der Maske ebenfalls den Wert 1, so verändert sich das Pixel nicht.

Formal wird die Erosion folgendermaßen notiert:  $I \ominus M$ . Wobei das Bild  $I$  mit der Maske  $M$  erodiert wird.

Der daraus resultierende Effekt ist, wie in Abbildung 2.7 ersichtlich, dass Ränder von Strukturen innerhalb des Bildes abgetragen werden und diese somit verdünnt werden. Es können aber auch Löcher innerhalb der Strukturen vergrößert werden.

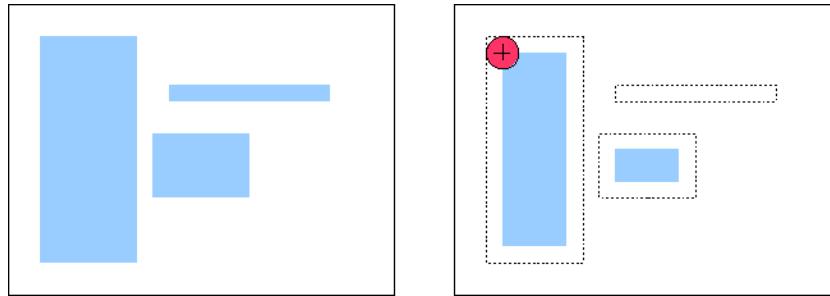


Abbildung 2.7: Anwendung der Erosion auf ein Bild

Quelle: <https://de.wikipedia.org/wiki/Datei:MorphologicalErosion.png> (12.3.2019)

## 2.6.2 Dilatation

Die Dilatation bildet das Gegenstück zur Erosion. Statt die jeweiligen Strukturen abzutragen, lässt man diese wachsen. Dabei kann dieselbe Maske wie bei der Erosion verwendet werden. Diese muss ebenfalls für jeden Pixel auf das Binärbild aufgelegt werden. Ist der Kern der Maske, also das Pixel auf welches die Maske gelegt wurde gesetzt, so werden im resultierenden Bild alle Pixel innerhalb der Maske ebenfalls gesetzt. Hat der Kern jedoch den Wert 0, so verändert sich für dieses Pixel nichts.

Die formale Notation ist  $I \oplus M$ , wobei hier Bild  $I$  mit Maske  $M$  dilatiert wird.

Wie in Abbildung 2.8 erkennbar, wachsen die Strukturen innerhalb des Bildes. Dabei können vorher getrennte Strukturen zusammenwachsen oder Löcher innerhalb einer Struktur geschlossen werden.

## 2.6.3 Opening

Im Gegensatz zu den vorhergehenden Operationen ist das Opening keine Basisoperation mehr, sondern setzt sich aus den Basisoperationen zusammen. Dabei wird ein Bild  $I$  zunächst mit einer Maske  $M$  erodiert und anschließend dilatiert:

$$I \circ M = (I \ominus M) \oplus M$$

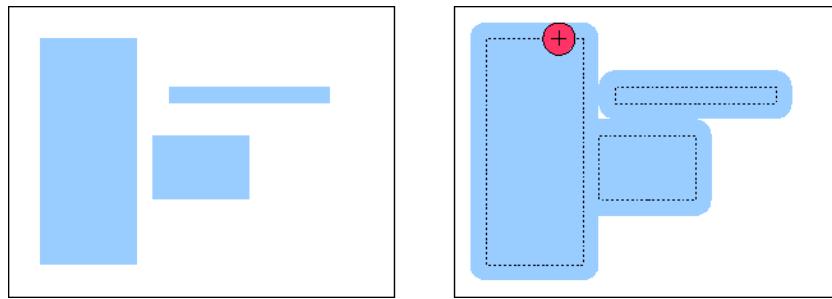


Abbildung 2.8: Anwendung der Dilatation auf ein Bild

Quelle: <https://de.wikipedia.org/wiki/Datei:MorphologicalDilation.png> (12.3.2019)

Die Struktur von hinreichend großen Elementen wird durch diese Operation nicht verändert. Es werden lediglich kleinere Artefakte, welche zumeist durch Störungen oder Rauschen innerhalb des Ursprungsbildes entstanden sind, durch Erosion entfernt. Durch die anschließende Dilatation können diese nicht mehr erzeugt werden. Es können auch Strukturen, die über eine Verbindungsstelle, welche schmäler als die Maske ist, voneinander getrennt werden, sodass aus einer Struktur zwei getrennte entstehen. Ein Beispiel lässt sich in Abbildung 2.9 sehen.

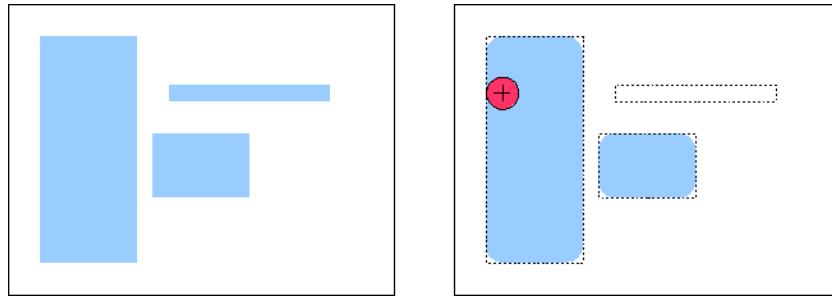


Abbildung 2.9: Anwendung von Opening auf ein Bild

Quelle: <https://de.wikipedia.org/wiki/Datei:MorphologicalOpening.png> (12.3.2019)

## 2.6.4 Closing

Closing bildet das Gegenstück zu Opening. Ein Bild  $I$  wird mit einer Maske  $M$  dilatiert und anschließend erodiert:

$$I \bullet M = (I \oplus M) \ominus M$$

Genutzt werden kann dieser Operator, um beispielsweise, wie der Name erschließen lässt, Löcher in Strukturen zu schließen, oder Verbindungen zwischen vorher getrennten Strukturen aufzubauen. Durch die Dilatation werden Lücken geschlossen, bzw. nahe beieinander liegende Strukturen verbunden. Die Größe der Elemente wird durch die anschließende Erosion wieder zurückgesetzt. Entstandene Verbindungen und geschlossene Lücken bleiben dadurch jedoch erhalten. Ein Beispiel ist in Abbildung 2.10 zu sehen.

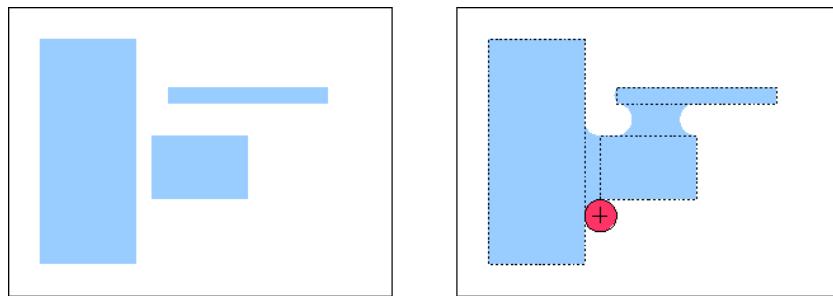


Abbildung 2.10: Anwendung von Closing auf ein Bild

Quelle: <https://de.wikipedia.org/wiki/Datei:MorphologicalClosing.png> (12.3.2019)

## 2.7 OpenCV

OpenCV ist eine Programmierbibliothek, die eine Reihe von Bildverarbeitungsalgorithmen zur Verfügung stellt. Es gibt Implementierungen in C, C++, Java und einigen weiteren Programmiersprachen. Unter anderem gibt es auch eine Bibliothek für Android Geräte.

Es existieren diverse Hilfsmittel zur Segmentierung und Klassifikation von Bildern. Darüber hinaus werden auch morphologische Operatoren bereitgestellt. Durch die Kombination verschiedener Algorithmen und Hilfsmittel hat man als Programmierer die Möglichkeit, sich eine Bildverarbeitungspipeline aufzubauen, um seine Bilder zu verarbeiten. Für mehr Informationen, siehe [12].

# **3 Analyse**

In diesem Kapitel wird die Problemlage im Bezug zum derzeitigen Stand der Technik analysiert. Etablierte Verfahren, die für eine Lösung geeignet wären, werden mithilfe von Beispielen aus der Literatur vorgestellt. Anschließend werden sie mit einem geeigneten Testdatensatz evaluiert. Hierbei werden auch die jeweilige Ressourcenlast und Effizienz gemessen. Das geeignete Verfahren wird für den weiteren Verlauf der Arbeit bestimmt.

## **3.1 Stand der Technik**

Bildverarbeitung ist ein Thema, das schon sehr lange im Bereich der Informations-technik erforscht wird. Besonders durch derzeitige Entwicklungen in der künstlichen Intelligenz und der Objekterkennung bekommt das Thema in der heutigen Zeit eine hohe Bedeutung für die technische Entwicklung. Im folgenden Kapitel werden einige der zum derzeitigen Zeitpunkt etablierten Verfahren der Bildverarbeitung im Kontext der Aufgabenstellung evaluiert und bewertet.

### **3.1.1 Statische Verfahren**

Verfahren, die ohne einen speziellen Kontext direkt auf die Pixelwerte eines Bildes angewendet werden können, sind im Folgenden als „statische Verfahren“ beschrieben.

Sie benötigen keine Vorverarbeitung oder Trainingsdaten, um verwendet zu werden. In den meisten Fällen sind diese Verfahren besonders ressourceneffizient, da sie mit weniger Speicher- und Zeitkomplexität auskommen.

#### **3.1.1.1 Pixelorientierte Bildanalyse**

##### **Beschreibung**

Ein trivialer Ansatz, um Veränderungen zwischen zwei oder mehreren Bildern zu er-

kennen, kann durch einen Vergleich der Pixelwerte ermöglicht werden. So ist es auch möglich benachbarte Pixel in einem Bild nach bestimmten Veränderungen abzusuchen. Bei den möglichen Veränderungen handelt es sich dabei um statistisch messbare Merkmale wie unter anderem auch Intensität, also Helligkeit, sowie die jeweiligen Farbwerte.

Helligkeits- oder Farbwechsel können im Bezug zur Objekterkennung auf einem Bild verwendet werden, indem man zum Beispiel eine Reihe an sogenannten „Hintergrund“-Pixeln wählt und diese mit den benachbarten Objektpixeln vergleicht. Für den Fall der Verkehrskameras würde der besagte Hintergrund die Pixelwerte der Straße in dem gewählten Bereich beschreiben. Sobald eine Reihe von Pixeln durch einen starken Farb- oder Intensitätswertwechsel unterbrochen wird, kann man in diesem Fall von einem beweglichen Objekt auf der Straße ausgehen.

## Literatur

O.K. Rahmat und Jumari beschreiben in ihrer Arbeit [3] ein solches Verfahren und wenden dies auf Bilder von Verkehrskameras an. Die Besonderheit ist, dass nicht alle Pixelwerte des Bildes verglichen werden, sondern nur ein relevanter Teilabschnitt zu Vergleichszwecken ausgesucht wird. Dieser Teilbereich wird von den Autoren auch als „Detektor“ beschrieben und befindet sich in der Mitte einer ausgewählten Fahrbahn. So werden auf Straßen mit mehreren Fahrbahnen auch mehrere solcher Detektoren benötigt.

Ziel ist es, wie bereits beschrieben die Intensität in dem Bereich eines Bildes zu überwachen. Sobald es einen stark abweichenden Wert gibt, wird davon ausgegangen, dass ein Auto den Streckenabschnitt des Detektors durchfährt und ein Zähler wird inkrementiert. Die Länge eines stockenden Verkehrsflusses kann ebenfalls über diese Methode ermittelt werden. Hierfür wird in der Arbeit der beiden Autoren ein länglicher Detektor verwendet, der sich über den kompletten im Bild ersichtlichen Streckenabschnitt zieht. Weiterhin werden die Farbwechsel innerhalb der Detektoren ausgewertet, um die einzelnen Objekte voneinander zu trennen.

## Fazit

Das Verfahren hat eine sehr geringe Komplexität und ist daher einfach zu implementieren. Jedoch werden auch schon in der Arbeit Situationen beschrieben, in denen das Verfahren keine zuverlässigen Ergebnisse liefert. So kann zum Beispiel der Lichtkegel

eines Scheinwerfers bei Nacht das Ergebnis verfälschen.

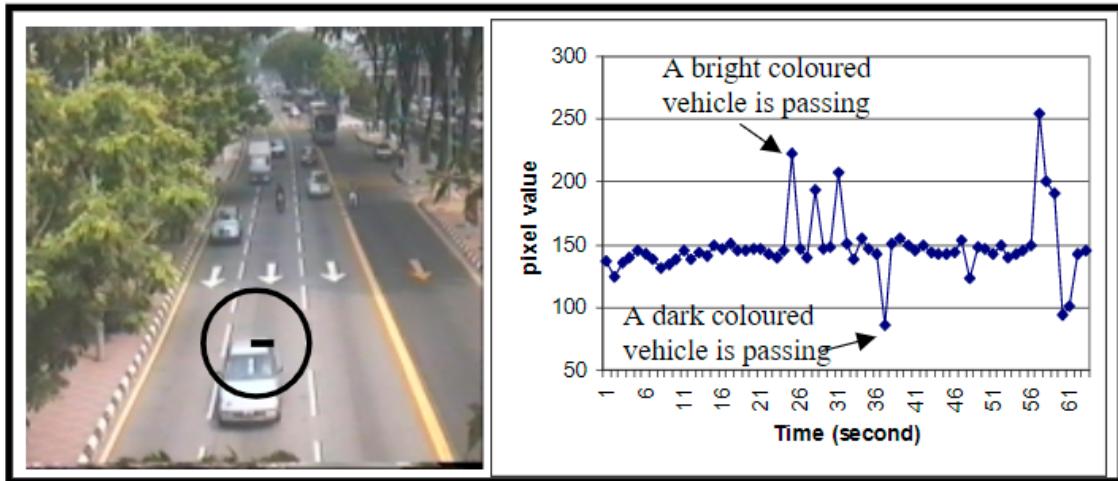


Abbildung 3.1: Detektor im Einsatz

Quelle: [3]

### 3.1.1.2 Kantenerkennung

#### Beschreibung

Kantenerkennung bezeichnet eine Klasse von Verfahren, bei welchen Kanten im Bild über abrupte Intensitätswert-Übergänge gefunden werden. Dabei werden in der Regel Faltungskerne (siehe 2.3) verwendet, um die Ableitung der Intensitätswert-Funktion in einem Grauwert-Bild zu ermitteln.

Für die Anwendung des Verfahrens muss das Bild in den Grauwert-Bereich übertragen werden. Die entstandenen Kanten (Abbildung 3.2) können dann anschließend für die Objekterkennung verwendet werden. Um leicht überlagernde Objektkanten voneinander zu trennen, können morphologische Operatoren verwendet werden (siehe 2.6). Diese werden auch verwendet, um kleine irrelevante Kanten aus dem Bild zu entfernen.

#### Literatur

Gupte und Papanikolopoulos stellen eine exemplarische Implementierung des Verfahrens in ihrer Arbeit [4] vor. Dabei werden Kanten auf zwei Bildern einer Verkehrskamera erkannt und dann beide Kantenbilder per XOR (exklusives Oder) vereinigt. Dies ermöglicht das Erkennen von Veränderungen zwischen den beiden Bildern der Verkehrskamera. Solche können in diesem Fall auf ein bewegliches Objekt, wie zum Beispiel ein Auto, hindeuten.

Das Ergebnis wird anschließend mit Hilfe der Anwendung von mehreren Dilatations-Operationen verbessert, da nicht geschlossene und im Objekt liegende Kanten zur einer großen Kante verbunden werden (Abbildung 3.3). Für den Fall, dass nach der Anwendung der jeweiligen Operationen immer noch unverbundene Kanten vorliegen, werden nah benachbarte Kanten durch ein großes Rechteck umrandet. Das resultierende Rechteck ist in diesem Fall dann das erkannte Objekt.

#### Fazit

Dieses Verfahren ist ebenfalls wenig komplex und lässt sich einfach implementieren. Jedoch besitzt es auch einige Nachteile, wie zum Beispiel, dass sich Schattenwurf negativ auf das Ergebnis auswirkt und eine Perspektive im Bild vorausgesetzt wird. Wettereinflüsse wie Regen oder Schnee würden sich ebenfalls sehr negativ auf das Ergebnis auswirken. Es existieren viele Implementierungen des Verfahrens, wobei die

Arbeit von Gupte und Papanikolopoulos nur ein Beispiel ist. So lassen sich auch durch Vorverarbeitung oder Maskierung der Bilder möglicherweise das Ergebnis verbessern.

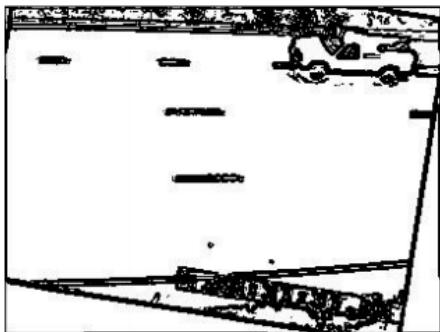


Abbildung 3.2: Kantenerkennung

Quelle: [4]

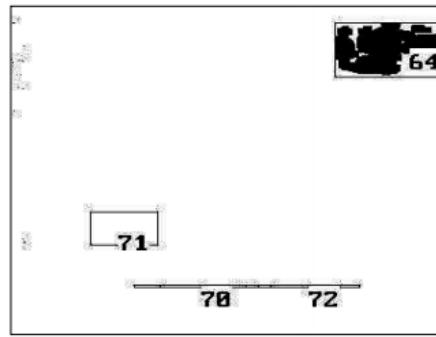


Abbildung 3.3: Ergebnis des Verfahrens

Quelle: [4]

### 3.1.2 Dynamische Verfahren

Als „dynamische Verfahren“ werden im folgenden Verfahren beschrieben, die nur innerhalb eines bestimmten Kontextes auf ein Bild angewendet werden sein. So kann es sein, dass das Verfahren bestimmte Trainingsdaten oder Vergleichswerte von anderen Bildern benötigt. Dies erhöht in den meisten Fällen die Genauigkeit des Verfahrens, aber die Ressourceneffizienz wird drastisch verringert.

#### 3.1.2.1 Neuronale Netze

##### Beschreibung

Neuronale Netze sind ein Beispiel für Verfahren aus dem maschinellen Lernen. Hierbei wird versucht die Arbeitsweise des menschlichen Gehirns nachzuempfinden. Neuronale Netze besitzen daher auch eine Art internen Speicher, der antrainiertes Wissen abbildet. Ein neuronales Netz kann somit für eine bestimmte Aufgabe trainiert werden, welche dann relativ zuverlässig von dem Netz gelöst werden kann. Ziel von dem Training ist hierbei durch Anwendung des Netzes die Fehlerquote zu minimieren. Sobald ein neuronales Netz fertig trainiert wurde, kann der Zustand des Netzes serialisiert und abgespeichert werden, damit es in der Zukunft weiterverwendet werden kann.

##### Literatur

In der Arbeit [5] wurden sogenannte „konvolutionale“ neuronale Netze für die Erkennung von stockendem Verkehr auf Bildern verwendet (siehe Abbildung 3.4). Diese Netze nutzen unter anderem auch Faltungskerne (siehe 2.3) um Objektklassifizierung auf einem Bild durchzuführen.

Diverse Schichten von Neuronen werden in einem der verwendeten neuronalen Netze verwendet, um die aus den Pixelwerten gewonnenen Informationen auf eine konkrete Einschätzung bezüglich des Verkehrsflusses herunterzubrechen. So kann eine Pixelmatrix als Eingabe an das neuronale Netzwerk weitergegeben werden und als Ausgabe ein einfacher boolescher Wert (eins oder null) empfangen werden. Eine Eins wäre beispielsweise eine positive Rückmeldung dafür, dass auf dem Bild stockender Verkehr sichtbar ist.

Wichtig hierbei sind die einzelnen Verbindungen zwischen den Neuronen-Schichten, welche je nach Training und Zustand des Netzes unterschiedlich gewichtet werden.

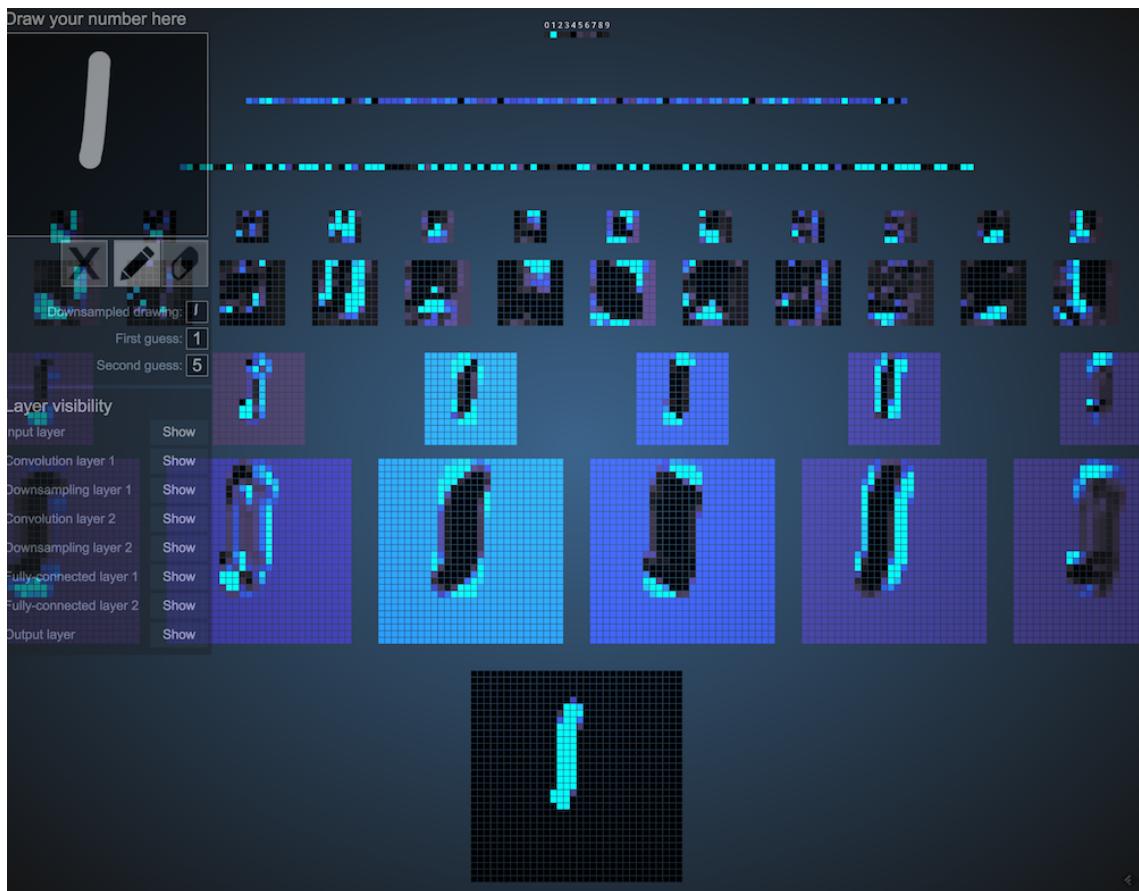


Abbildung 3.4: Visualisierung der Schichten eines konvolutionalen neuronalen Netzwerks (unten Eingang - oben Ausgabe)

Quelle: <http://scs.ryerson.ca/~aharley/vis/> (12.4.2019)

## Fazit

Bei der Laufzeit eines komplexen neuronalen fällt Netzes auf, dass das Laden und Benutzen der Neuronen einen relativ hohen Hauptspeicher-Bedarf mit sich bringt. Der Vorteil, der sich jedoch hierdurch bietet, ist, dass Bilder nicht statisch analysiert werden, sondern das Netz dynamisch auf Bilder und Verhältnisse, wie zum Beispiel Perspektive und Wetter, trainiert wird. Dieses Verfahren wurde für die folgende Implementierung nicht gewählt, da diese Arbeit einen besonderen Fokus auf die Ressourcensparsamkeit des Verfahrens setzt.

### 3.1.2.2 Background-Subtraction

#### Beschreibung

Background-Subtraction, auch als Foreground-Detection bekannt, ist eine Klasse von Verfahren, bei denen einer Serie von Bildern auf Veränderungen analysiert wird. Ziel ist es, sich bewegende Objekte im Vordergrund des Bildes vom Hintergrund zu trennen. Hierfür wird ein sogenanntes Hintergrund-Modell aus den ersten Bildern der Serie generiert, welches dann auf das letzte Bild angewendet wird, um ein Objekt im Vordergrund zu erkennen. Diese verwenden Statistiken, welche über alle Bilder der Serie generiert werden, um Gemeinsamkeiten zwischen den Bildern zu erkennen.

In dem Artikel [6] werden mehrere dieser Hintergrund-Modelle im Detail beschrieben. Bei der Auswahl des geeigneten Hintergrund-Modells sind mehrere Parameter zu berücksichtigen, wie zum Beispiel Perspektive, Farb- und Intensitäts-Spektrum.

#### Literatur

Akoum beschreibt im Artikel [1] wie das Hintergrund-Modell „Gaussian mixture Models“, oft auch als „Mixtures of Gaussians“ bezeichnet, auf ein Video einer Verkehrskamera angewandt werden kann. Bei diesem Hintergrund-Modell werden alle Intensitätswerte der Pixel eines Bildes mithilfe eines Gaußschen Mischmodells modelliert.

Ein Gaußsches Mischmodell ist dabei ein statistisches Clusterverfahren, welches die Hintergrundpixel kategorisiert. Alle Pixel, die nicht über das Mischmodell kategorisiert werden konnten, sind somit Vordergrund-Pixel und somit auch mögliche Objekte im Vordergrund. Objekte, die im Bild durch das Verfahren erkannt wurden, sind im Anschluss weiß eingefärbt, während der Hintergrund schwarz eingefärbt wird. Das Ergebnis des Verfahrens wird mit Dilatations- und Erosions-Filtern verbessert. Der Dilatations-Filter sorgt dabei für die Verbindung von nicht geschlossenen Kanten, während der Erosions-Filter kleine unwichtige Kanten aus dem Bild entfernt.

#### Fazit

Das Verfahren ist relativ komplex, bietet dafür aber eine sehr gute Genauigkeit. Aufgrund des Hintergrund-Modells müssen jedoch mehrere sich unterscheidende Bilder für das Verfahren verwendet werden. Im Falle der Verkehrsanalyse sollte dies kein Problem darstellen, da eine fixierte Kamera bereits gegeben ist, welche jede Minute

ein neues Bild aufnimmt. Parameter, die für das Anwenden des Hintergrund-Modells benötigt werden, müssen im Hauptspeicher vorbehalten werden. Daher hat dieses Verfahren auch nicht die beste Speichereffizienz. Dennoch können durch die jeweiligen Hintergrund-Modelle auch Wetter- und Lichtveränderungen miteinbezogen werden, was potenzielle Fehlerquellen in den statischen Verfahren waren. Weiterhin gibt es bereits einige Implementierungen diverser Hintergrund-Modelle in der Bibliothek OpenCV (siehe 2.7).

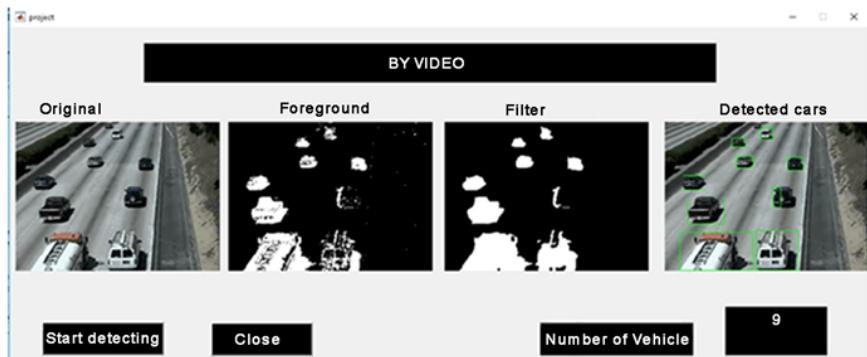


Abbildung 3.5: Background-Subtraction im Einsatz

Quelle: [1]

## 3.2 Bestimmung des Testdatensatzes

### 3.2.1 Verkehrskameras laden

Bevor die Kamerabilder geladen werden können, ist es zunächst relevant zu wissen, welche Kameras es überhaupt gibt und wo diese liegen. Hierfür stellt die Verkehrszentrale eine Liste aller Kameras unter dieser Webadresse bereit: [http://www.svz-bw.de/kamera/kamera\\_A.txt](http://www.svz-bw.de/kamera/kamera_A.txt) (Stand: 31.3.2019).

Diese liegen in einer durch Tabulatorzeichen separierten Tabellenstruktur vor.

lon	lat	title	description	linkextern	icon	iconSize	iconOffset
572330.13	5361095.33	...	...	/.../kameradetail.php?id=EXT006	...	16,16	-8,-8

Tabelle 3.1: Tabellenstruktur der Kamerliste

Die Werte *lat* und *lon* beschreiben die Position der Kamera durch Längen und Breitengrad. Der Eintrag *title* enthält einen ausführlicheren Titel der Kamera. Mit *description* hat man die Möglichkeit eine detailliertere Beschreibung der Kamera zu erhalten, welche jedoch mit HTML Tags angereichert ist. *Linkextern* ist die Webadresse, über welche auf die Kamera zugegriffen werden kann. *Icon*, *iconSize* und *iconOffset* sind Informationen, die das Straßenverkehrsamt selbst benötigt, um das Kameraicon auf einer Landkarte anzuzeigen.

Relevant sind also zunächst die Koordinaten, der Titel und der Link, da nur über diesen die ID der Kamera aufgelöst werden kann. Das erste Problem hierbei ist, dass die Koordinaten der Kamera nicht der üblichen Projektion der Weltkugel entsprechen. Gebräuchlich ist die Projektion *WGS 84* [10], welche auch von Navigationssystemen oder herkömmlichen Kartendiensten genutzt wird und den Globus von -180° westlich bis 180° nördlich, bzw. -90° südlich bis 90° nördlich einteilt.

Das Straßenverkehrsamt nutzt jedoch das Referenzsystem *ETRS89 / UTM zone 32N* [9], welches für Europa ausgelegt ist. Um Position der Kameras mit gängigen Kartendiensten nutzen zu können, müssen die Koordinaten in die übliche Projektionsform überführt werden.

Außerdem muss die ID der Kamera aus dem Link extrahiert werden, da es keinen Eintrag in der Tabelle gibt, der lediglich die ID enthält. Da der Link immer denselben Format folgt (*/<pfad>/kameradetail.php?id=<id>*), kann einfach nach dem Vorkommen der Teilzeichenkette *?id=* gesucht werden, und alles davor abgeschnitten werden.

Somit lassen sich alle relevanten Informationen, welche für die Weiterverarbeitung der Kameras benötigt werden, abrufen.

### 3.2.2 Verkehrsbilder laden

Sobald die ID einer Kamera verfügbar ist, lassen sich auch die Kamerabilder abrufen. Sie werden im Regelfall alle 1-5 Minuten aktualisiert und im JPEG Format auf dem Server der Straßenverkehrszentrale öffentlich gemacht. Es sind daher keine Videosequenzen, sondern nur Einzelbilder verfügbar.

Über die URL `https://www.svz-bw.de/kamera/ftpdata/<id>/<id>_gross.jpg` lässt sich stets die aktuellste Aufnahme für eine Kamera abrufen, wobei `<id>` der ID der Kamera entsprechen muss.

Jedoch verbietet die Straßenverkehrszentrale den Zugriff auf die Bilder von außerhalb ihrer Webpräsenz. Um dennoch darauf zugreifen zu können, kann der *Referer* HTTP-Header mit dem Hostnamen der Verkehrszentrale (`http://svz-bw.de`) beim Abrufen des Bildes mitgesendet werden, wodurch der Zugriff gewährt wird.

### 3.2.3 Vergleichsdatensätze generieren

Bevor verschiedene Verfahren getestet und miteinander verglichen werden können, wird zunächst ein Vergleichsdatensatz benötigt.

Um diesen nun zu generieren wird ein Python-Skript verwendet. Nach dem Start, fängt es an periodisch für einige der Kameras auf der A5 Bilder herunterzuladen.

Neben den Bildern selbst, wird auch eine Metadatei im JSON-Format generiert. Diese enthält zunächst einen Zeitstempel der Bildaufnahme. Außerdem enthält diese auch noch, wie viele Autos jeweils auf der linken und rechten Spur zu sehen sind, und ob anhand dessen Stau ist oder nicht.

Um diese Informationen zu generieren, wird ein sehr prototypischer Algorithmus verwendet. Die Resultate sind dementsprechend nicht gerade akkurat, weshalb ein Mensch diese zusätzlich verifizieren muss. Um jedoch den Menschen dabei zu unterstützen, ist eine grobe Vorauswertung sehr hilfreich.

Um als Mensch die Bilder auswerten zu können, kann ein zweites Skript Abhilfe schaffen.

Dazu gibt es in den Metadaten ein Feld, das speichert, ob ein Bild bereits von einem

Menschen verifiziert wurde. Mit dem Skript werden nun nacheinander alle noch nicht ausgewerteten Bilder angezeigt, wobei abwechselnd die linke und rechte Spur maskiert wird. Zusätzlich sieht man, ob die Vorauswertung für die angezeigte Spur Stau erkannt hat, oder nicht.

Als Mensch hat man nun die Möglichkeit durch das Drücken der Enter-Taste zu speichern, dass Stau auf dem Bild zu sehen ist, bzw. durch das Drücken einer anderen Taste zu bestätigen, dass kein Stau zu sehen ist.

Anschließend wird das nächste Bild bzw. die nächste Spur angezeigt. Dies passiert solange, bis keine Bilder mehr ausgewertet werden müssen, bzw. der Nutzer das Skript beendet.

### **3.3 Auswahl des Verfahrens**

Zur Auswahl des bestmöglichen Verfahrens werden verschiedene Ansätze anhand des Testdatensatzes ausgewertet.

#### **3.3.1 Helligkeit**

Das erste Verfahren klassifiziert ein Bild lediglich anhand der Häufigkeit der Intensitätswerte. Dazu werden zunächst, wie beim Histogramm, die Intensitätswerte zwischen 0 und 255 aufsummiert. Anhand eines Schwellwertes, in unserem Fall der Grauwert 40, wird dann versucht das Bild zu klassifizieren. Gibt es in Summe mehr Grauwerte unter dem Schwellwert als darüber, wird Stau auf dem Bild angenommen. Die Überlegung dahinter ist, dass die Straße einen sehr neutralen bis hellen Grauton aufweist. Fahren nun Autos auf der Fahrbahn, dann können diese in verschiedensten Farben, sowohl über, als auch unter dem Schwellwert auftreten. Jedoch werden von den Autos auch Schatten geworfen, welche die Straße verdunkeln, und so die Intensitätswerte reduzieren.

#### **3.3.2 Haar-Features**

Der Ansatz zur Stauerkennung über Haar-Features verläuft etwas anders als die Erkennung über die Helligkeit. Hierbei wird versucht über Autos zu klassifizieren, um diese anschließend zu zählen und abzuschätzen, ob Stau ist, oder nicht.

Zur Klassifizierung der Autos kann ein von OpenCV mitgelieferter Cascade-Classifier zur Erkennung von Autos genutzt werden. Dieser ist standardmäßig bei OpenCV verfügbar und soll über Haar-Features Autos auf Bildern erkennen können. Solche Kaskaden werden als XML-Dateien mitgeliefert, welche über die Methode *CascadeClassifier* geladen werden können.

Anschließend kann damit ein Bild untersucht werden. Als Ergebnis erhält man eine Liste von Konturen, die Autos auf dem Bild darstellen sollen.

Die Ergebnisse dieses Verfahrens sind jedoch nicht gut, falls die Auflösung der Kameras zu gering sind, da Merkmale von Autos nicht deutlich genug sind, um diese als Autos klassifizieren zu können. Besser arbeiten Classifier auf Bild-Sequenzen, wie z.B. Videos, da diese dabei helfen, Autos über die Sequenzen hinweg zu verfolgen. Da die Straßenverkehrszentrale jedoch nur Einzelbilder liefert, sind die Ergebnisse nicht sehr genau.

### 3.3.3 Edge-Detection

Ein weiteres Verfahren, um Stau zu erkennen, kann mittels des Canny-Algorithmus umgesetzt werden. Der Canny-Algorithmus erlaubt, Kanten innerhalb eines Bildes zu erkennen und verwendet dafür den Sobel-Operator 2.3.3.

Umsetzen lässt sich der Algorithmus über die in OpenCV verfügbare Funktion *Canny*. Dieser wandelt das Bild in ein Graustufenbild um, auf dem die Kanten weiß dargestellt sind und der Hintergrund schwarz.

Anschließend können über OpenCV die Konturen aus dem Bild entnommen werden und diese gezählt werden. Die Zahl der Konturen sollte der Zahl der Autos entsprechen, wobei ein Teil davon den Hintergrund repräsentiert, der jedoch relativ statisch sein sollte, da er sich fast nie ändert.

Aufgrund der stark verrauschten Bilder der Kameras werden jedoch zu viele Kanten erkannt, um verlässlich Fahrzeuge erkennen zu können. Eine vorherige Mittelung des Bildes mit Gauß verbessert das Ergebnis zwar, sorgt jedoch dafür, dass die Kanten nicht als getrennte Konturen ersichtlich werden, um von OpenCV entnommen werden zu können.

### 3.3.4 Background-Subtraction

Das Problem beim Canny-Algorithmus ist, dass der Hintergrund zu viel Einfluss auf das Resultat hat. Daher versucht dieser Ansatz den Hintergrund zu entfernen. Hierfür werden Background-Subtraction Techniken angewendet, die den Hintergrund vom Hauptbild herausrechnen. Übrig bleiben sollten danach lediglich die Autos.

Der Background-Subtraction Algorithmus benötigt jedoch eine Reihe von Trainingsbildern, um zu lernen, was der Hintergrund ist. Daher werden, bevor ein Bild analysiert werden kann, ca. 5-10 Vorlaufbilder benötigt.

Nun gibt es ebenfalls verschiedene Background-Subtraction Algorithmen, die verwendet werden können. Beispielsweise gibt es *GSOC*, *KNN*, *MOG* oder *MOG2*, welche jeweils andere Ansätze verwenden, um den Hintergrund zu entfernen.

*MOG* liefert bei der Bildqualität der Kameras die besten Ergebnisse.

Der Algorithmus wird vor der Analyse also trainiert. Anschließend berechnet er ein Schwarz-Weiß-Bild, wobei weiße Pixel Objekte im Vordergrund darstellen.

Anhand der Konturenerkennung können dann Autos auf dem Bild gezählt werden. Bei Überschreiten des Schwellwerts kann so Stau erkannt werden.

### 3.3.5 Auswahl

Für die Auswahl des Verfahrens werden die einzelnen Algorithmen und Verfahren anhand der Testdatensätze verglichen und bewertet. Von insgesamt 252 Bildern gab es 62, auf denen Stau zu sehen ist und 190 ohne Stau. Die Verfahren werden nun darauf getestet, wie gut sie Stau bei Bildern erkennen, die tatsächlich Stau andeuten (true positives), bzw. wie gut kein Stau auf Bildern ohne Stau erkannt wird (true negatives).

**Vergleichsalgorithmus** Dieser Algorithmus sagt für jedes Bild voraus, dass kein Stau zu sehen ist. Er dient zum Vergleich und soll sicherstellen, dass die Annahme „kein Stau“ nicht besser sein kann, als mit den verwendeten Verfahren die Bilder tatsächlich auszuwerten.

True Positives	True Negatives	Total
0%	100%	76.86%

Tabelle 3.2: Ergebnisse des Vergleichsalgorithmus

In 0% der Fällen „erkennt“ das Verfahren Stau, der existiert, aber in 100% der Fälle wird die Abwesenheit von Stau angenommen. Das bedeutet, dass die anderen Verfahren eine Gesamtgenauigkeit von 76.86% bei diesem Testdatensatz übertreffen müssen, um einsatztauglich zu sein.

**Helligkeitsvergleich** In Bezug auf Ressourceneffizienz ist der Helligkeitsvergleich extrem gut, da sehr wenige primitive Berechnungen durchgeführt werden. Jedoch liefert dieses Verfahren sehr schlechte Resultate. Unterschiedliche Tageszeiten, Lichtverhältnisse oder Autofarben haben einen zu großen Einfluss auf das Ergebnis.

True Positives	True Negatives	Total
20%	48%	41.74%

Tabelle 3.3: Ergebnisse des Helligkeitsvergleichs

In lediglich 20% der Fälle, in denen Stau war, wurde auch tatsächlich Stau erkannt, bzw. nur in 48% der Fälle, in denen kein Stau war, wurde auch kein Stau erkannt. Mit einer Gesamtgenauigkeit von 41.74% ist dieses Verfahren deutlich schlechter als der Vergleichsalgorithmus, weshalb es sich nicht für die weitere Entwicklung eignet.

**Haar-Features** Haar-Features liefen aufgrund der niedrigen Auflösung der Bilder keine verlässliche Erkennung von Autos. Bei einer höheren Auflösung werden Autos zwar gut erkannt, aber nicht auf Bildern der Verkehrszentrale. Zusätzlich ist die Berechnung deutlich aufwendiger, was sich in Bezug auf Ressourcensparsamkeit negativ auswirkt.

True Positives	True Negatives	Total
0%	100%	76.86%

Tabelle 3.4: Ergebnisse der Haar-Features

Zu erkennen ist, dass dieses Verfahren exakt die selben Ergebnisse wie der Vergleichsalgorithmus liefert. Es wird also gar nicht erst Stau erkannt, sondern bei jedem Bild davon ausgegangen, es sei kein Stau. Daher eignet sich dieses Verfahren nicht.

**Kantenerkennung** Bei der Kantenerkennung ist die Ressourcensparsamkeit ziemlich gut erfüllt, da Kantenerkennung auf einfachen Faltungen basiert, welche sich effizient

True Positives	True Negatives	Total
30%	77%	67.36%

Tabelle 3.5: Ergebnisse der Kantenerkennung

berechnen lassen. Das Problem ist jedoch, dass Autos und Hintergrund nicht unterschieden werden können und Autos nicht verlässlich erkannt werden.

Mit einer Gesamtgenauigkeit von 67.36% ist die Kantenerkennung zwar immer noch schlechter als der Vergleichsalgorithmus, liefert jedoch immerhin in 30% der Stau-Fälle eine korrekte Aussage und sogar 77% bei keinem Stau. Insgesamt sind die Ergebnisse jedoch viel zu ungenau, um für die weitere Arbeit relevant zu sein.

**Background-Subtraction** Background-Subtraction benötigt zwar Vorlaufbilder, die zunächst von der Straßenverkehrszentrale nicht verfügbar sind, da lediglich das aktuellste Bild abgerufen werden kann und nicht Bilder aus der Vergangenheit. Jedoch ist der Algorithmus nach Abschluss eines einmaligen Trainings in der Lage jedes weitere Bild effizient zu verarbeiten. Zudem werden Autos verlässlich erkannt.

True Positives	True Negatives	Total
79%	90%	87.6%

Tabelle 3.6: Ergebnisse der Background-Subtraction

Mit einer Gesamtgenauigkeit von 87.6% ist Background-Subtraction das mit Abstand beste Verfahren. Im Gegensatz zum Vergleichsalgorithmus erkennt es zwar nur in 90% der Fälle, dass kein Stau ist, wenn auch wirklich kein Stau ist, dafür erkennt es Stau in 79% der Fälle und liefert ausreichend gute Ergebnisse für die Implementierung der Arbeit.

# 4 Implementierung

Im folgenden Kapitel wird die praktische Implementierung der in der Arbeit erarbeiteten Methode beschrieben. Die Implementierung umfasst dabei die Schnittstelle zum Nutzer, sowie die Erhebung, Verarbeitung und Persistenz von relevanten Daten.

## 4.1 Backend

Für die Implementierung der Arbeit wurde ein Backend konzipiert, welches Informationen und Bilder von Verkehrskameras für Clients vorenhält. Das Ziel ist es dabei, Daten der Straßenverkehrszentrale regelmäßig anzufragen und für den Client zu persistieren. Somit wird ermöglicht, dass der Client auch auf ältere Datensätze der Straßenverkehrszentrale zugreifen kann. Weiterhin ermöglicht das Backend die Daten in einem möglichst kompakten Format an den Client zu senden, damit auch über schwache Netzwerkverbindungen die Daten empfangen können.

### 4.1.1 Infrastruktur

Im Rahmen der Implementierung des Backends wurden diverse Anforderungen an die zu verwendenden Technologien beachtet. So soll die resultierende Anwendung auf dem Apache Server eines geteilten Webhosts laufen. Der Apache Server des Webhosts erlaubt das Ausführen von PHP-Skripten, welche für das Ansteuern von Anfragen und die Verarbeitung von Daten der Straßenverkehrszentrale Baden-Württemberg verwendet werden können.

Für die Persistenz von Daten besitzt der Webhost alternativ zum Dateisystem auch eine MySQL-Datenbank. Die Datenbank ist für die Implementierung besser geeignet als das Dateisystem, da sie Transaktionen sowie eine relationale Abbildung der Datensätze ermöglicht. So ermöglicht die Datenbank auch das einfache Abspeichern von

Metadaten für Verkehrskameras und die Vernetzung zwischen Metadaten und Kamerabildern über Fremdschlüssel.

Ein weiterer Grund warum für den Anwendungsfall die Datenbank dem Dateisystem zu bevorzugen ist, ist, dass Datensätze einfach und schnell geordnet und gefiltert werden können. Dies wird über die SQL-Sprache ermöglicht, die sowohl „SELECT FROM WHERE“-Klauseln sowie Aggregationsfunktionen wie „SUM“ oder „MAX“ für das Filtern von Datensätzen bietet.

Auf Abbildung 4.1 ist zu sehen, dass die Verbindung zum Client über das HTTP-Protokoll verläuft. Der Vorteil ist hierbei, dass das Protokoll bereits von dem Apache Server unterstützt wird und eine Verschlüsselung über TLS eingesetzt werden kann.

Ein weiterer Bestandteil der Backend-Infrastruktur ist der Cron-Daemon des Linux Betriebssystem des Host-Systems. Über diesen lassen sich periodisch wiederholende Aufgaben einplanen, welche für das Backend nützlich sein könnten. Ein Beispiel hierfür ist das regelmäßige Abfragen von Bildern einer Verkehrskamera der Straßenverkehrszentrale.

Die letzte Komponente der Infrastruktur ist die Applikation „cURL“, die sich auf dem Betriebssystem des Hosts befindet. cURL unterstützt die Übertragung von Daten über diverse Protokolle und lässt sich auch über Schnittstellen anderen Programmen und Programmiersprachen kontrollieren. So ermöglicht dies auch das Anstoßen von HTTP-Anfragen über PHP-Skripte des Apache Servers. cURL bietet zusätzlich auch die Funktionalität verschiedene Parameter der Anfragen einzustellen. Einstellbare Parameter umfassen dabei unter anderem:

- HTTP-Header (auch Referer-Header)
- Timeout der Anfrage
- Überprüfung von SSL-Zertifikaten (z.B. bei HTTPS)
- Empfangen von spezifischen Headern (z.B. Last-Modified)

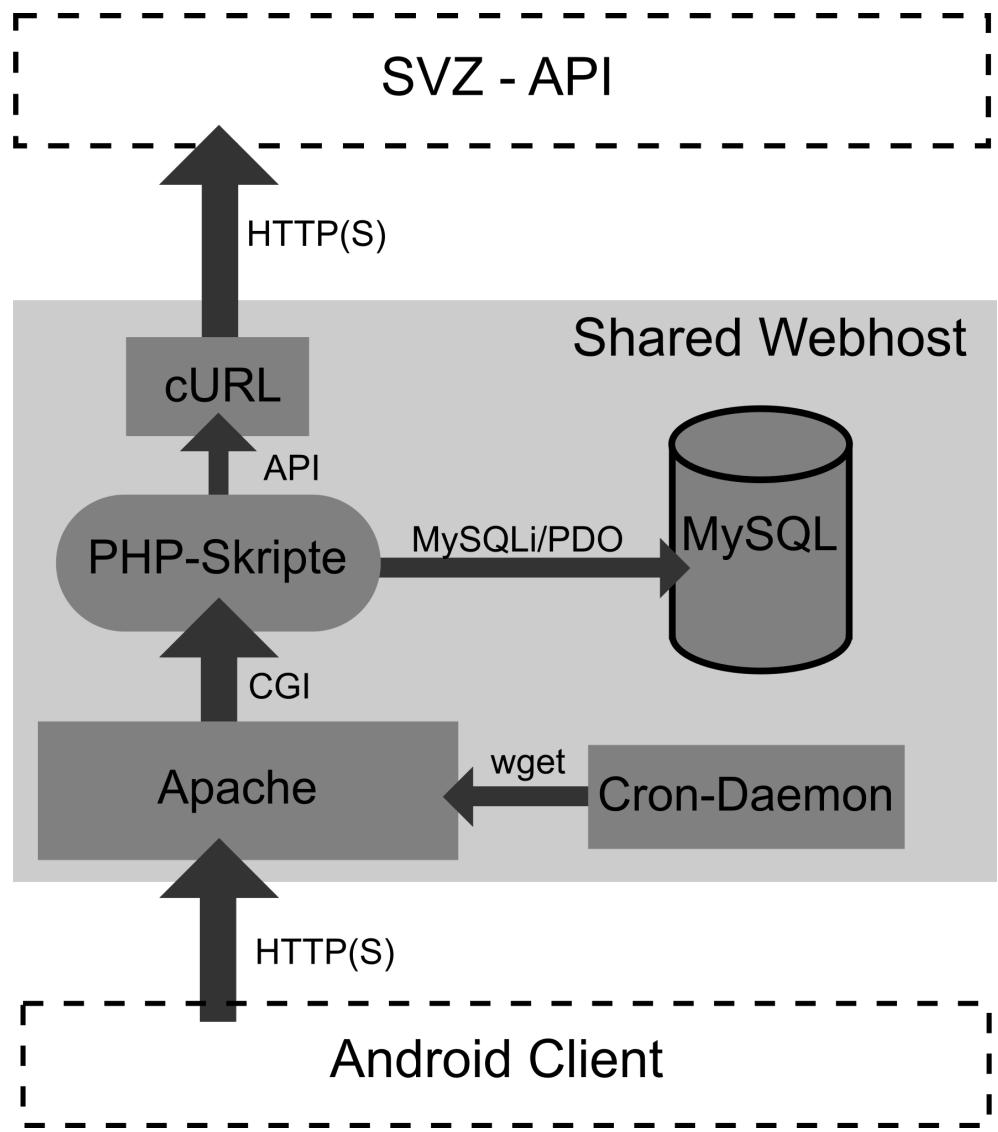


Abbildung 4.1: Server Architektur

### 4.1.2 Datenerhebung

Die Erhebung von benötigten Daten erfolgt durch HTTP-Anfragen an den Server der Straßenverkehrszentrale. Diese können mithilfe der cURL-API in PHP angestoßen werden.

Informationen über die jeweiligen Kameras werden über den SVZ-Server in verschiedenen Formaten bereitgestellt. So sind die Metadaten der Kameras als Textdatei abgespeichert, während die Bilder direkt im JPEG-Format abgerufen werden können. Unter Metadaten der Verkehrskameras sind folgende Informationen zu verstehen:

- Eine eindeutige alphanumerische Kamerakennung
- Die entsprechende Autobahnkennung
- Name der nächsten Ausfahrt mit Autobahnkennung
- Position der Kamera

Die Metadaten der Kameras müssen nur einmalig abgerufen und verarbeitet werden, während die Bilder der Verkehrskameras von der SVZ alle 30 Sekunden erneuert werden. Um immer aktuelle Datensätze zu empfangen, werden Bilder im 30-Sekunden-Takt angefragt. Dies lässt sich über PHP und einen Cronjob im Linux Betriebssystem des geteilten Webhosts realisieren. Ein Cronjob ist dabei eine der in Abschnitt 4.1.1 angesprochenen Aufgaben, die über den Cron-Daemon periodisch ausgeführt werden.

Ein Datensatz, der nicht über HTTP-Aufrufe erhoben werden kann, sind Masken für Bilder einer Verkehrskamera. Dabei handelt es sich um Bilder im PNG-Format, welche zur Vorverarbeitung von Bildern verwendet wird. Diese müssen manuell erstellt und in die Datenbank eingepflegt werden, da im Rahmen der Arbeit kein vollständig korrekter Algorithmus zur Automatisierung dieser Aufgabe gefunden wurde.

### 4.1.3 Datenpersistenz

Für die Persistenz von Daten wurde während der Implementierung auf das Dateisystem verzichtet und nur die MySQL-Datenbank verwendet, da diese die Vorteile einer relationalen Datenbank, sowie sichere Transaktionen bietet. Bei den Datensätzen, die zu persistieren sind, handelt es sich dabei um:

- Die Metadaten einer Verkehrskamera (siehe 4.1.2)
- Die Bilder einer Kamera im JPEG-Format
- Die Masken für eine Kamera im PNG-Format

Jeder dieser Datensätze besitzt in der Datenbank eine eigene Tabelle mit einem jeweils eigenen Datenbank-Schema. Wichtig bei der Erstellung der jeweiligen Schemata war dabei geeignete Datentypen für die Bilder zu finden, damit diese korrekt und vollständig eingepflegt werden können. Das Schema für die Datenbank ist auf Abbildung 4.2 zu sehen (Primärschlüssel ist fett gedruckt). „ABID“ bezeichnet dabei die jeweilige Kennung für die Straße. Ein Beispiel hierfür wäre die Autobahnkennung A5.

Metadaten	
<b>ID</b>	char(6)
<b>ABID</b>	char(10)
Beschreibung	varchar(100)
Geogr. Breitengrad	double(12,2)
Geogr. Längengrad	double(12,2)
Referenzen	int(11)

Bilder	
<b>Insert-Timestamp</b>	datetime
<b>Kamera-ID</b>	char(6)
Bild	mediumblob

Masken	
<b>Ausrichtung</b>	bit(1)
<b>Kamera-ID</b>	char(6)
Bild	mediumblob

Abbildung 4.2: Datenbank Schema

Damit PHP-Skripte auf die MySQL-Datenbank zugreifen können, gibt es zwei unterschiedliche APIs: Die „PDO“-API und die MySQLi-API. PDO steht dabei für

PHP Data-Objects und bietet eine Unterstützung von mehreren Datenbank-Treibern. MySQLi, auch MySQL Improved Extension, begrenzt sich auf die Unterstützung von MySQL, bietet aber auch eine erweiterte Unterstützung neuer Funktionen von MySQL. Da sich die Implementierung in dieser Arbeit auf eine MySQL-Unterstützung beschränkt wurde die letztere Programmierschnittstelle verwendet.

Damit die Datenbank mit Bildern, welche alle 30 Sekunden gespeichert werden, nicht vollläuft, gibt es einen Lösch-Mechanismus, der alte Datensätze automatisch aufräumt. Dieser ist in PHP implementiert und überprüft nach dem Einfügen eines Datensatzes, ob mehr als 10 Datensätze vorhanden sind. Ist dies der Fall, werden die zehn jüngsten Datensätze behalten und der Rest gelöscht.

#### 4.1.4 Datenbereitstellung

Daten lassen sich vom Client, wie auf Abbildung 4.1 zu sehen, über HTTP-Anfragen an PHP-Skripte abrufen. Es gibt hierbei unterschiedliche Skripte, die für jeweils unterschiedliche Datensätze und Darstellungsarten verantwortlich sind:

- `kameras.php`: Zuständig für das Abrufen aller erreichbaren Kameras mit deren Kennung
- `fetchmask.php`: Zuständig für die Abfrage einer Maske für eine Verkehrskamera
- `fetcher.php`: Zuständig für Abfragen von Bildern einer Verkehrskamera

Das Skript „`kameras.php`“ gibt bei einer Anfrage eine mit Leerzeichen separierte Liste aller Kennungen von Kameras zurück, welche eine Maske besitzen und einen Referenzzähler größer eins haben. Der Referenzzähler ist dabei eine Hilfsvariable für das Backend, welche Kameras von Clients benötigt werden und daher die Bilder von ihnen heruntergeladen werden sollten. Ein Client kann für eine oder mehrere Kameras den Referenzzähler inkrementieren, indem er das „`refcounter.php`“ Skript mit den jeweiligen PHP-Skripten aufruft.

Das Skript „`fetchmask.php`“ wird mit zwei Query-Paramteren aufgerufen: der Kamerakennung und der Ausrichtung, wobei Letzteres ein Binärwert ist. Es liefert, wie der Name bereits verrät, das PNG-Bild einer Maske für eine Kamera.

Das letzte Skript „`fetcher.php`“ wird mit nur einem Query-Parameter für die Kamerakennung aufgerufen. Dieses Skript liefert einen Binärstrom aller gespeicherten Bilder einer Verkehrskamera, sowie den jeweiligen Zeitpunkt (in Unix-Zeit), wann dieses Bild auf dem SVZ-Server gespeichert wurde. Damit diese Daten korrekt vom Client gelesen werden können, werden die Daten in folgender Reihenfolge gesendet: Die Länge in Bytes des Zeitpunktes und des Bildes zusammen, der Zeitpunkt und Unix-Zeit und abschließend die Bytes des zusendenden JPEG-Bildes.

## 4.2 Client

Der Client wird als eine Android Anwendung realisiert, die durch Kommunikation mit dem Backend und der Straßenverkehrszentrale das Problem der Stauerkennung so effizient und ressourcensparend wie möglich löst. Durch das Verlagern der Berechnungen auf das mobile Endgerät, bleibt die Last des Backends gering und die Rechenleistung des Smartphones wird soweit ausgeschöpft, dass Berechnungen effizient, aber sparsam durchgeführt werden können.

### 4.2.1 OpenCV

Der Kern der Verarbeitung wird durch Algorithmen der OpenCV Bibliothek 2.7 umgesetzt. Da OpenCV primär als native Bibliothek zur Verfügung steht, bzw. die für Android nutzbare Java-Schnittstelle auf die native (in C++ geschriebene) Implementierung zurückgreift, muss OpenCV für die jeweiligen Prozessoren der Android Geräte kompiliert werden, auf denen die Anwendung genutzt werden soll.

Eigentlich bietet OpenCV selbst bereits vorkompilierte Versionen der Bibliothek für alle gängigen CPUs und Android Geräte an, jedoch sind diese nicht vollständig. OpenCV besteht nämlich eigentlich aus zwei teilen: der Kernbibliothek, also OpenCV selbst, aber zusätzlich noch einem Erweiterungsmodul namens *OpenCV-Contrib*.

Diese Erweiterung bietet zusätzlich zu den Standardalgorithmen des Kerns Erweiterungen und diverse andere Algorithmen an, welche Randprobleme abdecken und bei der regulären Nutzung der Bibliothek nicht benötigt werden.

Da für die Umsetzung der Anwendung ein spezieller Background-Subtraction Algorithmus benötigt wird, welcher nur in dem Erweiterungsmodul zur Verfügung steht, wird dieses hierfür benötigt.

Die vorkompilierte Version von OpenCV beinhaltet jedoch nicht die Erweiterungen, sodass diese nicht nutzbar ist.

Um die Bibliothek jedoch nicht selbst für jede Architektur kompilieren zu müssen, gibt es die Möglichkeit, die Bibliothek von dritten zu beziehen. Entwickler unter dem Namen *QuickBird Studios* stellen auf der Plattform *GitHub* (<https://github.com/quickbirdstudios/opencv-android>) eine aktuelle Version von OpenCV mit allen Erweiterungen zur Verfügung, welche ganz einfach in die App eingebunden und genutzt werden kann.

### 4.2.2 Backend Kommunikation

Da das Backend in PHP geschrieben ist, verläuft die Kommunikation über das HTTP-Protokoll an den entsprechenden Webserver. Android stellt bei der Kommunikation über HTTP jedoch Ansprüche. Die Kommunikation muss auf dem Gerät asynchron geschehen. Das bedeutet, dass im Haupt-Thread, also dem Programmfad, welcher für das Darstellen der Oberfläche dient, keinerlei Netzwerkkommunikation geschehen darf. Dies könnte unter Umständen dazu führen, dass die Oberfläche einfriert, da, solange auf das Ende der Kommunikation gewartet wird, die Oberfläche nicht neu dargestellt werden kann.

Um sicherzustellen, dass Kommunikation asynchron verläuft, wird in der Klasse *Downloader* jegliche HTTP-Kommunikation gekapselt. Hierzu wird innerhalb der Klasse ein neuer Thread gestartet, welcher für die Kommunikation zuständig ist. Vor dem Starten des Threads kann ein Callback-Objekt übergeben werden, welches nach Abschluss der Kommunikation die erhaltenen Daten übergeben bekommt und aufgerufen wird.

Ebenfalls stellt der *Downloader* sicher, dass pro Instanz der Klasse nur eine Anfrage zur gleichen Zeit abgehandelt werden darf. Dieses Verhalten ist für das spätere Laden der Bilder relevant. Werden dennoch mehrere Verbindungen gleichzeitig benötigt, können einfach mehrere Instanzen der Klasse erstellt werden.

Für die Kommunikation mit dem Backend, ist der *BackendConnector* verantwortlich. Über den Konstruktor kann die URL des Backends übergeben werden. Mit Hilfe der *Downloader*-Klasse stellt dieser die Endpunkte des Backends als Methoden zur Verfügung:

- *cameras*: liefert eine Liste aller Unterstützten Kameras zurück
- *fetch*: liefert entweder alle gespeicherten Bilder einer Kamera, oder alle Bilder jünger als ein gewünschter Zeitpunkt
- *mask*: liefert die Maske für eine Kamera in eine gewünschte Fahrtrichtung

Die Ergebnisse aller Anfragen liegen jedoch im Rohformat als Byte-Array vor. Um die Ergebnisse also nutzen zu können, ist eine weitere Verarbeitung der Daten in einem späteren Schritt nötig.

### 4.2.3 Geolokalisierung

Um gezielt Kameras abzufragen und auszuwerten, ist es sinnvoll nur jene auszuwerten, die in unmittelbarer Nähe des Nutzers sind und ebenfalls Relevanz für die Fahrt haben. Hierfür muss demnach die Position des Nutzers ermittelt und ausgewertet werden.

Heutige Smartphones besitzen in der Regel ein GPS-Modul, mit welchem sich die Position bestimmen lässt. Bevor man jedoch die Möglichkeit hat, mit dem Modul kommunizieren zu können, muss eine Anwendung die erforderlichen Berechtigungen einfordern. Dafür muss zunächst in der Manifestdatei der Android-Anwendung die Berechtigung *android.permission.ACCESS\_FINE\_LOCATION* hinzugefügt werden. Bei neueren Android-Versionen reicht dies allein jedoch nicht mehr aus. Zusätzlich dazu muss der Nutzer explizit bestätigen, dass die Anwendung Zugriff auf den Standort erhält.

Durch Übergabe des Wertes *Manifest.permission.ACCESS\_FINE\_LOCATION* an die Funktion *requestPermission* der *Activity*-Klasse, die den Einstiegspunkt der Android-Anwendung bietet, wird dem Nutzer ein Dialog angezeigt, durch den er Zugriff erteilen oder verweigern kann.

Sind alle Berechtigungen erteilt, kann die App den Standort abfragen. Hierfür ist in der App die Klasse *GlobalPositionManager* verantwortlich. Diese kapselt die Interaktion mit dem GPS-Modul und kann über an einen übergebenen Callback Standortänderungen regelmäßig nach außen kommunizieren. Hierzu wird zunächst beim *LocationManager*, dem Systemdienst für die Standortverwaltung des Gerätes, die Lokalisierung angefragt, was dafür sorgt, dass alle 10 Sekunden der aktuelle Standort an den *GlobalPositionManager* übergeben wird. Dieser kann anschließend an den Callback übergeben werden und für die weitere Verarbeitung genutzt werden.

### 4.2.4 Richtungsfeststellung

Hat man nun die Position zur Hand, so ist als Nächstes relevant zu wissen, wohin gefahren wird, um die zu analysierende Straßenseite zu erkennen. Hierfür gibt es diverse Ansätze, um das Problem zu lösen.

Beispielsweise stellt Google einen Dienst zur Verfügung, über den bei Übergabe von verschiedenen Positionen die befahrene Straße und Richtung erkannt werden kann. Dieser ist jedoch kostenpflichtig und deshalb nicht geeignet.

Um auch hier den Fokus auf Ressourcensparsamkeit nicht zu verlieren, wurde ein

sehr einfacher, jedoch nicht generischer Ansatz gewählt. Da die Anwendung primär für eine Nutzung auf der A5 gedacht ist, ist das Ziel zunächst nur die Fahrtrichtung auf der A5 zu erkennen. Auf anderen Autobahnen kann daher nicht ohne Weiteres erkannt werden, auf welcher Straßenseite gefahren wird.

Die A5 erstreckt sich von Niederaula, nördlich von Frankfurt am Main bis nach Weil am Rhein, kurz vor der Schweizer Grenze. Man kann sagen sie erstreckt sich ungefähr von Frankfurt nach Basel.

Zur Erkennung der Fahrtrichtung, können die Standorte über die Zeit protokolliert und kumuliert werden, um daraus einen Richtungsvektor zu bilden. Dieser kann genutzt werden, um abzuschätzen, ob sich der Nutzer Frankfurt oder Basel nähert.

Dazu werden die vom *GlobalPositionManager* erfassten Positionen durch die Klasse *PositionHistory* protokolliert. Über die bereitgestellten Methoden *getStart* und *getEnd* werden die Positionen so aufsummiert, dass Startpunkt und Endpunkt ausgelesen werden können.

Anschließend ermittelt die Klasse *DirectionCalculator* die euklidische Norm zwischen sowohl Frankfurt, als auch Basel und den Start- und Endpunkten. Diese vernachlässigt zwar die Krümmung der Erdoberfläche und jegliche Höhenunterschiede, ist aber für solch kurze Distanzen völlig ausreichend.

Ist der Startpunkt nun weiter entfernt von Frankfurt als der Endpunkt, so bewegt sich der Nutzer nach Frankfurt, ansonsten in Richtung Basel.

Anfangs ist das Feststellen der Richtung noch recht ungenau. Je mehr Positionen jedoch über die Zeit protokolliert werden, desto genauer wird die Ermittlung der Fahrtrichtung.

Obwohl diese Form der Richtungsfeststellung nur auf der A5 funktioniert, ist die Berechnung sehr effizient und akkurat, da nur wenige einfache Rechenschritte benötigt werden, um somit der Ressourcensparsamkeit gerecht zu werden.

#### 4.2.5 Verkehrskameras laden

Bevor die Informationen zu den Verkehrskameras von der Straßenverkehrszentrale geladen werden können, muss zunächst geprüft werden, welche Kameras das Backend unterstützt.

Hierfür kann über die Klasse *AvailableCamerasLoader*, wie im Abschnitt 4.2.2 beschrieben, über den *BackendConnector* mit der Methode *cameras* die Liste der unter-

stützten Kameras abgerufen werden. Die IDs der Kameras zeilenweise zurückgeliefert. Der *AvailableCamerasLoader* verarbeitet die Zeilen anschließend, sodass eine Liste der verfügbaren Kamera IDs zur Verfügung steht.

Zusätzlich werden noch die detaillierten Kamerainformationen der Straßenverkehrszentrale benötigt. Dafür kann über den *CameraLoader*, wie im Abschnitt 3.2.1, die Liste der Kameras mit dem *Downloader* empfangen werden und verarbeitet werden. Hierfür wird das Tabellenformat aufgeschlüsselt und die benötigten Werte wie ID, Name, Beschreibung und Standort der Kamera in einem *Camera*-Modell abgelegt werden.

Da der Standort jedoch ein unterschiedliches Koordinatensystem verwendet, muss dieser jedoch noch konvertiert werden. Hierfür kann über die Bibliothek Proj4J [8] eine Konvertierung durchgeführt werden.

Diese empfangene Liste wird nun anhand den vom Backend unterstützten Kameras gefiltert, sodass nur noch die verfügbar sind, mit denen auch gearbeitet werden kann.

Schließlich wird beim Aktualisieren der Position des Nutzers, bzw. auch der Fahrtrichtung, geprüft, welche Kameras für die Weiterfahrt relevant sind. Das bedeutet: Kameras, die nicht auf der Strecke liegen, werden erst gar nicht benötigt. Liegen Kameras zwar auf der Strecke, aber hinter dem Nutzer, also ist der Nutzer bereits an ihnen vorbei gefahren, sind diese ebenfalls nicht mehr relevant. Lediglich die Kameras, die vor dem Nutzer auf der Strecke sind, werden für die Stauanalyse benötigt. Es wird also nach jeder Positionsänderung geprüft, welche Kameras relevant sind. Diese werden dann markiert, sodass die spätere Bildanalyse weiß, welche Kameras analysiert werden sollen.

#### 4.2.6 Bilder laden

Das Laden der Bilder gliedert sich in 3 Hierarchiestufen. Auf der untersten Ebene kann über die *fetch*-Methode des *BackendConnectors* ein Binärstream an Bilddaten vom Backend für eine Kamera geladen werden. Da das Backend innerhalb dieses Streams alle Bilder liefert, die es zwischen gepuffert hat, müssen diese zunächst voneinander getrennt werden.

Eine Hierarchieebene darüber regelt der *MultiImageLoader* die Verarbeitung. Nach Erhalt der Rohdaten des Backends wird der Binärstrom aufgespalten. Jedes Teilbild innerhalb des Stroms hat einen Kopf, welcher aus 4 Bytes für die Länge des Bildes

und 8 Bytes für den Aufnahmezeitpunkt des Bildes besteht. Nach den Kopfdaten folgt das eigentliche Bild. Der gesamte Strom besteht nun aus einer Folge von Kopf und Bild. Wobei zu beachten ist, dass die Bytereihenfolge des Kopfes *Little-Endian* ist, und nicht *Big-Endian* wie auf den meisten Android Geräten verwendet.

Es wird also die Länge des Bildes gelesen und konvertiert, danach der Zeitpunkt gelesen und konvertiert und anschließend das Bild über die vorher eingelesene Länge extrahiert. Dieser Vorgang wiederholt sich solange, bis idealerweise keine Bytes mehr im Strom sind, oder im Fehlerfall die aus dem Kopf entnommene Länge des Bildes länger ist, als übrige Bytes im Strom vorhanden sind.

Das Backend liefert standardmäßig jedoch alle Bilder für eine Kamera die es zwischengespeichert hat. Um also nicht bei jeder Anfrage alle Bilder alten Bilder erneut empfangen und verarbeiten zu müssen, wird in der obersten Hierarchiestufe vom *CameraImageFetcher* das strukturierte Laden der Bilder koordiniert.

Für jede der relevanten Kameras existiert eine Instanz, die von außerhalb alle 29 Sekunden aufgefordert wird, neue Bilder abzurufen. Das sorgt, dafür, dass mindestens 2 mal innerhalb der Aktualisierungsperiode des Straßenverkehrszentrums auf neue Bilder geprüft wird, um möglichst schnell auf Stau reagieren zu können und die Latenz des Ladens und Verarbeitens der Bilder zu kompensieren.

Die vom *MultiImageLoader* empfangenen Bilder werden also zwischengespeichert. Bei einem neuen Abfrageintervall kann über das Durchreichen des Zeitstempels des letzten empfangenen Bildes an den *BackendConnector* das Backend dazu instruiert werden, lediglich die Bilder zu senden, die jünger als der besagte Zeitstempel sind.

#### 4.2.7 Bilder verarbeiten

Die Verarbeitung der Bilder wird von der Klasse *EvaluationExecutor* verwaltet. Diese hört sowohl auf Änderungen der relevanten Kameras, als auch auf Änderungen der Fahrtrichtung.

Für jede Kamera wird zunächst eine passende Maske für die entsprechende Fahrtrichtung vom Backend geladen. Hierfür kann über den *BackendConnector* mit der *mask* die entsprechende Maske angefordert werden.

Diese dient bei der Verarbeitung dazu, die irrelevanten Teile des Bildes, wie beispielsweise Fahrbahnräder oder die Gegenfahrbahn zu verdecken, um das Ergebnis zu optimieren.

Ebenfalls wird der *EvaluationExecutor* benachrichtigt, sobald neue Bilder vom Backend empfangen wurden. Um diese nun zu verarbeiten, gibt es für jede Kamera eine Instanz der *ImageEvaluator*-Klasse, welche für die eigentliche Verarbeitung der empfangenen Bilder zuständig ist.

Die eigentliche Verarbeitung der Bilder gliedert sich in 5 Teilschritte.



Abbildung 4.3: Zu verarbeitendes Beispielbild

**Vorverarbeitung** Bei der Vorverarbeitung wird ein empfangenes Bild so aufbereitet, dass die nachfolgende Verarbeitung bessere Ergebnisse liefert.

Hierfür wird das Bild zunächst gemittelt, um mögliche Rauschanteile zu reduzieren. Mit der von OpenCV bereitgestellten Methode *Imgproc.blur* lässt sich ein Bild weichzeichnen.

Anschließend wird auf das weichgezeichnete Bild die geladene Maske aufgetragen.

Wichtig ist, dass die Maske nach der Mittelung aufgetragen wird und nicht vorher, da sonst die Maske negative Auswirkungen auf die Qualität des gemittelten Bildes haben kann.

**Verarbeitung** Bei der Verarbeitung gilt es, die Autos vom Hintergrund zu trennen. Dazu wird der von OpenCV bereitgestellte Background-Subtractor namens *MOG* verwendet. Dieser liefert auf ein übergebenes Bild ein Binärbild zurück, bei welchem schwarze Pixel den Hintergrund symbolisieren und weiße Pixel die Objekte im Vordergrund, welche idealerweise Autos sein sollten.

Damit dieser Algorithmus jedoch verlässliche Ergebnisse liefert, muss dieser zunächst trainiert werden. Aus diesem Grund liefert das Backend nicht nur das aktu-



Abbildung 4.4: Ergebnis nach der Vorverarbeitung

ellste Bild, sondern eine Sequenz von zeitlich eng beieinanderliegenden Bildern, mit welchen der Background-Subtractor trainiert werden kann.

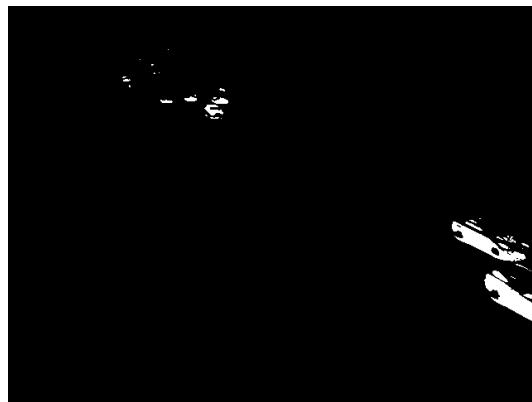


Abbildung 4.5: Ergebnis nach der Verarbeitung

**Nachbearbeitung** Dieser Schritt und alle Nachfolgenden sind lediglich für die Bilder relevant, die auch ausgewertet werden sollen, und nicht nur zum Trainieren des Background-Subtractors dienen.

Bei der Nachbearbeitung wird versucht, jegliche Artefakte oder Ungenauigkeiten, die bei der Verarbeitung des Bildes entstanden sind, zu entfernen. Beispielsweise werden Autos nicht immer als geschlossene Konturen dargestellt oder es entstehen Verbindungen zwischen Autos und größeren Transportern, die parallel zueinander fahren, wodurch diese als eine einzige Kontur erkannt werden könnten. Darüber hinaus kann es vorkommen, dass vereinzelt weiße Pixel auftauchen, welche vermutlich durch Rau-

schen, Schatten, oder andere Bedingungen fälschlicherweise als Objekte eingestuft wurden.

Um diese Artefakte zu entfernen, kann eine Kombination von morphologischen Operatoren angewendet werden.

Zunächst wird versucht über *Closing* die Konturen der Autos zu schließen, bzw. getrennte Teile eines Autos zu vereinen. Anschließend sollen über *Opening* kleine weiße Pixel und alleinstehende Artefakte entfernt werden.

Zuletzt wird mit einer *Dilatation* versucht, die Konturen der Autos zu verkleinern, um mögliche Verbindungen oder Fehlinterpretationen von Verbindungen zu Transportern zu verhindern.

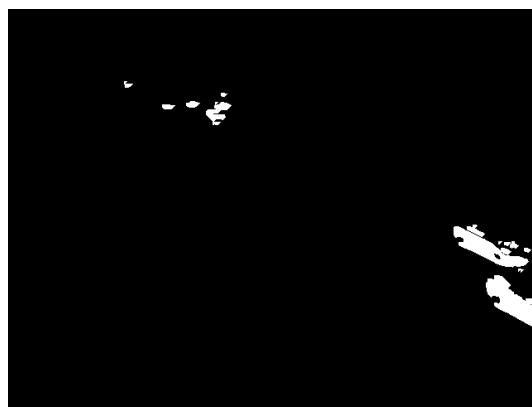


Abbildung 4.6: Ergebnis nach der Nachbearbeitung

**Analyse** Bei der Analyse wird versucht, Konturen zu erkennen und diese zu vereinen. Über die Methode *Imgproc.findContours* von OpenCV, lassen sich Konturzüge aus einem Binärbild auslesen. Zur effizienteren Verarbeitung wird der Polygonkonturzug jedoch auf einfache Rechtecke reduziert, indem der Minimale und Maximale Punkt der Kontur in sowohl X-, als auch Y-Richtung zur Bildung des Rechtecks gewählt wird, wodurch eine Hülle um den Zug gebildet wird.

Da es trotz Nachbearbeitung vorkommen kann, dass Konturen innerhalb von Konturen erkannt werden, oder einige sich überlappen, wird versucht Konturen zu vereinen.

Dabei werden Konturen innerhalb einer anderen Kontur vollständig entfernt. Überlappen sich zwei Konturen, so werden diese zu einer großen Kontur vereint.

Dieser Prozess wird solange iterativ durchgeführt, bis sich keine Kontur mehr verändert.

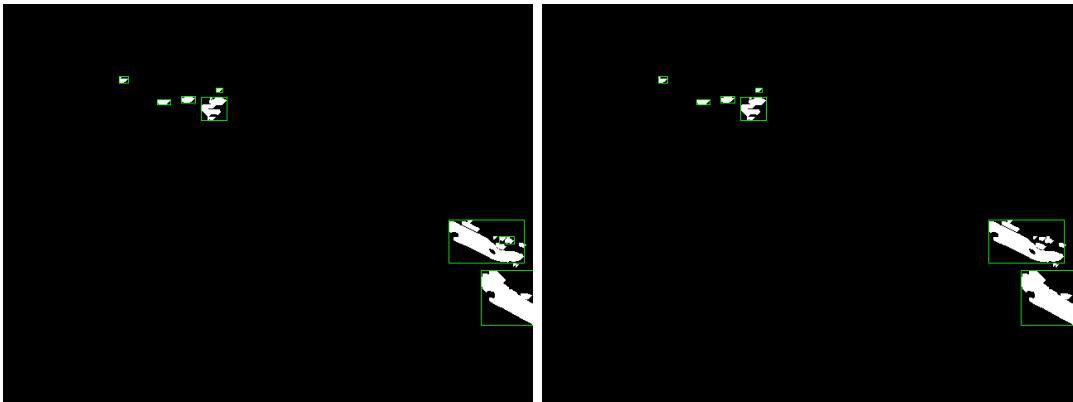


Abbildung 4.7: Konturen erkennen und schließen

**Klassifikation** Bei der Klassifikation wird nun versucht anhand der Konturen eine Aussage über die Verkehrssituation zu treffen. Hierbei wird jede erkannte Kontur als Auto betrachtet. Um nun einzuschätzen, ob Stau ist oder nicht, wird der Anzahl der Konturen mit einem vordefinierten Limit für die jeweilige Kamera verglichen. Hierbei wird jedoch nicht wie bei einem Thresholding-Verfahren ab überschreiten der Grenze Stau erkannt, sondern versucht, ab einer Anzahl an Konturen größer als die Hälfte des Limits eine prozentuale Aussage über die Stauwahrscheinlichkeit zu treffen.

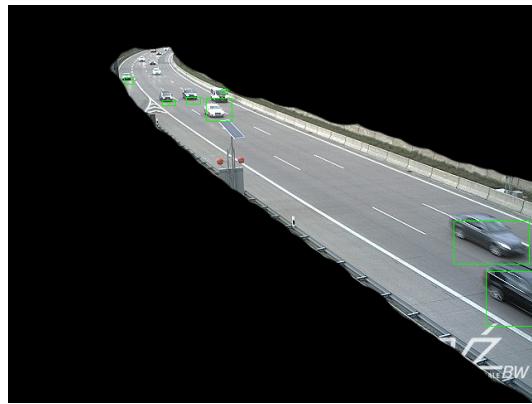


Abbildung 4.8: Vollständig verarbeitetes Bild

Liegt das Limit einer Kamera bei beispielsweise 40 Autos und es wurden 30 Autos erkannt, so ist die Hälfte des Limits (20) überschritten. Es wird also versucht eine Aussage über die Wahrscheinlichkeit zu treffen. Da 30 Autos genau in der Mitte zwischen der Hälfte und dem gesamten Limit liegen, wird eine Stauwahrscheinlichkeit von 50% angenommen.

## 4.2.8 Bluetooth

Ein Wunsch war es, dass sich die Anwendung automatisch startet, sobald mit dem Auto gefahren wird. Hierfür muss demnach erkannt werden, wann ein Nutzer im Auto sitzt.

Der beste Ansatz hierfür ist es, zu erkennen, wann das Handy eine Bluetooth-Verbindung zur Freisprechanlage des Autos aufgenommen hat, um daraufhin die Anwendung zu starten.

Diese Aufgabe übernimmt der *BluetoothDeviceChecker*. Zunächst ermöglicht er die mit dem Gerät verbundenen Bluetooth-Geräte auszulesen.

Darüber hinaus können Bluetooth-Geräte gespeichert, ausgelesen und gelöscht werden, die einen Start der App bei einer Verbindung auslösen sollen. Dazu werden die MAC Adressen (Media-Access-Control) der Geräte als JSON codierte Liste auf dem Speicher des Smartphones abgelegt.

Ebenfalls gibt es die Möglichkeit, lediglich zu prüfen, ob das Smartphone zurzeit mit einem Gerät gekoppelt ist, welches den Start der App auslösen soll.

Um nun vom System benachrichtigt zu werden, wenn ein neues Bluetooth-Gerät verbunden wurde, ohne dass die Anwendung bereits läuft, gibt es in Android die Möglichkeit, einen Service zu registrieren. Dieser kann dem System mitteilen, welche Ereignisse für ihn relevant sind, sodass das System den Service benachrichtigt, sobald ein solches Ereignis eingetreten ist.

Für den Start der Stauanalyse wird also ein Service mit dem Namen *BluetoothService* registriert, welcher auf Events, bzw. Intents des Typs *android.bluetooth.device.action.ACL\_CONNECTED* hören. Dieses Ereignis wird ausgelöst, wenn eine neue Bluetooth-Verbindung aufgebaut wurde.

Innerhalb des *BluetoothService* kann nun bei eintreffen der Benachrichtigung mit dem *BluetoothDeviceChecker* geprüft werden, ob das Gerät den Start der App auslösen soll. Ist das der Fall, so kann zum Start der Anwendung ein neuer Intent ausgelöst werden. Über diesen Intent kann dem System mitgeteilt werden, dass die Anwendung gestartet werden soll, falls sie nicht bereits gestartet wurde.

## 4.2.9 Benutzeroberfläche

Obwohl die Anwendung hauptsächlich akustisch mit dem Nutzer kommuniziert, wird eine grafische Oberfläche benötigt. Dennoch ist das Aussehen und die Funktionalität

der Oberfläche sehr schlicht, da diese nur eine sekundäre Rolle spielt.

Untergliedert wird die Anwendung dabei in 3 verschiedene Tabs, durch die navigiert werden kann:

**Map** Die erste Ansicht ist eine Landkarte, die auf dem freien Kartendienst OpenStreetMaps basiert. Diese soll sowohl den Standort des Nutzers anzeigen, als auch alle geladenen Kameras anzeigen, sodass der Nutzer eine Übersicht hat, welche Kameras wo positioniert sind.



Abbildung 4.9: Kartenansicht der Anwendung

**Details** Die Detailseite zeigt dem Nutzer Details zum Analyseprozess. Zum einen lässt sich in einer Debugansicht sehen, welche Schritte des Analyseprozesses durchlaufen wurden (beispielsweise Feststellung des Standorts, Feststellung der Richtung, Laden der Kameras, Analyse der Bilder, usw.). Darüber hinaus hat der Nutzer auch die Möglichkeit, zu jeder analysierten Kamera das zuletzt ausgewertete Bild zu sehen, auf welchem die erkannten Autos markiert wurden. Anhand dessen kann der Nutzer dann selbst noch einmal einschätzen, ob die Aussage der Anwendung zur Verkehrslage korrekt ist, oder nicht.

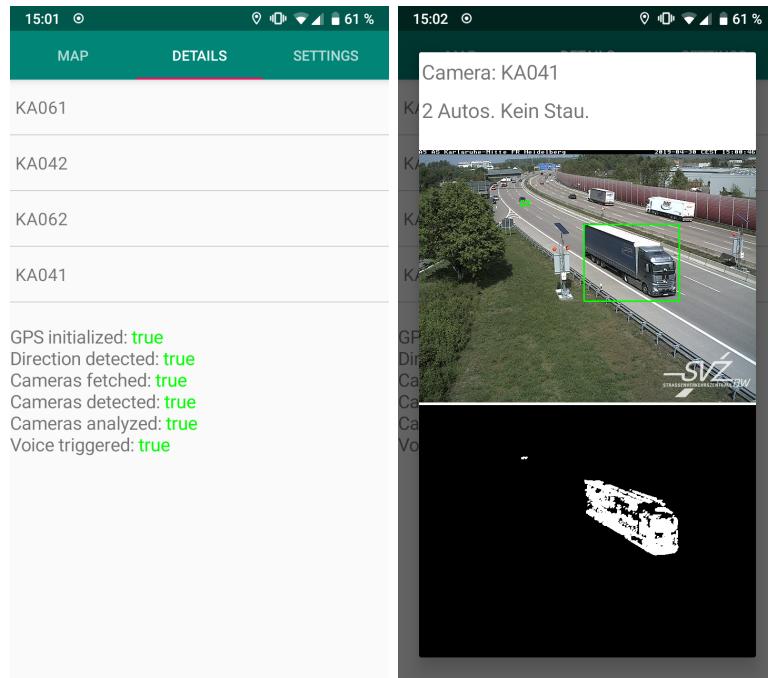


Abbildung 4.10: Detailansicht der Anwendung

**Settings** Die Einstellungsseite dient zur Konfiguration der Bluetooth-Geräte. Hierbei sieht der Nutzer eine Liste aller mit dem Handy gepaarten Geräte und hat durch Anklicken dieser, die Möglichkeit, das Gerät zu speichern, sodass die Kopplung den Start der App auslöst.

Ebenfalls gibt es eine zweite Liste, über die der Nutzer Geräte wieder entfernen kann, sodass diese den Start der App nicht mehr auslösen.

#### 4.2.10 Stauansage

Die Ansage des Staus untergliedert sich technisch in zwei Teile. Im ersten Teil wird die grundsätzliche Textansage realisiert und zusätzlich die Integration einer Bluetooth-Freisprechanlage ermöglicht. Der zweite Teil übernimmt die kontextbezogene Ansage des Verkehrszustandes.

Um überhaupt Informationen per Sprachausgabe übermitteln zu können, wird die interne Android *Text-To-Speech*-API verwendet. Die Klasse *Speaker* übernimmt die Kapselung der Schnittstelle.

Nach außen exponiert die Klasse lediglich eine Methode *speak*, die bei Übergabe

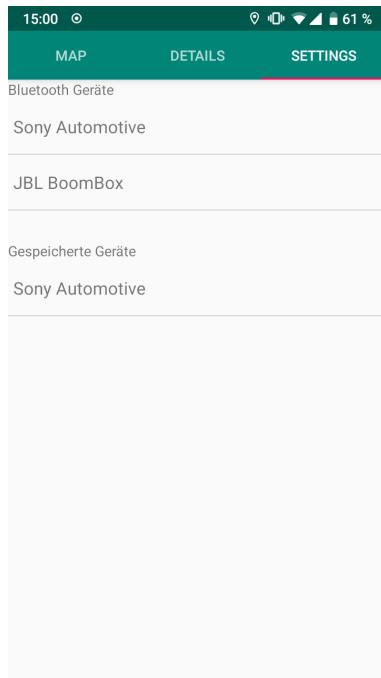


Abbildung 4.11: Einstellungen der Anwendung

von Text eine Sprachausgabe auslösen soll.

Da die interne *Text-To-Speech*-API eine unbestimmte Zeit zur Initialisierung benötigt, können Nachrichten nicht direkt ausgegeben werden, sondern werden zunächst in einer Warteschlange gesammelt, bis die Initialisierung abgeschlossen ist.

Um die Sprachausgabe auch über eine Freisprechanlage in einem Auto zu ermöglichen, wird vor der Übergabe einer Nachricht an das System geprüft, ob das Gerät mit einem Auto gekoppelt ist. Hierzu kann der *BluetoothDeviceChecker* genutzt werden.

Ist ein kompatibles Gerät gekoppelt, übermittelt der *Speaker* an das System, dass der Bluetooth-SCO-Modus für die Audiomeldung aktiviert werden soll. Das sorgt dafür, dass Töne nicht mehr über reguläre Lautsprecher ausgegeben werden sollen, sondern über ein Bluetooth-Headset.

Nun gilt es den Verkehrszustand über die Sprachausgabe zu übermitteln. Optimal wäre es, vor jeder möglichen Ausfahrt anzusagen, ob Stau ist oder nicht, um noch rechtzeitig abfahren zu können. Da das Erkennen der Ausfahrten mit rechtzeitiger Ansage an die Funktionalität eines eigenständigen Navigationssystems grenzt, übersteigt dieser Ansatz den Rahmen der Arbeit. Stattdessen wird Stau für jede Kamera individuell angesagt. Ist jedoch kein Stau auf den Kameras erkennbar, soll einmalig

angesagt werden, dass kein Stau erkannt wurde. Da die Kameras jedoch zu unterschiedlichen Zeitpunkten ausgewertet werden, wird zunächst, nach Auswertung einer Kamera, geprüft, ob für die jeweilige Kamera bereits eine Ansage getätigter wurde, sofern auch Stau erkannt wurde. Falls nicht, wird der Name und der Standort der Kamera mit der ausgerechneten Stauwahrscheinlichkeit an den *Speaker übergeben und angesagt*. Liegt für jede relevante Kamera eine Auswertung vor, wobei bei keiner Kamera ein Stau erkannt wurde, so wird eine Ansage an das System übergeben, die dem Nutzer vermittelt, dass kein Stau erkannt wurde.

Ändern sich nun die relevanten Kamereras, weil der Nutzer beispielsweise an einigen vorbeigefahren ist, so wird erkannter Stau bei einer neuen Kamera zwar angesagt, jedoch nicht, wenn bei der Kamera bereits ein Stau erkannt wurde. Wurde nach Aktualisierung der Kamereras immer noch kein Stau erkannt, so wird der Nutzer nicht erneut informiert.

# 5 Bewertung

In diesem Kapitel wird das im vorherigen Kapitel beschriebene System beurteilt und mit dem vorhergehenden System verglichen. Aufgrund der sich unterscheidenden Infrastrukturen der beiden Systeme, ist es nicht möglich einen direkten Vergleich durchzuführen. Stattdessen müssen verschiedene Kennzahlen bezüglich der zu erfüllenden Anforderungen gefunden werden, um einen Vergleich zwischen den Kennzahlen des jeweiligen Systems durchzuführen.

## 5.1 Abweichungen der Systeme

Das in dieser Arbeit beschriebene System wurde besonders im Hinblick auf Ressourceneffizienz optimiert. Somit gibt es auch starke Unterscheidungen zum alten System, welches lediglich ergebnisorientiert entworfen wurde und daher einen erhöhten Resourcenverbrauch besitzt.

Um weniger Ressourcen zu verbrauchen, wurde zum einen auch die Auswertung von Verkehrskamerabildern vereinfacht. Statt der Verwendung eines neuronalen Netzes, welches einen sehr hohen Ressourcenbedarf mit sich zieht, wird nun ein weniger komplexer Background-Subtraction Algorithmus verwendet.

Die Auswertung wird dadurch so stark vereinigt, dass sie sich auch auf die Clients auslagern lässt. Das Backend kann also als einfacher Zwischenspeicher für Daten der Straßenverkehrszentrale Baden-Württemberg verwendet werden und ist somit auch für den Nutzen auf einem Shared-Webhost optimiert.

Durch die Auslagerung der Auswertung auf die Clients ist jedoch auch nun der Großteil der Berechnung auf dem Client zu finden. Es ist daher wichtig, bestimmte Kenngrößen zu finden, um das eher client-lastige neue System mit dem server-lastigen Altsystem zu vergleichen.

## 5.2 Kennzahlen

Wichtige Kennzahlen für den Vergleich zwischen Neu- und Altsystem sollten vor allem die Anforderungen an das System widerspiegeln. So ist es sinnvoll, auch die während der Laufzeit benötigten Ressourcen zu messen, sowie die Genauigkeit des Systems zu bestimmen. Während der Arbeit wurden folgende Kennzahlen zum Vergleich der Systeme bestimmt:

- Arbeitsspeicher-Last des Systems
- CPU-Last des jeweiligen Servers
- Energieverbrauch des Clients
- Genauigkeit des Systems

Wobei die Arbeitsspeicher-Last des gesamten Systems jeweils am Client und am Server gemessen werden muss.

Diese Arbeit befasst sich aufgrund der Aufgabenstellung hauptsächlich mit der Optimierung der Arbeitsspeicher-Last. Diese Ressource ist dabei auch eine der Ressourcen, welche kritisch für die Funktionalität des Systems sind. Wenn das System an die Arbeitsspeichergrenze kommen würde, könnte dies bedeuten, dass die Applikation frühzeitig beendet wird oder erst gar nicht starten kann. Wohingegen Energieverbrauch und CPU-Last meist nicht an konkrete Limitierungen gebunden sind.

**Nicht berücksichtigte Kennzahlen** Da Sekundärspeicher in der heutigen Zeit relativ günstig ist und das System keinen hohen Gebrauch des Speichers macht, wurde dieser nicht für den Vergleich gemessen. Ebenfalls wurde die Auswirkung von mehreren Clients auf die Ressourcen-Last nicht berücksichtigt, da das System in der Zielumgebung keine großen Nutzermengen unterstützen muss.

## 5.3 Datenerhebung

Die beschriebenen Kennzahlen müssen nun zur Laufzeit im jeweiligen System gemessen werden. Hierzu stehen verschiedene Methoden und Programme zur Verfügung, welche das Messen der jeweiligen Kenngröße ermöglichen. Im folgenden wird die für die Arbeit angemessene Herangehensweise beschrieben, um die jeweilige Kennzahl während der Laufzeit zu messen.

### **Arbeitsspeicher-Last des Systems**

Die Arbeitsspeicher-Last des Systems ist an zwei Stellen zu messen: am Client und am Server. Für die Messung der Arbeitsspeicher-Last am Server wurde zur Laufzeit das Programm „htop“ verwendet. htop ist dabei ein Programm das auf dem Linux Betriebssystem installiert werden kann und eine Übersicht der derzeit laufende Prozesse bietet. Zusätzlich werden auch alle belegten Systemressourcen für die jeweiligen Prozesse angegeben.

### **CPU-Last des Servers**

Die CPU-Last des Servers lässt sich ebenfalls über das Programm htop ermitteln. Da die CPU-Last während der Laufzeit jedoch stark schwankt, macht es mehr Sinn hierfür einen Mittelwert über die Laufzeit zu erstellen.

### **Energieverbrauch des Clients**

Der Energieverbrauch des Clients lässt sich über die Profiler-Ansicht von Android Studio ermitteln. Der sogenannte „Energy Profiler“ überwacht dabei zusätzlich zur CPU auch andere Peripheriegeräte, wie zum Beispiel den eingebauten GPS Sensor in dem jeweiligen Smartphone. Weiterhin werden auch System-Events, wie Alarme, Jobs oder Ortungsanfragen analysiert, welche ebenfalls den Energieverbrauch einer App beeinflussen können.

## 5.4 Auswertung

### 5.4.1 Backend

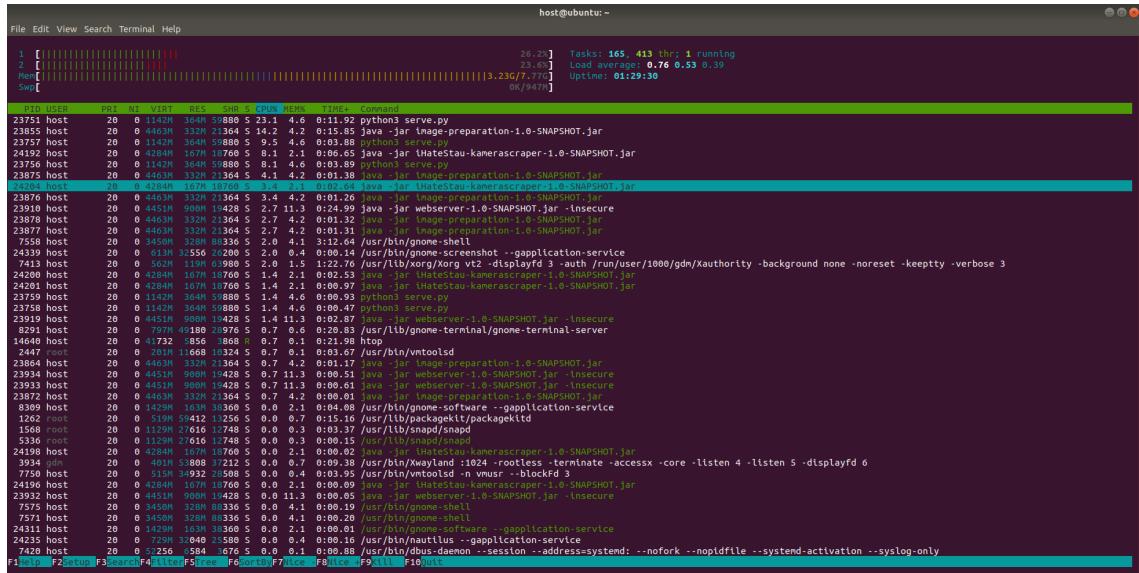


Abbildung 5.1: Ressourcenverbrauch alter Server in htop

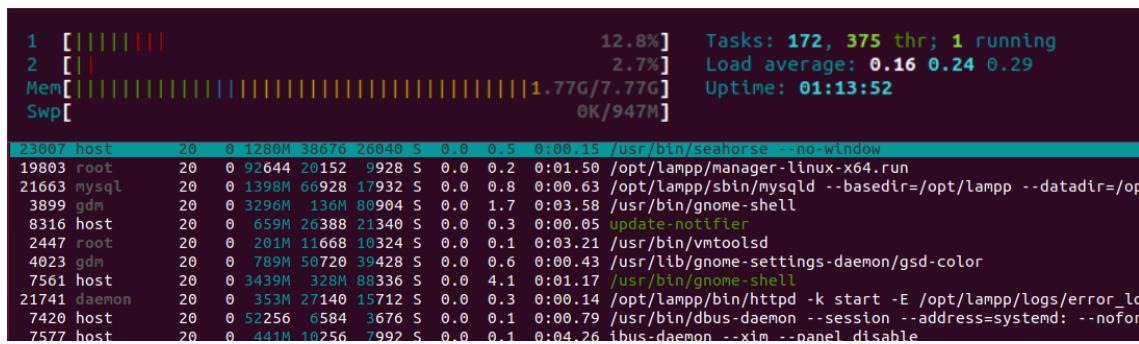


Abbildung 5.2: Ressourcenverbrauch neuer Server in htop

Auf den Abbildungen 5.1 und 5.2 wird der Arbeitsspeicher- und CPU-Nutzungsbedarf des alten und neuen Backends visualisiert.

Bei dem Anblick der Abbildung wird auch klar, dass das alte System eher auf ein Anwendungsszenario im Großbetrieb ausgelegt war, da es sehr starken Gebrauch von Nebenläufigkeit und mehreren Threads macht. Dies ist für ein Szenario in dem sehr viele Benutzer unterstützt werden sollen durchaus sinnvoll. Jedoch bedeutet dies

auch, dass das System einen hohen initialen Bedarf an Ressourcen hat, der sich pro Benutzer auch noch steigert. In der Abbildung 5.2 sieht man daher nur wenige Prozesse und keine Threads, die explizit von dem System verwendet werden. Somit ist der Ressourcenverbrauch beim Starten des Systems stark reduziert.

Weiterhin werden auch weniger Ressourcen pro Benutzer benötigt. Was hier jedoch ganz klar ein Nachteil im neuen System darstellt, ist, dass es nicht mehr für große Nutzermengen ausgelegt ist. Es nutzt keine Skalierungsmaßnahmen wie Threads und Nebenläufigkeit aus und kann daher leicht bei großen Nutzerzahlen an die Systemgrenzen stoßen.

Diese Arbeit konzentriert sich auf den Betrieb des Backends für eine sehr kleine Nutzermenge, daher ist dieser Nachteil für das Ergebnis unerheblich.

#### 5.4.2 Frontend

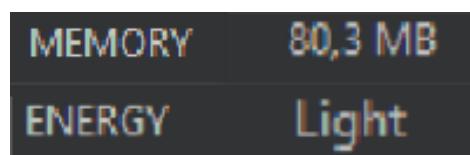


Abbildung 5.3: Ressourcenverbrauch alter Client

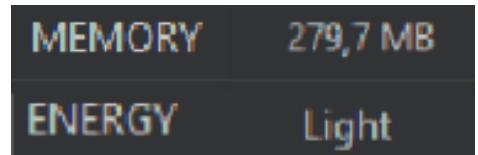


Abbildung 5.4: Ressourcenverbrauch neuer Client

Abbildung 5.4 visualisiert einen Anstieg des Arbeitsspeicher-Verbrauchs im Vergleich zum alten Client, dessen Kennzahlen auf Abbildung 5.3 zu sehen sind. Dabei handelt es sich fast um eine Vervierfachung des ursprünglichen Arbeitsspeicherbedarfs. Dies lässt sich natürlich vor allem durch die client-seitige Auswertung von Bildern erklären, da der hierfür benötigte Algorithmus mehrere Bilder im Arbeitsspeicher behält (siehe Abschnitt 3.1.2.2). Jedoch befindet sich der Arbeitsspeicherbedarf, selbst beim neuen Client, immer noch unter einem halben Gigabyte.

Mit den derzeitigen Hardware-Standards für Smartphones lässt sich dies gut vereinbaren, da aktuelle Smartphones bereits über mehrere Gigabytes an Arbeitsspeicher besitzen. So lassen sich auch andere Applikationen parallel zur Verwendung des Clients nutzen. Die Android Applikation „YouTube“ benötigt beispielsweise ebenfalls etwa einen halben Gigabyte an Arbeitsspeicher bei Benutzung.

Der Energiebedarf des neuen Clients ist laut dem Energy Profiler von Android Studio im Vergleich zum alten System gleich geblieben. Dies lässt sich durch die wenigen Anfragen erklären, die der Client im Vergleich zum alten System tätigen muss. Da der Client in einer Anfrage an das Backend direkt alle nötigen Bilder gesendet bekommt, muss er nicht einzeln anfragen. Weiterhin verwendet der neue Client keine Google-APIs mehr, welche auch zusätzlichen Energiebedarf mit sich bringen.

Der gezeigte Wert für den Energieverbrauch ist jedoch nur gemittelt und als Rangmerkmal beschrieben, so kann es sein, dass einzelne Ausreißer im Energieverbrauch der Applikation existieren. Um diese Ausreißer festzustellen, müsste jedoch eine konkrete Analyse der Applikation auf dem Betriebssystem des Smartphones stattfinden, welche im Rahmen dieser Arbeit übersprungen wurde.

# **6 Zusammenfassung**

## **6.1 Fazit**

Das im Rahmen der Arbeit entwickelte System erfüllt die gestellten Anforderungen an die Ressourceneffizienz. Dennoch bedeutet eine Neuentwicklung auch Veränderungen im Vergleich zum alten System, welche nicht zwingend die Benutzbarkeit des Systems fördern.

Ein Beispiel hierfür ist der neue optimierte Algorithmus zur Bildauswertung, welcher durchaus weniger Ressourcen verbraucht, aber auch etwas ungenauer als der alte Algorithmus ist. So musste während der Auswahl des Algorithmus auch ein Ausgleich zwischen Effizienz und Genauigkeit geschaffen werden. Weiterhin kann das System durch den optimierten Algorithmus auch weniger dynamisch auf Veränderungen des Wetters und der Umgebung reagieren.

Der Vorteil ist, dass die Implementierung des Algorithmus auch auf den Betrieb im zukünftigen Host-System ausgerichtet ist. So soll das Backend des Systems zukünftig auch auf einem geteilten Webhost der Firma 1&1 laufen können. Wie bereits in Abschnitt 4.1.1 angesprochen, erfüllt die Infrastruktur des neuen Systems die Anforderungen für den Betrieb in einem geteilten Webhost.

Schlussendlich gab es auch Veränderungen am Frontend des Systems. So wurde die Benutzerschnittstelle vereinfacht und es gibt eine Audio-Ausgabe über Bluetooth, welche eine Unterstützung von Freisprechanlagen im Auto bietet. Benutzer des Systems können somit einfach die App starten und sich mit der Freisprechanlage des Autos verbinden, um die komplette Funktionalität des Systems ausnutzen zu können.

## 6.2 Ausblick

Die Implementierung des neuen Systems bietet weiterhin Freiraum für neue Verbesserungen und Features, welche während der Arbeit nicht implementiert werden konnten.

So könnte man auch die Genauigkeit des Systems durch ein optimiertes Background-Modell für den Background-Subtraction Algorithmus verbessern. Eine weitere Möglichkeit, die Genauigkeit des Systems zu verbessern, bietet sich durch das Einfügen weiterer Vor- beziehungsweise Nachbereitungsschritten während der Auswertung von Verkehrskamera-Bildern. Zusätzlich könnte der Schwellwert für die Trennung zwischen stockendem und fließendem Verkehr durch einen Algorithmus dynamisch bestimmt werden.

Die Straßenverkehrszentrale bietet nicht immer eine zuverlässige Auskunft für Verkehrskamerabilder. Es könnte daher auch vorkommen, dass die Server für Kamerabilder der Straßenverkehrszentrale für mehrere Tage nicht erreichbar sind. Um auch in Zukunft eine Ausfallfreiheit des Systems zu garantieren, würde es daher Sinn machen, eine alternative Datenquelle zu finden.

Außerdem arbeitet die Anwendung nur auf der A5. Eine Weiterentwicklung des Systems auf alle durch Verkehrskameras überwachten Straßen wäre auch möglich, aber deutlich ressourcenintensiver.

Zusätzlich lässt sich die Benutzerfreundlichkeit des Frontends verbessern, indem man Verbesserungen wie die Erkennung der Entfernung zu Ausfahrten einführen würde. Weiterhin würde ein Knopf auf der Benutzeroberfläche, welcher die letzte Audio-Ausgabe wiederholt, die Benutzerfreundlichkeit fördern.

Schlussendlich muss die Implementierung des System erstmals ausgiebig auf dem geteilten Webhost getestet werden, bevor neue Verbesserungen eingeführt werden sollten. Somit wird sichergestellt, dass das System auch weiterhin alle Anforderungen erfüllt und korrekt ist.

Abschließend lässt sich sagen, dass die App die Anforderungen erfüllt und alle Aspekte beachtet, was aber nicht bedeutet, dass das Projekt dadurch abgeschlossen ist. Auch zukünftig können einige Teile des Systems noch verbessert und erweitert werden, sodass möglicherweise in ein Leben ohne Stau möglich ist.

# Literaturverzeichnis

- [1] Al Akoum. Automatic traffic using image processing. *Journal of Software Engineering and Applications*, 10:765–776, 01 2017. doi: 10.4236/jsea.2017.109042.
- [2] Straßenverkehrszentrale Baden-Württemberg. Die straßenverkehrszentrale, o. J. <https://www.svz-bw.de/strassenverkehrszentrale.html>, abgerufen am: 12.03.2019.
- [3] Riza Atiq bin OK Rahmat and Kasmiran bin Jumari. Vehicle detection using image processing for traffic control and surveillance system. In *Universiti Kebangsaan Malaysia, 8th World Congress on ITS, Sydney*, 2001.
- [4] Surendra Gupte and Nikolaos P Papanikolopoulos. Algorithms for vehicle classification. 2000.
- [5] Marco Herglotz, Simon Knab, and Jonas Kümmelin. Individuelle Verkehrsoptimierung durch Analyse von Verkehrskameras mittels Machine Learning, 2018. [https://github.com/W-i-Z-o/ihatestau-public/raw/master/Studienarbeit\\_2018\\_Herglotz\\_Knab\\_Kuemmerlin.pdf](https://github.com/W-i-Z-o/ihatestau-public/raw/master/Studienarbeit_2018_Herglotz_Knab_Kuemmerlin.pdf), abgerufen am: 07.03.2019.
- [6] Alan M McIvor. Background subtraction techniques. *Proc. of Image and Vision Computing*, 4:3099–3104, 2000.
- [7] Nobuyuki Otsu. A Threshold Selection Method from Gray-level Histograms. *IEEE Transactions on Systems, Man and Cybernetics*, 9(1), 1979. doi: 10.1109/TSMC.1979.4310076. URL <http://dx.doi.org/10.1109/TSMC.1979.4310076>.
- [8] Proj4J. Proj4j, 2017. <https://trac.osgeo.org/proj4j/>, abgerufen am: 9.4.2019.

- [9] Spatial Reference. EPSG:25832 (ETRS89 / UTM zone 32N), o. J.. <http://spatialreference.org/ref/epsg/wgs-84/>, abgerufen am: 31.3.2019.
- [10] Spatial Reference. EPSG:4326 (WGS 84), o. J.. <http://spatialreference.org/ref/epsg/etrs89-utm-zone-32n/>, abgerufen am: 31.3.2019.
- [11] OpenCV team. Face detection using haar cascades, 2015. [https://docs.opencv.org/3.1.0/d7/d8b/tutorial\\_py\\_face\\_detection.html](https://docs.opencv.org/3.1.0/d7/d8b/tutorial_py_face_detection.html), abgerufen am: 30.03.2019.
- [12] OpenCV team. Opencv library, 2019. <https://opencv.org/>, abgerufen am: 29.03.2019.