



---

# Detektion von liegendem Schnee mit Bildanalyse

## Bachelorthesis

Studiengang: BSc Informatik (CPVR)  
Autor: Marko Public  
Betreuer: Marcus Hudritsch  
Auftraggeber: Lorenz Martin - Daniel Bättig - ASTRA - MeteoSchweiz  
Experte: Mathis Marugg  
Datum: 19. Januar 2017

# Management Summary

## Ausgangslage

Das Bundesamt für Straßen (ASTRA) entwickelt in Zusammenarbeit mit der BFH ein neues Frühwarnsystem für Strassenglätte, das 2018 in Betrieb gehen soll. Aus gemessenen Meteorologischen Daten kann für die meisten Situationen bereits jetzt die Wahrscheinlichkeit für Glätte errechnet werden.

Es gibt aber ein Szenario, das aus Wetterdaten nicht berechenbar ist: Wenn Schnee neben der Fahrbahn liegt, in der Nachmittagssonne schmilzt und auf den Asphalt fliesst. Gegen Abend kann die Temperatur wieder unter den Gefrierpunkt sinken und das Schmelzwasser auf der Fahrbahn zum Gefrieren bringen.

Die Idee ist nun dieses Szenario frühzeitig zu erkennen: Mit Methoden der Bildverarbeitung und der Merkmalsextraktion sowie Webcam-Bildern soll automatisch bestimmt werden, ob neben der Straße Schnee liegt oder nicht.

## Datenaufbereitung

Für den Aufbau einer Wissensbasis wurden über mehrere Winter Webcam-Bilder gesammelt. Die Bilder wurden zu Kategorien (Schnee / kein Schnee) zugeordnet und auf Bildausschnitte reduziert. Aus den Bildausschnitten konnten Merkmale wie Histogramm, Durchschnittsfarbe pro Farbkanal und Kontrast extrahiert und in einer Datenbank gespeichert werden.

Diese Datenbasis stellt die Grundlage, um Referenzwerte für verschiedene Tageszeiten, Sonnenstände und Wetterlagen (Sonnig / Niederschlag / Nebel) zu gewinnen – jeweils für beide Kategorien.

Der Ansatz ist nun, dass ein Distanzmaß Anwendung findet, das aussagt, ob ein Bild näher an den Referenzwerten der einen oder der anderen Kategorie liegt. Dieses Distanzmaß stützt sich auf die extrahierten statistischen Merkmale aus der Datenbasis.

## Algorithmus

1. In kurzer Reihenfolge werden Bilder von der Webcam heruntergeladen und kombiniert. So ist sichergestellt, dass keine Autos auf dem Bild sind, welche die Farbe des betrachteten Bildausschnitts verfälschen könnten
2. Es folgt die statistische Auswertung aller Bildausschnitte
3. Schliesslich die Suche nach den nächsten Nachbaren unter allen Referenzwerten für jeden betrachteten Bildausschnitt
4. Das Bild wird derjenigen Kategorie zugewiesen, zu welcher eine Mehrzahl der nächsten Nachbaren angehört

## Ergebnisse

Im Rahmen der Arbeit entstand ein mächtiges Werkzeug als .NET-Applikation, das Aufbau und Pflege der Datenbasis unterstützt und den parametrierbaren Algorithmus systematisch testet. Der mehrmals optimierte Algorithmus erreicht eine genügend hohe Erkennungsrate für den Einsatz in einer produktiven Umgebung.

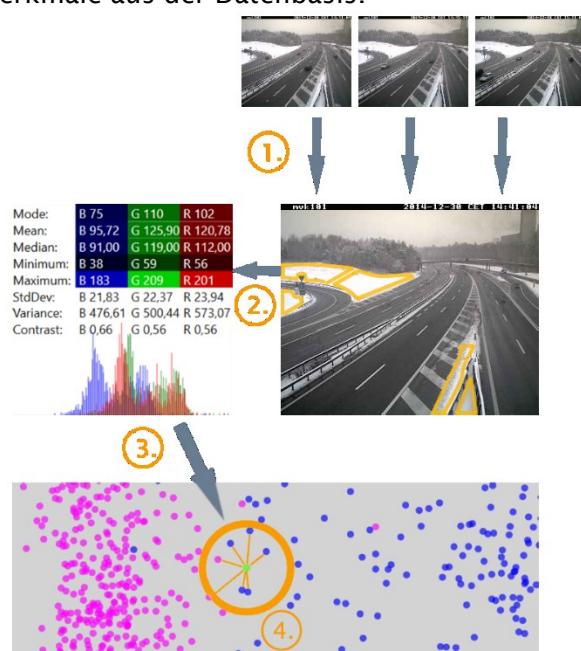


Abbildung 1: Schematischer Ablauf einer Detektion

# Inhaltsverzeichnis

1 Einleitung	5
2 Kameras	6
2.1 Aufzeichnung der Bilder	6
2.2 Standorte	6
2.2.1 Karte	11
2.2.2 Zeitdiskrepanz zwischen Dateinamen und Meta-Daten	11
2.3 Beschaffenheit der JPEG-Dateien	12
3 Projekt 2	13
3.1 Technologie	13
3.2 Metadaten in Datenbank	13
3.3 Segmentierung des Bildes	14
3.3.1 Polygon Editor	15
3.4 Bereinigung von Autos	15
3.4.1 Algorithmus	15
3.5 Naive Schneedetektion	19
3.6 Resultate	20
3.7 Weitere Themen	20
3.7.1 Wetterdaten	20
3.7.2 Intelligenter Cron-Job	21
4 Bachelorarbeit	22
4.1 Verbesserung zu Projekt 2	22
4.2 F-Test	22
4.3 Stichprobe aufbauen	22
4.3.1 Kategorien	23
4.3.2 Programm	24
4.3.3 Anpassungen Galerie Modul	25
4.3.4 Erfassung der Kategorien	25
4.3.5 Verhältnis	25
4.3.6 Wahrscheinlichkeit für eingeschränkte Sichtverhältnisse	26
4.4 Neue Segmentierung (Patches)	27
4.5 Statistische Werte pro Patch	28
4.5.1 Histogramm	28
4.5.2 Modalwert	30
4.5.3 Durchschnittsfarbe (Mittelwert)	30
4.5.4 Median	31
4.5.5 Varianz und Standardabweichung	31
4.5.6 Kontrast, Minimum und Maximum	32
4.5.7 Berechnung der Kennzahlen ohne OpenCV-Methoden	32
4.5.8 Statistik Modul	32
4.6 Statistische Werte in Datenbank ablegen	34
4.6.1 Datenstruktur	34
4.6.2 Programm	34
4.6.3 Grösse der Datenbank	35
4.7 Plot Modul	36
4.7.1 Erkenntnisse	38
4.8 2D Plot Modul	38
4.8.1 Erkenntnisse	40
4.9 Statistische Werte zeitlich kombinieren	40
4.9.1 Statistische Werte in 2-Stunden-Zeitfenster kombinieren und abspeichern	40
4.9.2 Erfasste Jahreswochen	41
4.9.3 Zeitfenster	42
4.9.4 Kategorie-Permutationen	42

4.9.5 Skript für die Kombination	43
4.9.6 Combined Plot Modul	43
4.10 Klassifikation	44
4.10.1 Distanzmass	44
4.10.2 Algorithmus	45
4.10.3 Benutzerschnittstelle	46
4.10.4 Resultate	47
4.10.5 Optimierung: Median-Kombination	48
4.10.6 Optimierung: Inputbild Kombination	48
4.10.7 Optimierung: Patches aus 3 Inputs auswählen	50
4.10.8 Ergebnisse	51
4.11 Weitere Themen	51
4.11.1 Sonneneinstrahlungs-Patch	51
4.11.2 Logit Regression	51
4.11.3 Weitere Klassifikatoren	52
5 Schlussfolgerungen / Fazit	53
6 Abbildungsverzeichnis	54
7 Tabellenverzeichnis	55
8 Programmcodeausschnitte	55
9 Glossar	56
10 Literaturverzeichnis	57
11 Selbständigkeitserklärung	58

# 1 Einleitung

Im Auftrag der ASTRA betreibt das Institut für Risiko- und Extremwertanalyse der Berner Fachhochschule ein Strassenglätte-Prognosen-System. Dieses bezieht Wetterdaten von SwissMeteo und modelliert Prognosen. Das Ziel ist rechtzeitig und korrekt zu warnen, wenn sich der Strassenzustand verschlechtert hat. So können Schneeflüge und Salzstreuer rechtzeitig ausrücken.

Aus den Wetterdaten lässt sich genau und örtlich granular voraussagen, ob in den nächsten Stunden Schnee geräumt werden muss. Es gibt eine Gegebenheit, die das Ausrücken des Winterdiensts unverzichtbar macht aber aus Wetterdaten nicht errechnet werden kann: Neben der Strasse, zum Beispiel auf dem Pannenstreifen oder auf benachbarten Oberflächen, liegt Schnee. Dieser schmilzt tagsüber bei Temperaturen über Null und fliesst auf die Fahrbahn. Nach Sonnenuntergang und tieferen Temperaturen, kann dieses Wasser gefrieren und Strassenglätte verursachen.

Die kritischen Tageszeiten sind die Stosszeiten am Abend und am Morgen. Darum müssen Räumungsfahrzeuge früh genug alarmiert werden, wenn es zu einer Glatteis-Situation kommen kann. Konkret bedeutet das, dass Wetter und Schneeverhältnisse am Morgen des Tages beobachtet werden. Spätestens um 11 Uhr morgens sollen die Räumungsfahrzeuge alarmiert werden, damit um 3 Uhr nachmittags an allen kritischen Abschnitten des Nationalstrassennetzes bereits Salz verteilt werden konnte.

Um bei dieser Situation zuverlässig Alarm geben zu können, sollen die Webcams, die den Verkehrsfluss überwachen, genutzt werden. Die Webcam Bilder wurden über zwei Winter gesammelt. Dieses Bildarchiv bietet nun die Grundlage für das Erarbeiten einer Wissensbasis. Mit der Wissensbasis und Methoden der Bildverarbeitung und der Merkmalsextraktion sollen Algorithmen entstehen, die zuverlässig detektieren können, ob neben der Strasse Schnee liegt. In einem potentiellen Livebetrieb, als Teil des neuen Strassenglätte-Warnsystems, sollen die Resultate dieser Arbeit beitragen bei Risikosituationen frühzeitig zu alarmieren.

## 2 Kameras

Für die Überwachung des Verkehrsflusses betreibt das Bundesamt für Strassen (ASTRA) Webcams<sup>1</sup>. Die Kameras sind an verschiedenen Orten im Nationalstrassenetz platziert. Die Geräte sind an Kandelabern, Brücken wie auch an Verkehrsschildern montiert und sind rund um die Uhr in Betrieb.

### 2.1 Aufzeichnung der Bilder

Die Aufzeichnung der Bilder begann anfangs Dezember 2014. Ein Cron-Job auf einem BFH-Server namens ‘athena’ stösst ein Skript an, dass alle 10 Minuten das aktuelle Bild der Webcam als JPG-Datei herunterlädt und das Bild auf einem File-Share ablegt. So funktioniert das Bash-Skript:

```
#!/bin/bash

# URL der Online-Webcams der ASTRA
url=http://www.astramobcam.ch/kamera
# Ziel: File-Share auf athena
dest=/srv/athena.bfh.ch/projects/astra/

# Liste der Kameras erstellen. Ziel-Ordner müssen existieren!
cams=(mvk021 mvk101 mvk105 mvk107 mvk110 mvk120 mvk122 mvk131)

# Aktuelles Datum auslesen in der Form JahrMonatTag_StundeMinuteSekunde
date=$(date -u +%Y%m%d_%H%M%S)
echo $date

# Zugriff auf File-Share prüfen
# Wenn nicht erreichbar, wird abgebrochen und eine Meldung per Mail versendet
if [ ! -d ${dest} ]; then
    echo ${date} | mail -s "astrafetch - storage destination not available" vep2@bfh.ch --
-f vep2@bfh.ch
    exit -1
fi

# Pro Kamera in der Liste
for cam in ${cams[@]}; do
    # Leere Log-Datei erstellen
    log=${dest}/${cam}_${date}.log
    #Leere JPG-Datei erstellen
    img=${dest}/${cam}_${date}.jpg
    # Kamerabild per WGET herunterladen.
    # Daten in JPG-Datei schreiben und Log in Log-Datei schreiben
    /usr/bin/wget ${url}/${cam}/live.jpg -O ${img} -o ${log}
done
```

Code 1: Bash-Skript für das Herunterladen der Bilder

Über zwei Winter wurden für die ausgewählten 8 Kameras insgesamt 418'111 Bilder gesammelt. Da nur die Kamerabilder im Winter von Interesse sind, lief der Cron-Job von Dezember 2014 bis Mai 2015 und wieder vom September 2015 bis April 2016.

### 2.2 Standorte

Die ursprünglich gewählten Standorte befinden sich im Grossraum Bern und entlang der Achse Thunersee-Brienzersee-Brünigpass. In Bern zeigen die Kameras auf die Autobahn A1 und A6, während die andere Gruppe auf der Nationalstrasse A8 verteilt ist. Ursprünglich ging man davon aus, dass die Kameras an einem Ort fest installiert sind. Bei der Verarbeitung der Bilder fiel aber auf, dass gewisse Kameras zwar die Bezeichnung behielten aber während der Datensammlung physisch verschoben wurden. Die Bezeichnungen der Kameras lassen darauf schliessen, dass Geräte in unregelmässigen

<sup>1</sup> Webplattform der Mobilen Video Kameras: <http://www.astramobcam.ch/>

Abständen bewegt werden. Das Kürzel 'mvk' bedeutet Mobile-Video-Kamera<sup>2</sup>. Dieser Umstand war bei Projektbeginn nicht bekannt.

Vier der acht Kameras lieferten im Verlauf der Datenaufnahme somit Bilder aus verschiedenen Perspektiven. So kamen Autobahnabschnitte in Kirchberg (A1) und bei Grenchen (A5) hinzu. Somit existieren Daten für folgende Kameras:



### **mvk021**

Bezeichnung: A1 Grauholz > Bern

Koordinaten: 46°59'20.1"N 7°28'24.0"E

Aufzeichnung Winter 14/15: 2014-12-02 – 2015-06-10

Aufzeichnung Winter 15/16: 2015-09-21 – 2015-12-04

Nach dem 4. Dezember 2015 lieferte die Kamera unter der gleichen Bezeichnung Bilder von einem anderen Ort aus. Für die Weiterverarbeitung wird die Bezeichnung mvk022 verwendet.

Abbildung 2: Mobile Video Kamera 021



### **mvk022**

Bezeichnung: A5 Pieterlen > Biel

Koordinaten: 47°10'06.2"N 7°21'33.3"E

Aufzeichnung Winter 15/16: 2015-12-07 – 2016-04-15

Abbildung 3: Mobile Video Kamera 022



### **mvk101**

Bezeichnung: A1 Weyermannshaus > Grauholz

Koordinaten: 46°57'02.9"N 7°24'29.2"E

Aufzeichnung Winter 14/15: 2014-12-02 – 2015-06-10

Aufzeichnung Winter 15/16: 2015-09-21 – 2015-11-12

Nach dem 11. November 2015 lieferte die Kamera unter der gleichen Bezeichnung Bilder von einem anderen Ort aus. Für die Weiterverarbeitung wird die Bezeichnung mvk102 verwendet.

Abbildung 4: Mobile Video Kamera 101

<sup>2</sup> Informationen zu den Mobile Video Kameras: [http://www.astramobcam.ch/benutzer/MVK\\_Benutzer-Kurzanleitung\\_V1.pdf](http://www.astramobcam.ch/benutzer/MVK_Benutzer-Kurzanleitung_V1.pdf)



Abbildung 5: Mobile Video Kamera 102

### **mvk102**

Bezeichnung: A1 Tunnel Kirchb. Port. Zürich  
Koordinaten: 47°05'12.4"N 7°34'25.9"E  
Aufzeichnung Winter 15/16: 2015-11-16 – 2016-04-15



Abbildung 6: Mobile Video Kamera 105

### **mvk105**

Bezeichnung: A6 Wankdorf > Thun  
Koordinaten: 46°57'58.2"N 7°28'15.0"E  
Aufzeichnung Winter 14/15: 2014-12-02 – 2015-06-10  
Aufzeichnung Winter 15/16: 2015-10-20 – 2015-10-28

Nach dem 28. Oktober 2015 lieferte die Kamera unter der gleichen Bezeichnung Bilder von einem anderen Ort aus. Für die Weiterverarbeitung wird die Bezeichnung mvk106 verwendet.



Abbildung 7: Mobile Video Kamera 106

### **mvk106**

Bezeichnung: A1 Rampendosierung Kirchberg (Ost)  
Koordinaten: 47°04'45.6"N 7°34'20.5"E  
Aufzeichnung Winter 15/16: 2015-11-04 – 2016-04-15



Abbildung 8: Mobile Video Kamera 107

### **mvk107**

Bezeichnung: A6 Ostring > Wankdorf

Koordinaten: 46°57'04.7"N 7°28'19.5"E

Aufzeichnung Winter 14/15: 2014-12-02 – 2015-06-10

Aufzeichnung Winter 15/16: 2015-09-21 – 2015-10-28

Nach dem 28. Oktober 2015 lieferte die Kamera unter der gleichen Bezeichnung Bilder von einem anderen Ort aus. Für die Weiterverarbeitung wird die Bezeichnung mvk108 verwendet.



Abbildung 9: Mobile Video Kamera 108

### **mvk108**

Bezeichnung: A1 Rampendosierung Kirchberg (West)

Koordinaten: 47°04'45.6"N 7°34'20.5"E

Aufzeichnung Winter 15/16: 2015-11-04 – 2016-04-15



Abbildung 10: Mobile Video Kamera 110

### **mvk110**

Bezeichnung: A8 Gnoll (Brünig) > Luzern

Koordinaten: 46°45'05.5"N 8°07'42.0"E

Aufzeichnung Winter 14/15: 2014-12-02 – 2015-06-10

Aufzeichnung Winter 15/16: 2015-09-21 – 2016-04-15

**mvk120**

Bezeichnung: A8 Brienz > LU

Koordinaten: 46°44'33.3"N 8°03'46.0"E

Aufzeichnung Winter 14/15: 2014-12-02 – 2015-06-10

Aufzeichnung Winter 15/16: 2015-09-21 – 2016-04-15

Abbildung 11: Mobile Video Kamera 120

**mvk122**

Bezeichnung: A8 Faulensee > LU

Koordinaten: 46°40'04.6"N 7°43'20.5"E

Aufzeichnung Winter 14/15: 2014-12-18 – 2015-06-10

Aufzeichnung Winter 15/16: 2015-09-21 – 2016-04-12

Abbildung 12: Mobile Video Kamera 122

**mvk131**

Bezeichnung: A8 Soliwaldtunnel Süd > LU

Koordinaten: 46°45'05.5"N 8°06'29.8"E

Aufzeichnung Winter 14/15: 2014-12-18 – 2015-06-10

Aufzeichnung Winter 15/16: 2015-09-21 – 2016-04-15

Abbildung 13: Mobile Video Kamera 131

## 2.2.1 Karte

Die erfassten zwölf Kamera-Standorte auf Google Maps<sup>3</sup> markiert und gekennzeichnet:

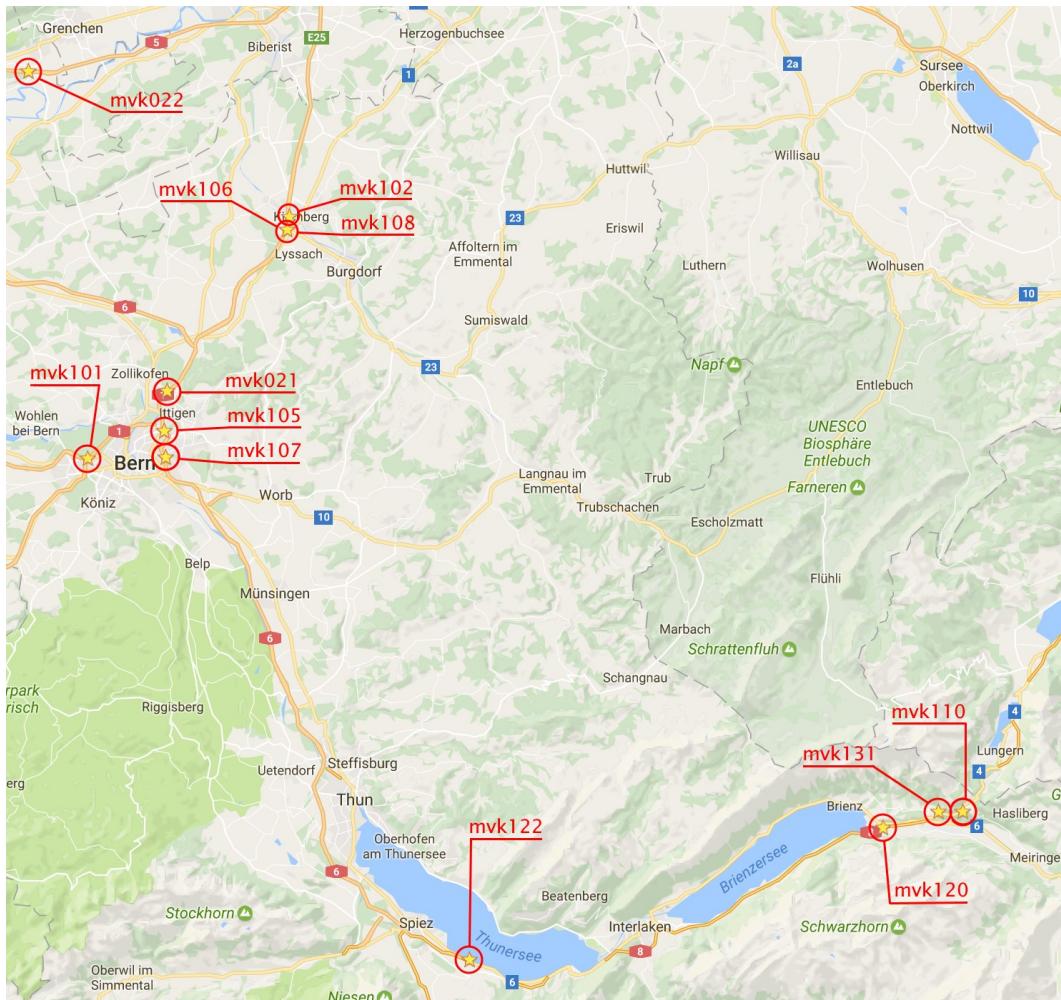


Abbildung 14: Standorte der 12 Kameras auf Google Map

## 2.2.2 Zeitdiskrepanz zwischen Dateinamen und Meta-Daten

Das Skript, welches die Bilder herunterlädt, gab jedem Bild einen Dateinamen. Dieser ist eine Kombination aus Kameraname und Uhrzeit des Downloads. Bei näherer Betrachtung der Bilder entstand der Eindruck, dass die Zeit im Dateinamen um eine oder zwei Stunden von der echten Aufnahmezeit abweicht. Diese Diskrepanz konnte aus Analyse des Sonnenstandes (Sonnenaufgang und Sonnenuntergang) auf den Bildern rekonstruiert werden. Beim Abgleich des Sonnenstandes mit der echten Uhrzeit kam die Web-Applikation von [timeanddate.com](http://www.timeanddate.com)<sup>4</sup> zu Hilfe.

Die Uhrzeit im Dateinamen entspricht der Systemzeit des Servers, welche nicht mit der 'echten' Aufnahmezeit übereinstimmt. Glücklicherweise speichert die Kamera, neben den eigentlichen Bilddaten auch weiter Metainformationen in die Datei. Die JPG lässt sich in einem Texteditor öffnen und gibt viele zusätzliche Information in einem Header-Teil der Datei preis. Unter den Metainformationen befindet sich auch eine Sektion namens 'Fingerprint', die den genauen Aufnahmezeitpunkt beinhaltet:

<sup>3</sup> Kartenausschnitt auf Google Maps: <https://www.google.ch/maps/@46.919277,7.6737975,10.5z>

<sup>4</sup> Sonnenstand in Bern und historische Daten: <https://www.timeanddate.com/sun/switzerland/bernopenweather>

Auszug aus dem Header-Teil	Erklärung der relevanten Schlüsselwörter
[...] SECTION FINGERPRINT VER=1.0 PRD=MOBOTIX FRM=381865 DAT=2014-12-30 TIM=14:41:04.009 TZN=CET TIT=1419946864.009 ENO=0 IMT=IMAGE ENDSECTION FINGERPRINT [...]	Version: Version der Kamera / Firmware Producer: Hersteller der Kamera / Firmware  Datum Time: Zeit Timezone: Zeitzone Timestamp: Unix-Zeitstempel

Code 2: Auszug aus dem Header-Teil der Bilddatei mvk101\_20141530\_134001.jpg

Beim Registrieren der Bildinformationen in die Datenbank wurden nicht der Dateiname, sondern die korrekten Informationen aus dem Header-Teil berücksichtigt.

### 2.3 Beschaffenheit der JPEG-Dateien

Die gesammelten Bilddateien sind alles JPG-Dateien. Sie sind 352 Pixel breit und 288 Pixel hoch, haben eine Auflösung von 96dpi und eine Farbtiefe von 24bit (True Color<sup>5</sup>). Pro Farbkanal (Rot, Grün, Blau) und Bildpunkt ist die Farbintensität in einem Byte (0-255) kodiert. Die Bilddateien beinhalten, abgesehen vom proprietären Header-Teil, keine weiteren Metadaten in der Form von Exif-Informationen.

Von blossem Auge wirken die Bilder stark komprimiert und weisen sichtbare Kompressionsartefakte (Blockartefakte, Unschärfe) auf. Abhängig vom Bildinhalt sind die Dateien zwischen 6 und 25 Kilobyte gross. Die kleinen Dateien entstehen hauptsächlich nachts, wenn die Webcams von einem Farbbild auf Schwarz-Weiss umschalten.

<sup>5</sup> True Color: [https://de.wikipedia.org/wiki/True\\_Color](https://de.wikipedia.org/wiki/True_Color)

## 3 Projekt 2

Im Projekt 2, mit Durchführung im Herbstsemester 2015, war der Ansatz die Schneedektion ohne Vorwissen über Lichtverhältnisse auf dem Bild durchzuführen.

Folgende Themen standen während der Projektarbeit im Zentrum:

- Metadaten des Bildarchivs in eine Datenbank abspeichern
- Autos aus den Aufnahmen bereinigen
- Manuelle Segmentierung der Bilder
- Detektion mit globalen Schwellwerten pro Bild und Segment

### 3.1 Technologie

Während dem Projekt ist eine Applikation entstanden, die die grosse Datenmenge übersichtlicher macht. Diese ermöglicht die Segmentierung der Bilder und macht die Schneedektion für verschiedene Szenarien testweise ausführbar. Bei der Entwicklung kamen folgende Technologien und Plattformen zum Einsatz:

- Entwicklungsumgebung: Microsoft Visual Studio Community 2015
- Die Benutzerschnittstelle ist umgesetzt mit XAML (Windows Presentation Foundation)
- Für einige Komponenten der Benutzerschnittstelle wurde auf das Extended WPF Toolkit zurückgegriffen
- Die Programmlogik ist in C# der Version .NET 4.5.2 geschrieben
- Die Schnittstelle zur Datenbank bildet LINQ to SQL
- Für die Serialisierung und Deserialisierung von Objekten wurde Newtonsoft.Json verwendet
- Bei der Bildverarbeitung wird OpenCV genutzt. Das ist eine quelloffene und weit verbreitete Programmbibliothek für Computer Vision und Machine Learning.
- Da OpenCV keine Programmschnittstelle zu C# bereitstellt, werden OpenCV Operationen über eine weitere quelloffene Programmschnittstelle aufgerufen: Emgu CV
- Für Quellcodeverwaltung wurde GitHub gewählt. Hier ist der gesamte Programmcode sowie die Dokumentation abgelegt und versioniert.

<https://github.com/uzapy/ch.bfh.bti7321.w2016.schneedektion>

### 3.2 Metadaten in Datenbank

Um eine handhabbare Übersicht der bestehenden Daten zu schaffen, wurde eine Datenbank aufgebaut. Wegen der überschaubaren Datenmenge und der Möglichkeit der Persistierung in einer Quellcodeverwaltung, ist die Datenbank als lokale MDF-Datei realisiert. Diese Datei funktioniert ohne SQL-Server und ist direkt in die Entwicklungsumgebung eingebunden. Alle zugänglichen Daten über die Bilder konnten so in Tabellenform abgelegt werden. Zu jedem Bild sind folgende Werte pro Zeile abgespeichert: Dateipfad zu Bild, Kameraname und Zeitpunkt der Aufnahme.

Für die Datenvisualisierung wurde eine Applikation entwickelt. Mit welcher die Betrachtung und Analyse der Bilder pro Kameraperspektive, Aufnahmezeitpunkt und Bildausschnitt ermöglicht wird:

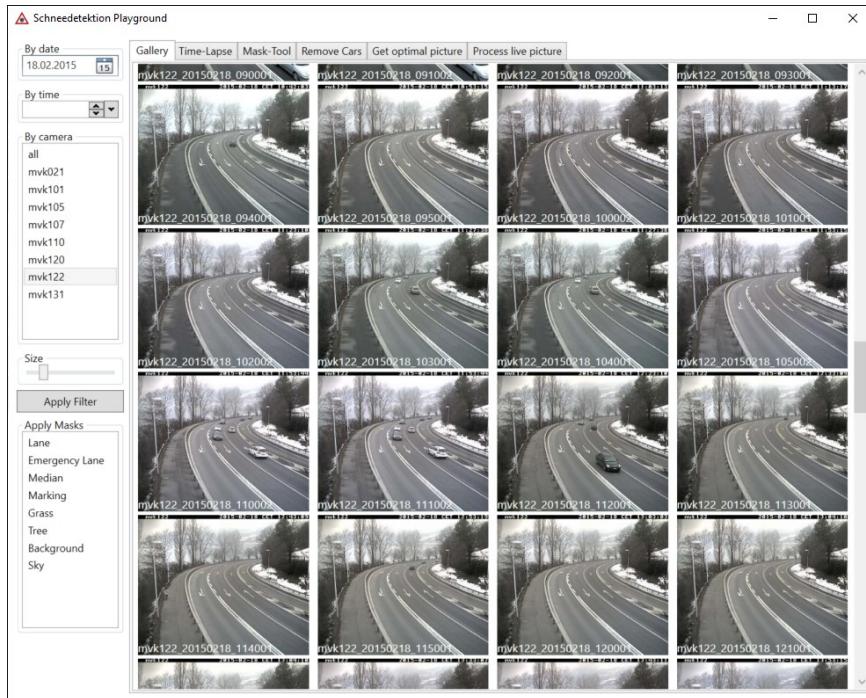


Abbildung 15: Bildschirmfoto des Galerie Moduls der Applikation

### 3.3 Segmentierung des Bildes

Da der Schnee auf der Fahrbahn nicht lange liegen bleibt, sollte die Detektion sich auf die umliegenden Regionen der Strasse auf dem Bild konzentrieren. Hierzu wurde ein Werkzeug entwickelt, dass ermöglicht, das Bild virtuell in verschiedene Regionen zu unterteilen. Zum Einsatz kamen folgende Bildregionen: Fahrbahn, Pannenstreifen, Mittelstreifen, Strassenmarkierung, Rasen, Hintergrund und Himmel. Zu jeder Kameraperspektive wurden die Regionen so definiert, dass möglichst jeder Bildpunkt zu genau einer Region gehört.

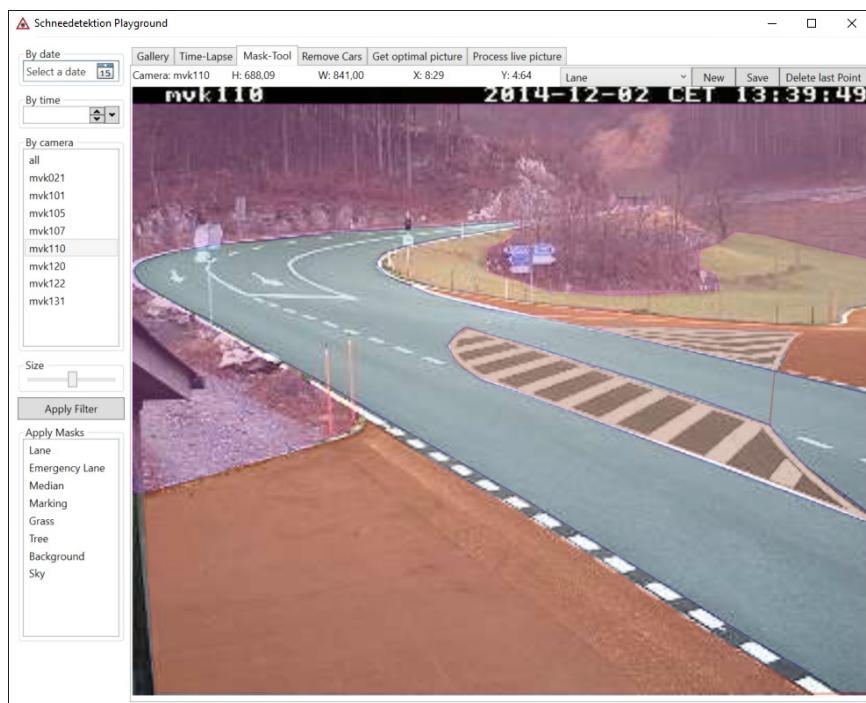


Abbildung 16: Bildschirmfoto des Segmentierungsmoduls der Applikation

### 3.3.1 Polygon Editor

Die Applikation wurde um einen einfachen Polygon-Editor erweitert. Dieser macht es möglich über das Bild per Mausklick farbige Polygone zu zeichnen. Über die Darstellungsebene wird eine virtuelle Zeichnungsebene gelegt. Auf diesem lassen sich die Polygone definieren. Jede Bildregion hat eine eigene Farbe:

- Fahrbahn: Blau
- Pannenstreifen: Rot
- Mittelstreifen: Gelb
- Straßenmarkierung: Braun
- Rasen: Violett
- Hintergrund: Magenta
- Himmel: Gold

Per Drop-Down lässt sich die gewünschte Region auswählen. Die Zeichnungsebene verfolgt die Mausposition. Bei jedem Klick wird die X- und Y-Position der Maus auf der Ebene als neuer Punkt des Polygons hinzugefügt und dargestellt. Die definierten Polygone werden ebenfalls in die Datenbank abgespeichert. Davor werden die Positionen der Polygonpunkte auf zwischen 0 und 1 skaliert.

```
public static string SerializePointCollection(  
    Polygon polygon, double viewWidth, double viewHeight)  
{  
    PointCollection pointCollection = new PointCollection();  
    foreach (Point point in polygon.Points)  
    {  
        // Skalierung der Punkte zwischen 0 und 1  
        // 1 / Fensterbreite * X-Position  
        // 1 / Fensterhöhe * Y-Position  
        pointCollection.Add(new Point(1 / viewWidth * point.X, 1 / viewHeight * point.Y));  
    }  
    // Serialisierung zu Json-Objekt  
    return JsonConvert.SerializeObject(pointCollection);  
}
```

Code 3: Serialisierung und Skalierung eines Polygons

Dies ist nötig, damit das Polygon nicht abhängig ist von der aktuellen Fenstergrösse der Applikation.

### 3.4 Bereinigung von Autos

Damit Autos, welche sich gerade auf der Aufnahme befinden, die Farben der Regionen nicht verfälschen, muss ein Weg gefunden werden die Fahrzeuge aus dem Bild zu entfernen. Die Idee ist hier, dass mehrere Bilder, die in kurzen Abständen aufgenommen wurden mit mathematischen Operationen miteinander verschmolzen werden. So soll ein neues, autofreies Bild entstehen, dass für die Weiterverarbeitung bestimmt ist.

#### 3.4.1 Algorithmus

Zunächst wird ein erstes Bild über die Web-Schnittstelle heruntergeladen und im Dateisystem abgelegt. Danach muss einige Sekunden gewartet werden. Falls in zu hoher Frequenz Bilder von der Webcam angefordert werden, ist es möglich, dass die Kamera dasselbe Bild noch einmal liefert. Anschliessend erfolgt die Berechnung der Differenz der beiden Bilder:

```
public void CalculateAbsoluteDifference  
    (string imagePath0, string imagePath1, string resultPath)  
{  
    // Bilder einlesen  
    Image<Bgr, byte> image0 = new Image<Bgr, byte>(imagePath0);  
    Image<Bgr, byte> image1 = new Image<Bgr, byte>(imagePath1);  
  
    // Neues, leeres Bild erzeugen  
    Image<Bgr, byte> result1 = new Image<Bgr, byte>(new byte[288, 352, 3]);  
  
    // Absolute Differenz der beiden Bilder berechnen
```

```

result1 = image0.AbsDiff(image1);
// Thresholden und Invertieren
result1._ThresholdBinaryInv(new Bgr(50, 50, 50), new Bgr(255, 255, 255));
// Schwarze Fläche grösser machen per Erosions-Operation
result1._Erode(3);

// Differenz der Bilder in neuem Bild abspeichern
result1.Save(resultPath);
}

```

Code 4: Berechnung der Absoluten Differenz zweier Bilder

Das Resultat ist eine Bitmaske, welche schwarze Flecken enthält, dort wo sich die beiden Bilder unterscheiden. Die Bilder unterscheiden sich an jenen Stellen, wo ein Auto sich nur im einem der Bilder befindet.



Abbildung 17: Differenz aus zwei Kamerabildern und resultierende Bitmaske

Aus dem ersten Bild und der Bitmaske kann ein neues Bild (Kandidat) mit leeren, schwarzen Flecken erzeugt werden. Dies ist mit einem einfachen EmguCV-Aufruf ausführbar:

```

public void GetMaskedImage(string imagePath, string maskPath, string resultPath)
{
    // Bilder einlesen
    Image<Bgr, byte> image = new Image<Bgr, byte>(imagePath);
    Image<Bgr, byte> mask = new Image<Bgr, byte>(maskPath);
    // Neues, leeres Bild erzeugen
    Image<Bgr, byte> result = new Image<Bgr, byte>(new byte[288, 352, 3]);

    // Bildpunkte in neues Bild kopieren, ausser dort wo Bitmaske Flecken hat
    CvInvoke.cvCopy(image, result, mask);

    result.Save(resultPath);
}

```

Code 5: Bild Maskieren mit Bitmaske

So sieht ein Kandidatenbild aus. Die Autos aus den Quell-Bildern sind entfernt. Es folgt der Versuch diese Löcher zu füllen.



Abbildung 18: Kandidatenbild mit schwarzen Löchern

Es kann kalkuliert werden, welcher Anteil der entstandenen Bitmaske, die zum Kandidaten gehört, schwarz ist. Falls mehr als ein Prozent der Bitmaske schwarz ist, wird ein weiteres Bild im Bereinigungsprozess berücksichtigt.

Erneut wird kurz gewartet und ein weiteres Bild von der Webcam heruntergeladen. Aus dem zweiten und dem dritten Bild lässt sich wieder die Bitmaske erzeugen, welche mit der vorangegangenen Bitmaske verglichen wird. An denjenigen Stellen, wo sich die zwei Bitmasken unterscheiden, können nun Bildinhalte aus dem dritten Bild zum Auffüllen des Kandidatenbildes entnommen werden:

```
public void FillMaskHoles(string maskPath0, string maskPath1,
    string newImagePath, string candidateImagePath, string resultPath)
{
    // Masken und Bilder einlesen
    Image<Bgr, byte> mask0 = new Image<Bgr, byte>(maskPath0);
    Image<Bgr, byte> mask1 = new Image<Bgr, byte>(maskPath1);
    Image<Bgr, byte> newImage = new Image<Bgr, byte>(newImagePath);
    Image<Bgr, byte> candidate = new Image<Bgr, byte>(candidateImagePath);
    // Neues, leeres Bild erzeugen
    Image<Bgr, byte> resultMask = new Image<Bgr, byte>(new byte[288, 352, 3]);

    // Neue Maske erzeugen: Repräsentiert die Unterschiede der beiden Input-Masken
    for (int i = 0; i < mask0.Cols; i++)
    {
        for (int j = 0; j < mask0.Rows; j++)
        {
            if (mask0.Data[j, i, 0] == 0 &&
                mask0.Data[j, i, 1] == 0 &&
                mask0.Data[j, i, 2] == 0 &&
                mask1.Data[j, i, 0] == 255 &&
                mask1.Data[j, i, 1] == 255 &&
                mask1.Data[j, i, 2] == 255)
            {
                // Schwarzes Pixel setzen, wenn die Masken unterschiedlich sind
                resultMask.Data[j, i, 0] = 0;
                resultMask.Data[j, i, 1] = 0;
                resultMask.Data[j, i, 2] = 0;
            }
            else
            {
                // Weisses Pixel setzen, wenn die Masken gleich sind
                resultMask.Data[j, i, 0] = 255;
                resultMask.Data[j, i, 1] = 255;
                resultMask.Data[j, i, 2] = 255;
            }
        }
    }
    // Maske invertieren
    resultMask._Not();

    // Bildpunkte in neues Bild kopieren, ausser dort wo Bitmaske Flecken hat
    CvInvoke.cvCopy(newImage, candidate, resultMask);

    candidate.Save(resultPath);
}
```

Code 6: Ausfüllen von Löchern im Kandidatenbild mit Inhalten

So kann das Ausfüllen der Löcher im Kandidatenbild aussehen:

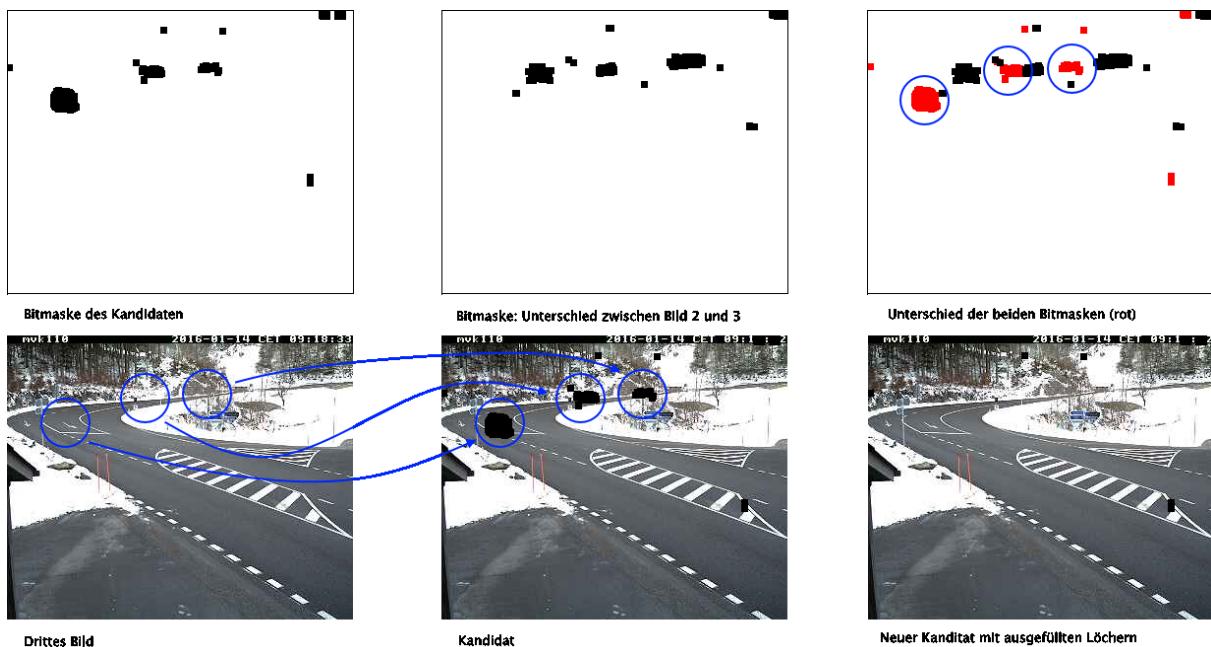


Abbildung 19: Auffüllen von schwarzen Löchern im Kandidatenbild

Dieser Prozess wird so lange wiederholt, bis die Bitmaske des Kandidatenbildes weniger als ein Prozent schwarz ist:

- Neues Bild herunterladen
- Differenz der letzten zwei Bildern erstellen (Bitmaske)
- Differenz zwischen dieser Bitmaske und der Kandidaten-Bitmaske erstellen
- Löcher im Kandidatenbild füllen mit Inhalt aus dem neuesten Bild
- Neue Kandidaten-Bitmaske erstellen
- Schwarzanteil der Kandidaten-Bitmaske berechnen

Nach der Bereinigung der Autos, wird das Bild in die vordefinierten Segmente auseinandergeschnitten:

```
public BitmapImage GetPatchBitmapImage (string imagePath, IEnumerable<Point> polygon)
{
    // UMatrix aus Bild erstellen
    Mat matrix = new Mat(imagePath, LoadImageType.AnyColor);
    UMat uMatrix = matrix.ToUMat(AccessType.ReadWrite);

    // Polygon auf Bildgrösse skalieren
    List<Point> scaledPoints = GetScaledPoints(polygon, uMatrix.Cols, uMatrix.Rows);
    // Polygon invertieren. Es wird ist ein Polygon erstellt,
    // dass alles ausser die ursprüngliche Polygonfläche abdeckt
    List<Drawing.Point> polygonPoints =
        GetInvertedPoints(scaledPoints, uMatrix.Cols, uMatrix.Rows);

    // Invertiertes Polygon auf das Bild anwenden
    using (VectorOfPoint vPoint = new VectorOfPoint(polygonPoints.ToArray()))
    using (VectorOfVectorOfPoint vvPoint = new VectorOfVectorOfPoint(vPoint))
    {
        // Alles schwarz anmalen, dass nicht im invertierten Polygon ist
        CvInvoke.FillPoly(uMatrix, vvPoint, new Bgr(0, 0, 0).MCvScalar);
    }

    // Matrix zu Image umwandeln
    Image<Bgr, byte> image = new Image<Bgr, byte>(uMatrix.Bitmap);

    // Bild auf Polygon-Region beschrneiden (Bounding-Box)
    // (schwarze Ränder oben, unten, links und rechts abschneiden)
    image.ROI = GetRegionOfInterest(scaledPoints);
```

```

    return OpenCVHelper.BitmapToBitmapImage(image.Bitmap);
}

```

Code 7: Ausschneiden eines Bildausschnitts

Es resultieren von Autos bereinigte Bildsegmente für die Schneedetektion. Ausgehend vom oben bearbeiten Beispiel entstehen fünf Segmente:

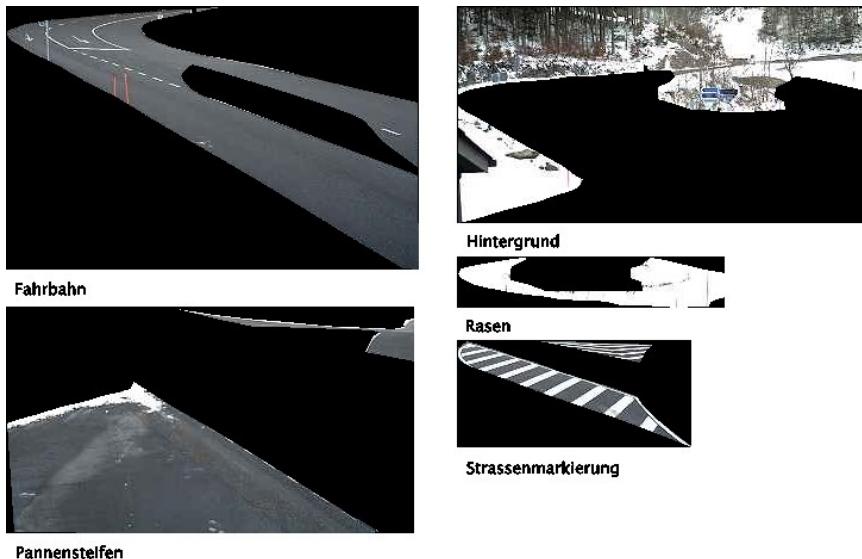


Abbildung 20: Ausgeschnittene Bildsegmente von mvk110

### 3.5 Naive Schneedetektion

Die Schneedetektion verfolgt nun einen naiven Ansatz: Für jede Region wird die Durchschnittsfarbe ausgerechnet. Dies bewerkstelligt ein einfacher OpenCV-Aufruf:

```
Bgr average = region.GetAverage(bitmask);
```

Code 8: Berechnung der Durchschnittsfarbe einer Region mit Bitmaske

Der Rückgabewert ist eine OpenCV-Struktur (Bgr = Blue-Green-Red), dass den Durchschnittswert pro Kanal als Fließkommazahl enthält. Diese Struktur kann als dreidimensionaler Vektor betrachtet werden. Die Distanz zu vorberechneten Referenzwerten lässt sich mit der Euklidischen Distanz<sup>6</sup> errechnen:

```

double distanceToSnow = Math.Sqrt(
    Math.Pow(snow.Blue - average.Blue, 2) +
    Math.Pow(snow.Green - average.Green, 2) +
    Math.Pow(snow.Red - average.Red, 2));
double distanceToNotSnow = Math.Sqrt(
    Math.Pow(normal.Blue - average.Blue, 2) +
    Math.Pow(normal.Green - average.Green, 2) +
    Math.Pow(normal.Red - average.Red, 2));

```

Code 9: Berechnung der Euklidischen Distanz zu den Referenzfarbwerten

Wenn die Durchschnittsfarbe von mehr als der Hälfte aller Regionen näher am Referenzwert für die Kategorie (Schnee oder kein Schnee) liegt, wird dem Bild die entsprechende Kategorie zugewiesen.

Da über das betrachtete Bild nichts über Licht- und Wetterverhältnisse bekannt ist, kann nur ein Vergleich zu globalen Referenzwerten gezogen werden. Für die Referenzwerte wurden für jede Kamera Bilder ausgewählt, welche die Kategorien 'es liegt Schnee' und 'es liegt kein

<sup>6</sup> Euklidischer Abstand: [https://de.wikipedia.org/wiki/Euklidischer\\_Abstand](https://de.wikipedia.org/wiki/Euklidischer_Abstand)

Schnee' gut repräsentieren und nicht von Licht- und Wetterverhältnissen signifikant beeinflusst sind. Die Bilder wurden auf Regionen aufgetrennt. Pro Region konnte der Durchschnittswert ausgerechnet und in der Datenbank abgelegt werden.

### 3.6 Resultate

Ursprünglich war nicht bekannt wie viel Zeit die Verarbeitung eines Bildes in Anspruch nimmt. Im Projektverlauf wurde jedoch klar, dass die Verarbeitungszeit nicht signifikant ist. Die Detektion für alle Kameras dauert nur einige Sekunden und stellt für einen potentiellen Live-Betrieb keine Einschränkung dar.

Die Schneedetektion liefert nicht genügend gute Ergebnisse. Es gibt keine Möglichkeit die Referenzwerte dynamisch auf den Bildinhalt anzupassen. Häufig wird ein Bild zur Schnee-Kategorie zugewiesen, obwohl kein Schnee liegt, dafür beispielsweise Nebel die Umgebung viel weißer erscheinen lässt. Abgesehen von Nebel gibt es weitere Situationen in denen die Farben verfälscht sind:

- Dämmerung: Veränderte Farben, lange Schatten und direkte Sonneneinstrahlung in die Linse verändern die Farben aller Bildregionen
- Nachts wird nur ein schwarz-weißes Bild aufgenommen
- Bei Regenwetter können Scheinwerfer der Autos und die Sonne Reflektionen auf der nassen Fahrbahn entstehen lassen

Der Fokus in der Bachelorarbeit wurde aus diesen Gründen auf folgende Punkte gelegt:

- Referenzwerte für verschiedene Licht- und Wetterverhältnisse herstellen
- Systematisches testen des Algorithmus ermöglichen
- Eine Stichprobe erstellen, bei der zu jedem Bild die 'Wahrheit' bekannt ist: Zu jedem Bild sollen Kategorien erfasst werden, wie Schnee-, Licht- und Wetterverhältnisse. Die Kategorien sollen zum Bild in der Datenbank abgespeichert werden
- Weitere statistische Werte einbeziehen wie Histogramm, Kontrast und Farbspektrum. Nicht nur die Durchschnittsfarbe
- Konzepte aus Machine Learning für die Detektion verwenden
- Kleinere Regionen (Patches) betrachten und nicht das gesamte Bild in grosse Segmente einteilen. Für die ASTRA ist hauptsächlich interessant, ob unmittelbar neben der Strasse Schnee liegt. Es sollen Patches in Strassennähe betrachtet werden, auf welchen Schnee längere Zeit liegen bleibt.

### 3.7 Weitere Themen

Im Verlauf der Projektarbeit wurden noch weitere Themen bearbeitet, welche schliesslich keine nennenswerten Resultate geliefert haben:

#### 3.7.1 Wetterdaten

Einbezug von historischen und aktuellen Wetterdaten: Um dynamisch entscheiden zu können welche Licht- und Wetterverhältnisse auf dem Bild zu erwarten sind, sollen Wetterdaten von MeteoSchweiz berücksichtigt werden. Von Interesse wäre beispielsweise lokale Temperatur, Niederschlagsmenge und Luftfeuchtigkeit.

Leider betreibt das Bundesamt für Meteorologie keine öffentliche Programmschnittstelle. Es existieren zwar öffentliche Schnittstellen (wie OpenWeatherMap.org<sup>7</sup>), diese bieten aber Datenabfragen ab einer bestimmten Menge nur gegen ein Entgelt an. Die einzige Möglichkeit auf diese Daten zuzugreifen ist eine Applikation, die von der Abteilung für Extremwertanalyse der Fachhochschule entwickelt und betrieben wird. Diese bietet eine Exportfunktion aber keine Programmschnittstelle.

Ein beträchtlicher Teil der Projektzeit wurde in das Einarbeiten in die Applikation (DataViewer) investiert. Aus Zeitgründen wurde das Thema aber nicht weiterverfolgt.

<sup>7</sup> Programmschnittstelle von Openweathermap.org: <http://openweathermap.org/api>

### **3.7.2 Intelligenter Cron-Job**

Der Cron-Job sollte so erweitert werden, dass der Bilddownload häufiger als alle zehn Minuten geschieht. Gleichzeitig soll ein Vorverarbeitungs-Job die Bilder bereits auf dem Server von Autos bereinigen. Dies wurde nicht weiterverfolgt, weil nicht genug Kenntnisse über Batch-Skripts vorhanden sind.

## 4 Bachelorarbeit

### 4.1 Verbesserung zu Projekt 2

Im Projekt 2 konnte viel Wissen über den Datenstand und das Handwerk angeeignet werden. Nun aber sollen sich die Anstrengungen vermehrt auf die Verbesserung der Schneedetektion richten. Die Erkennungsrate des Algorithmus soll messbar und zuverlässiger sein als der naive Ansatz aus der Projektarbeit. Hierzu sollen Informationen aus dem Bildarchiv gewonnen werden, die für den Computer verständlich sind:

Zu jedem Bild soll in der Datenbank gespeichert sein, ob neben der Strasse Schnee liegt oder nicht. Mit dieser Zusatzinformation kann die Güte eines neuen Algorithmus gemessen werden (F-Test). Weiter sollen auch Licht- und Wetterverhältnisse auf dem Bild bekannt sein. So wird es möglich, dass abhängig vom Bildinhalt mit adäquaten Referenzwerten verglichen wird.

### 4.2 F-Test

Um den Algorithmus systematisch testen zu können, kommt der F-Test zum Zug. Dieser Test kann eine Aussage über die Qualität eines Klassifikators machen (Hudritsch, 2016)F. Dabei ist der Klassifikator ein Verfahren, das Objekte zu verschiedenen Klassen zuordnet. In unserem Fall gibt es die Klassen ‘Schnee’ und ‘kein Schnee’.

Bei der Klassifizierung können Fehler passieren: Entweder wird einem Schnee-Bild die Kategorie ‘kein Schnee’ oder einem schneefreien Bild die Kategorie ‘Schnee’ zugewiesen. Daraus entstehen vier verschiedene Fälle:

	Es liegt Schnee	Es liegt kein Schnee
Test positiv	Richtig positiv ( $r_p$ )	Falsch positiv ( $f_p$ )
Test negativ	Falsch negativ ( $f_n$ )	Richtig negativ ( $r_n$ )

Tabelle 1: Mögliche Resultate einer Klassifizierung

Nun können nach der Durchführung einer Klassifizierung des Datenstandes, die richtig und falsch eingeordneten Bilder aufgezählt werden. Aus diesen Zahlen lassen sich drei Kennwerte berechnen:

$$\text{Sensitivität} = \frac{r_p}{r_p + f_n}$$

$$\text{Genauigkeit} = \frac{r_p}{r_p + f_p}$$

Die Sensitivität ist das Verhältnis zwischen den richtig detektierten Objekten zu allen positiv klassifizierten Objekten. Während die Genauigkeit ein Verhältnis zwischen den richtig detektierten und allen tatsächlich positiven Objekten ist. Aus diesen zwei Kennzahlen bildet sich der F-Wert:

$$F = 2 * \frac{\text{Sensitivität} * \text{Genauigkeit}}{\text{Sensitivität} + \text{Genauigkeit}}$$

Der Wertebereich aller drei Zahlen ist zwischen 0 und 1. Je höher die Werte sind, desto besser ist der Klassifikator.

### 4.3 Stichprobe aufbauen

Damit der Klassifikator prüfbar ist, muss also die Wahrheit bekannt sein. Das bedeutet zu möglichst jedem Bild aus dem Bildarchiv muss erfasst werden, ob auf dem Bild Schnee liegt oder nicht. Dafür wird die bestehende Applikation erweitert mit einer ‘Klassifizierung per Hand’.

### 4.3.1 Kategorien

Neben den zwei offensichtlichen Kategorien ‘Schnee’ und ‘kein Schnee’, gibt es weitere Fälle, bei denen der Bildinhalt beeinflusst ist. Abhängig vom Sonnenstand und den Wetterverhältnissen verändern sich die abgebildeten Farben und Sichtverhältnisse. Es kamen insgesamt folgende Bild-Kategorien zum Vorschein:



Abbildung 21: Kategorie 'kein Schnee'



Abbildung 22: Kategorie 'Schnee'



Abbildung 23: Kategorie 'Tag'



Abbildung 24: Kategorie 'Dämmerung'  
(Morgen oder Abend)



Abbildung 25: Kategorie 'Nacht'



Abbildung 26: Kategorie 'Nebel'



Abbildung 27: Kategorie  
'Niederschlag'



Abbildung 28: Kategorie 'schlechte  
Lichtverhältnisse' (Reflektionen oder  
direkte Sonneneinstrahlung)

Zudem können auch mehrere Beeinträchtigungen gleichzeitig eintreffen. So können zum Beispiel Scheinwerfer auf einer regennassen Strasse Reflektionen verursachen oder Sonneneinstrahlung bei Nebel am Morgen das Bild heller erscheinen lassen. Dies macht die naive Detektion mit dem Vergleich zu globalen Referenzwerten zu fehleranfällig.

### 4.3.2 Programm

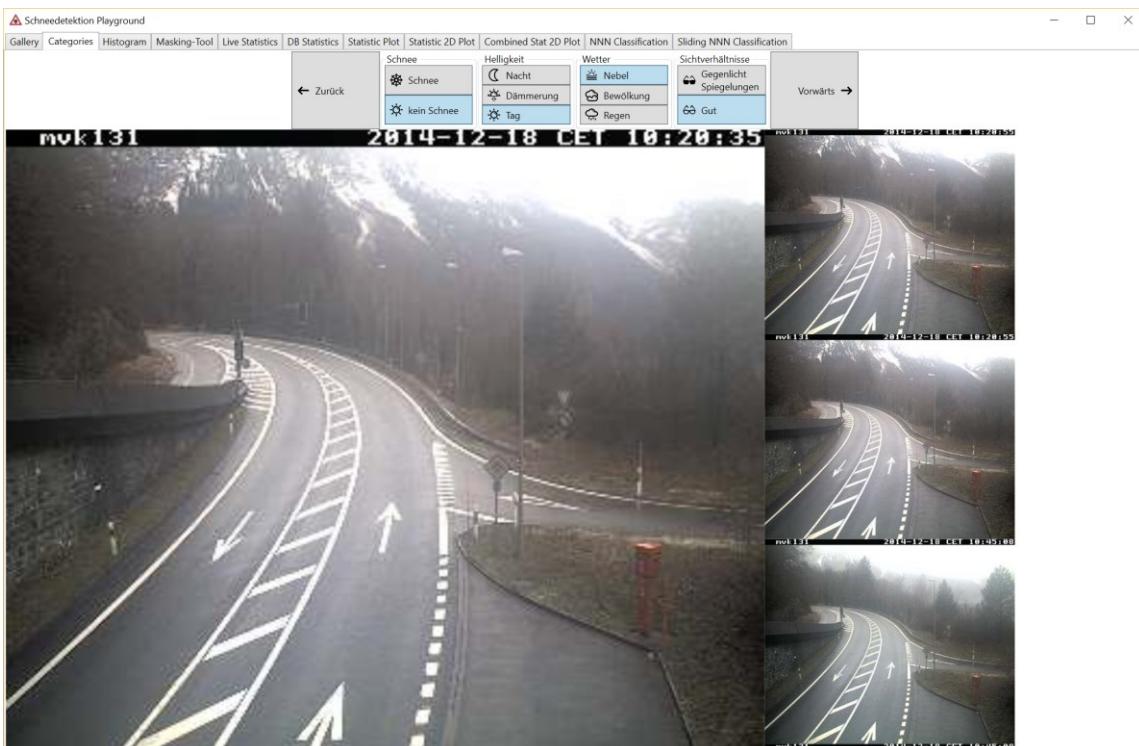


Abbildung 29: Bildschirmfoto des Kategorieerfassungsmoduls

Um den Aufbau der Stichprobe in einer möglichst kurzen Zeit zu bewältigen, wurde die Applikation um ein ‘Kategorisierungs-Modul’ erweitert. In diesem Modul werden Bilder in ihrer Aufnahmereihenfolge geladen und eines nach dem anderen gross angezeigt. Daneben werden nachfolgende Bilder angezeigt, die einen Überblick geben, wie sich die Sichtverhältnisse in den darauffolgenden Minuten verändert haben.

Mit den Knöpfen oberhalb der Bilder lässt sich die Situation im Bild auf die Kategorien verteilen und das Bild wechseln. Um die Bilder schneller durcharbeiten zu können ist das Modul vollständig mit der Tastatur bedienbar:

- Vorwärts und Zurück auf der Zeitachse ist auf der rechten beziehungsweise linken Pfeiltaste abgebildet. Bei jedem Wechsel auf das nächste oder vorangehende Bild, wird der definierte Zustand zum Datensatz gespeichert.
- Nummerntaste 1 schaltet zwischen ‘Schnee’ und ‘kein Schnee’ um
- Nummerntaste 2 wechselt periodisch die Tageszeitoptionen:  
Nacht => Dämmerung => Tag => Dämmerung => Nacht => ...
- Nummerntasten 3, 4 und 5 aktivieren und deaktivieren die Nebel-, Bewölkungs- und Regen-Option
- Nummerntaste 6 schaltet zwischen guten und schlechten Lichtverhältnissen um

Auf Datenbankebene wurde die Bild-Tabelle so erweitert, dass neben dem Dateipfad, der Kamera und dem Aufnahmzeitpunkt auch die Kategoriezugehörigkeit als Wahrheitswert (Boolean / Bit) gespeichert werden kann:

```
CREATE TABLE [dbo].[Images] (
    [ID]          INT            IDENTITY (1, 1) NOT NULL,
    [Name]         NVARCHAR (50)  NOT NULL,
    [Place]        NVARCHAR (50)  NOT NULL,
    [DateTime]    DATETIME       NOT NULL,
    [TimeZone]    NVARCHAR (5)   NULL,
    [UnixTime]    FLOAT (53)     NULL,
    [Snow]         BIT            NULL,
    [NoSnow]       BIT            NULL,
```

```

[Night]      BIT      NULL,
[Dusk]       BIT      NULL,
[Day]        BIT      NULL,
[Foggy]      BIT      NULL,
[Cloudy]     BIT      NULL,
[Rainy]       BIT      NULL,
[BadLighting] BIT      NULL,
[GoodLighting] BIT      NULL,
PRIMARY KEY CLUSTERED ([ID] ASC)
);

```

Code 10: Erweiterung der Images-Tabelle um Kategorien

#### 4.3.3 Anpassungen Galerie Modul

Für die stichprobenartige Kontrolle und allfällige Korrektur der gespeicherten Kategorien wurde das Galerie-Modul um zwei Funktionen erweitert: Einerseits ein Kontextmenü welches die zugewiesenen Kategorien anzeigt und anpassbar macht. Andererseits ein Filter mit dem sich die Bildliste neben Kamera und Datum auch auf Kategorien einschränken lässt.

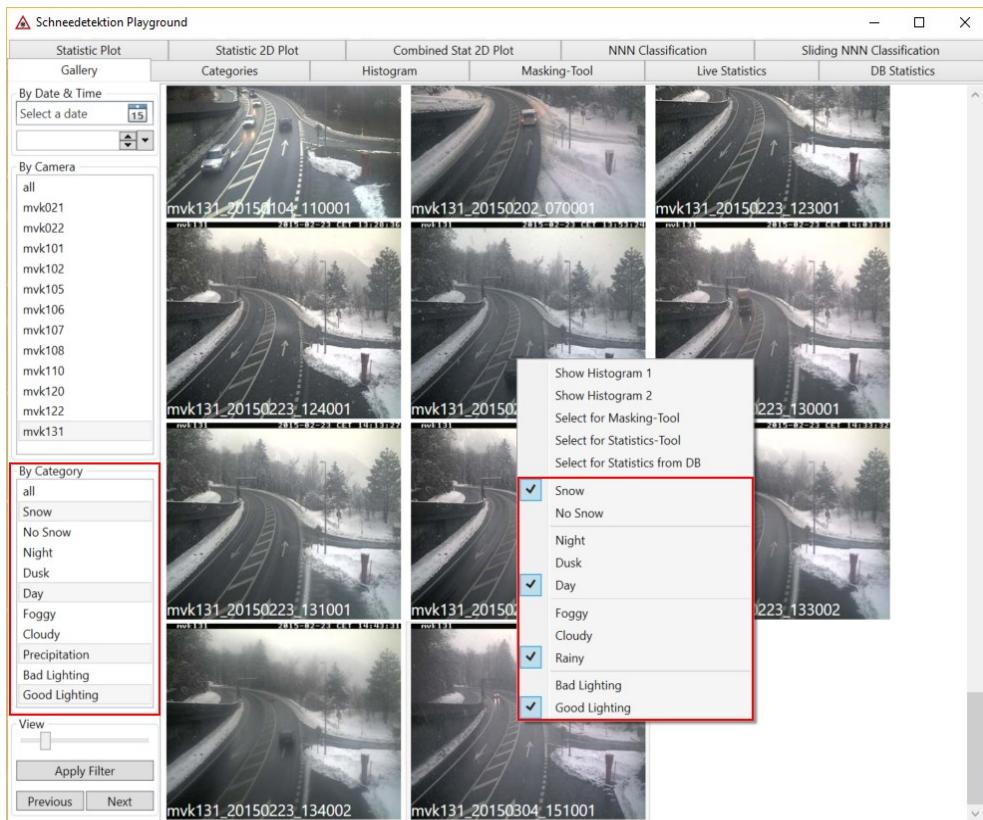


Abbildung 30: Bildschirmfoto des Galeriemoduls mit Kategorieerweiterungen

#### 4.3.4 Erfassung der Kategorien

Das Kategorisierungs-Modul ermöglichte es die ca. 400'000 Bilder zügig durchzuarbeiten. Im Schnitt wurde pro Bild weniger als eine halbe Sekunde aufgewendet. Das bedeutet, dass die Erfassung der Kategorien für das gesamte Bildarchiv etwa 45 Arbeitsstunden im Oktober in Anspruch nahm.

#### 4.3.5 Verhältnis

Aus den nun verfügbaren Daten können nun die ersten einfachen statistischen Analysen erstellt werden. So kann ein Überblick über die absolute Anzahl von Schneebildern und das Verhältnis zu den schneefreien Aufnahmen erstellt werden:

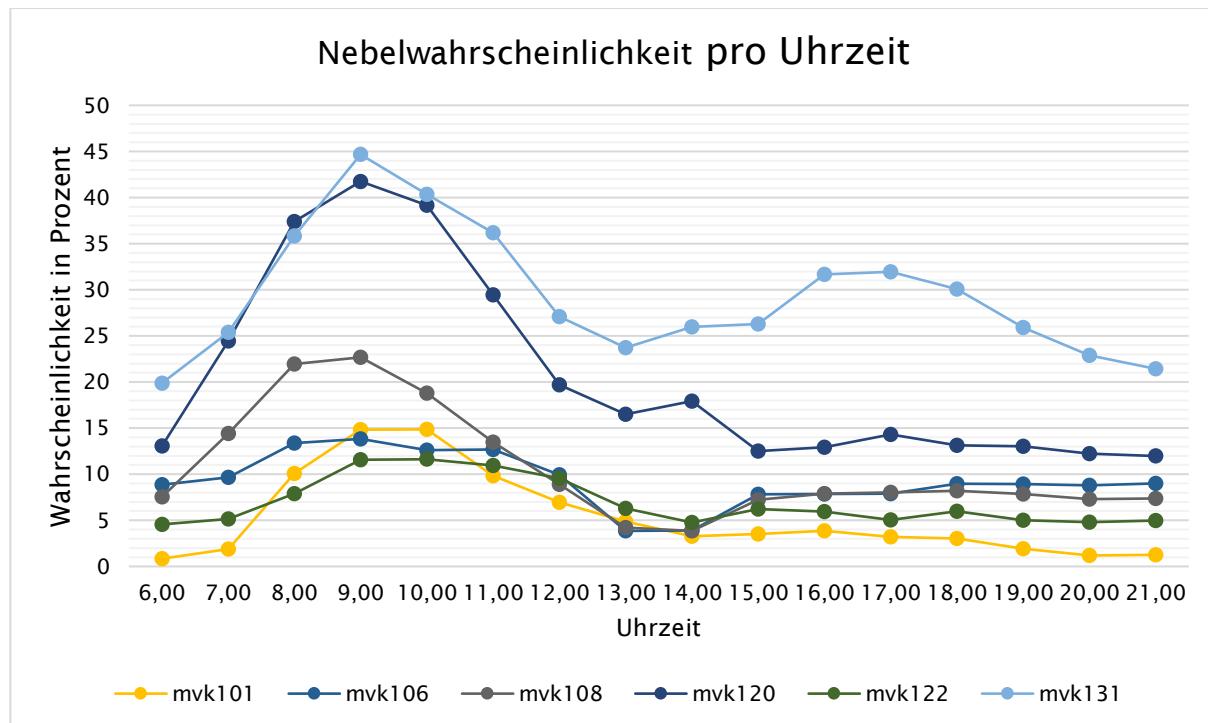
Kamera	Bilder insgesamt	Bilder ohne Schnee	Anteil ohne Schnee (%)	Bilder mit Schnee	Anteil mit Schnee (%)
mvk021	37894	32264	85,14	5630	14,86
mvk022	15760	14020	88,96	1740	11,04
mvk101	34145	29396	86,09	4749	13,91
mvk102	19377	18344	94,67	1033	5,33
mvk105	27815	24385	87,67	3430	12,33
mvk106	21100	19599	92,89	1501	7,11
mvk107	31949	28077	87,88	3872	12,12
mvk108	21114	19148	90,69	1966	9,31
mvk110	51763	30066	58,08	21697	41,92
mvk120	54118	34734	64,18	19384	35,82
mvk122	51232	43172	84,27	8060	15,73
mvk131	51844	29166	56,26	22678	43,74

Tabelle 2: Verhältnis zwischen Schneebildern und schneefreien Bildern pro Kameraperspektive

Es ist sichtbar, dass für drei Kameras (mvk110, mvk120, mvk131) das Schneeverhältnis deutlich höher ist als bei den anderen. Diese Kameras befinden sich auf der A8, die Strasse zwischen Brienzsee und Brünigpass-Passhöhe. Diese befinden sich auf einer höheren Lage und in einer Region, in welcher hohe Berge im Tagesverlauf länger Schatten auf die Strasse werfen. Hier bleibt der Schnee länger liegen.

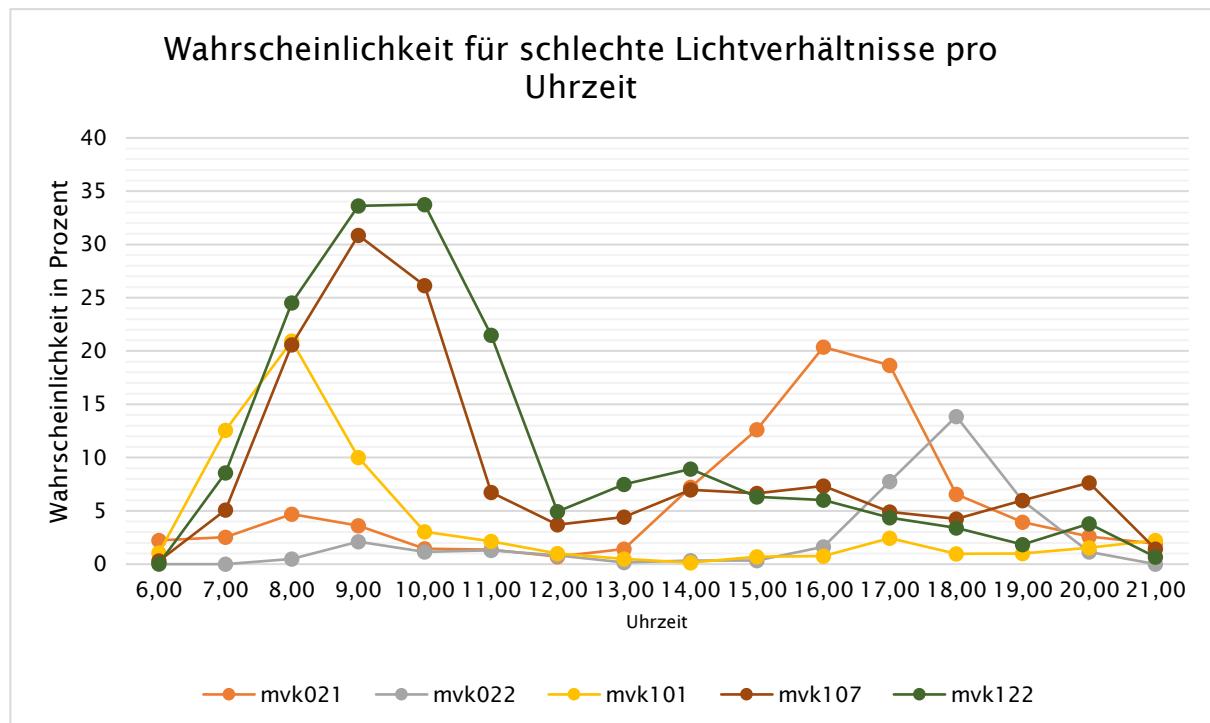
#### 4.3.6 Wahrscheinlichkeit für eingeschränkte Sichtverhältnisse

Beim Erfassen der Kategorien ist aufgefallen, dass eingeschränkte Sichtverhältnisse pro Kamera immer wieder zu gleichen Uhrzeiten auftreten. Dies bestätigt sich bei genauerer Betrachtung der Zahlen:



In der Winterzeit ist die Wahrscheinlichkeit für eine Nebellage für alle Kameras zwischen 8 und 10 Uhr morgens am höchsten. Hier stechen zwei Kameras (mvk120 und mvk131) auf

der A8-Route besonders heraus: Hier ist die Wahrscheinlichkeit für Nebel um 9 Uhr morgens mit über 40 Prozent am höchsten.



Abhängig davon wie die Kamera platziert ist, ist die Wahrscheinlichkeit für schlechte Lichtverhältnisse zu gleichen Uhrzeiten höher: Kameras die nach Osten ausgerichtet sind und teilweise den Himmel im Bild haben, sind am Morgen von der direkten Sonneneinstrahlung beeinflusst (mvk101, mvk107 und mvk122). Dasselbe gilt für Kameras mit Ausrichtung nach Westen – nur scheint da die Abendsonne in die Kamera (mvk021 und mvk022).

#### 4.4 Neue Segmentierung (Patches)

Im Verlauf des Projekt 2 stellte sich heraus, dass es für die Detektion keinen Sinn macht das gesamte Bild in grosse Segmente zu unterteilen. Die farbliche Variation in den grossen Segmenten ist viel zu gross für eine Detektion. Das Interesse der Auftraggeber konzentriert sich auf die Information, ob unmittelbar neben der Strasse Schnee liegt oder nicht. Aus der näheren Betrachtung fällt darum die Fahrbahn heraus – hier bleibt kein Schnee liegen. Die Räumfahrzeuge säubern schneebedeckte Strassen innerhalb von Stunden. Auch Hintergrund des Bildes und der Himmel sind für eine Detektion nicht nützlich.

Die Regionen wurden mit dem bestehenden Polygon-Editor neu definiert und fortan als Patches (Flickstellen) bezeichnet. Diese decken Regionen ab, auf denen Schnee lange nach Schneefall liegen bleibt: Mittel-, Pannenstreifen und Rasenflächen neben der Fahrbahn.



Abbildung 31: Neue Segmentierung für mvk110

Für die Kamera mvk101 wurden beispielsweise fünf Patches definiert: Drei Rasenflächen um die Autobahneinfahrt, ein Stück zwischen Leitplanken und ein Strassenstück mit Markierungen.

#### 4.5 Statistische Werte pro Patch

Eine weitere Lehre aus der Projektarbeit war, dass allein der Durchschnittsfarbwert für die Detektion nicht ausreicht. Aus den Bildern und Bildausschnitten lassen sich noch viele weitere Informationen extrahieren. Hierzu unterziehen wir die Bilder einer statistischen Auswertung.

##### 4.5.1 Histogramm

Die zu analysierenden Bilder im Archiv sind normale JPEG-Dateien mit einer Farbtiefe von 24 Bit. Das bedeutet, dass jeder Bildpunkt aus einer roten, blauen und grünen Farbkomponente besteht. Jeder Komponentenwert ist in 8 Bit codiert. Jede Farbkomponente kann so Werte zwischen 0 und 255 annehmen. Je höher der Wert desto intensiver ist die Farbe. Die Tabelle unten zeigt alle Farben pro Kanal, die eine 8-Bit-Farbe annehmen kann:

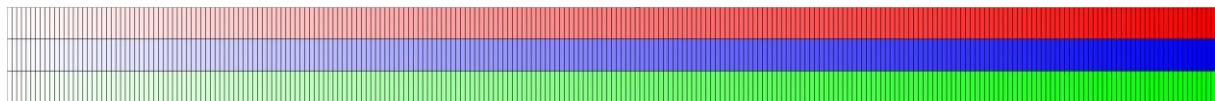


Abbildung 32: Alle möglichen 8 Bit Farben (rot, blau und grün)

Das Histogramm ist eine diskrete Häufigkeitsverteilung von Farbwerten in einem Bild (Hudritsch, 2016). Um das Histogramm eines Bildes zu generieren, muss pro Bildpunkt und pro Farbkanal aufgezählt werden, wie häufig jeder der 256 verschiedenen Werte im Bild vorkommt. Die Aufzählung ist im Programm mit drei Listen und zwei For-Schleifen gelöst:

```
private Statistic GetStatistic(Image<Bgr, byte> image, Image<Gray, byte> bitmask)
{
    // Pro Kanal eine leere Liste der Länge 256 erstellen
    List<double> blueHistogram = new List<double>();
    blueHistogram.AddRange(new double[256]);
    List<double> greenHistogram = new List<double>();
    greenHistogram.AddRange(new double[256]);
    List<double> redHistogram = new List<double>();
    redHistogram.AddRange(new double[256]);

    // Für alle Spalten im Bild
    for (int i = 0; i < image.Cols; i++)
    {
        // Für alle Zeilen im Bild
        for (int j = 0; j < image.Rows; j++)
        {
            // Wenn keine Bitmaske angegeben wurde,
            // oder die Bitmaske an dieser Stelle schwarz ist
            if (bitmask == null || bitmask[j, i].MCvScalar.V0 == 0)
```

```

    {
        // Histogram pro Kanal abfüllen:
        // Listen-Element im Farbkanal inkrementieren,
        // dass dem Farbwert des aktuellen Bildpunkts entspricht
        blueHistogram[(int)image[j, i].MCvScalar.V0]++;
        greenHistogram[(int)image[j, i].MCvScalar.V1]++;
        redHistogram[(int)image[j, i].MCvScalar.V2]++;
    }
}
[...]

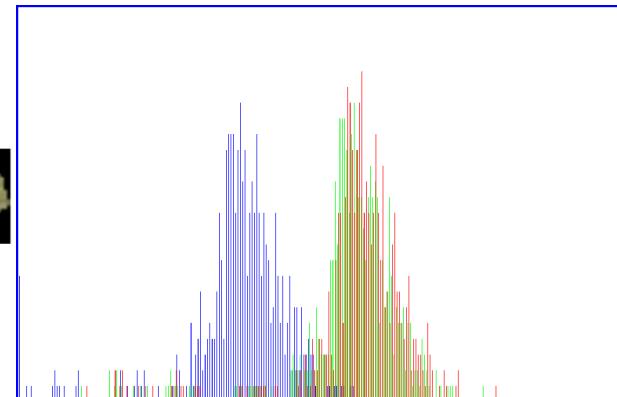
```

Code 11: Berechnung eines Histogramms

Die resultierenden Häufigkeitswerte in den drei Listen lassen sich nun in Balkendiagrammen darstellen. Stichprobenartig wurden Bilder einer Kameraperspektive bei unterschiedlichen Schneeverhältnissen verglichen. Eine erste Tendenz konnte so beobachtet werden. Das Histogramm eines Patches mit liegendem Schnee ist meist deutlich nach rechts verschoben und hat eine kleinere Streuung (höhere Varianz) als Patches ohne Schnee:



Abbildung 33: Ausgeschnittenes Bildsegment von mvk110 ohne Schnee



Histogramm des Bildsegments ohne Schnee



Abbildung 34: Ausgeschnittenes Bildsegment von mvk110 mit Schnee



Histogramm des Bildsegments mit Schnee

#### 4.5.2 Modalwert

Der Modalwert ist der häufigste Farbwert im Histogramm. Dieser kann Werte zwischen 0 und 255 annehmen. Dieser Wert kann pro Farbkanal einfach aus dem entsprechenden Histogramm herausgelesen werden:

```
// Mode: Farbwert (Index) des grössten Werts im Histogramm
statistic.ModeBlue = blueHistogram.IndexOf(blueHistogram.Max());
statistic.ModeGreen = greenHistogram.IndexOf(greenHistogram.Max());
statistic.ModeRed = redHistogram.IndexOf(redHistogram.Max());
```

Code 12: Berechnung des Modalwerts

#### 4.5.3 Durchschnittsfarbe (Mittelwert)

Als weiteren Kennwert eines Patches wird die Durchschnittsfarbe berechnet. Um den arithmetischen Mittelwert zu berechnen, werden alle Farbwerte pro Kanal  $g(x, y)$  aufsummiert und durch die Gesamtanzahl ( $N$ ) von Bildpunkten dividiert:

$$\bar{g} = \frac{1}{N} \sum_{x=0}^{n-1} x \sum_{y=0}^{m-1} y g(x, y)$$

Hierzu werden zunächst alle Farbwerte pro Kanal aus dem zweidimensionalen Pixel-Array in eine eindimensionale Liste kopiert. Danach kann der Wert mit einem Aufruf ausgelesen werden:

```
// Eindimensionale Liste aller Pixel im Bild
List<double> bluePixels = new List<double>();
List<double> greenPixels = new List<double>();
List<double> redPixels = new List<double>();

// Für alle Spalten im Bild
for (int i = 0; i < image.Cols; i++)
{
    // Für alle Zeilen im Bild
    for (int j = 0; j < image.Rows; j++)
    {
        // Wenn keine Bitmaske angegeben wurde,
        // oder die Bitmaske an dieser Stelle schwarz ist
        if (bitmask == null || bitmask[j, i].MCvScalar.V0 == 0)
        {
            // Pixel in entsprechende Farbkanal-Liste abspitzen
            bluePixels.Add(image[j, i].MCvScalar.V0);
            greenPixels.Add(image[j, i].MCvScalar.V1);
            redPixels.Add(image[j, i].MCvScalar.V2);
        }
    }
}

// Mean
statistic.MeanBlue = bluePixels.Average();
statistic.MeanGreen = greenPixels.Average();
statistic.MeanRed = redPixels.Average();
```

Code 13: Berechnung des Mittelwerts

#### 4.5.4 Median

In einer sortierten Auflistung ist der Median der Wert, welcher in der Mitte der Liste steht. Der Unterschied zwischen Mittelwert und Median kann bezüglich der Verteilung der Werte im Histogramm aussagekräftig sein. Um diesen Wert zu berechnen, muss die Liste zunächst sortiert werden. Danach kann der Wert in der Mitte ausgelesen werden:

```
// Median
bluePixels.Sort();
greenPixels.Sort();
redPixels.Sort();
int middle = bluePixels.Count / 2;
if (bluePixels.Count % 2 == 0) // Gerade Anzahl
{
    statistic.MedianBlue =
        (bluePixels.ElementAt(middle) + bluePixels.ElementAt(middle - 1)) / 2d;
    statistic.MedianGreen =
        (greenPixels.ElementAt(middle) + greenPixels.ElementAt(middle - 1)) / 2d;
    statistic.MedianRed =
        (redPixels.ElementAt(middle) + redPixels.ElementAt(middle - 1)) / 2d;
}
else // Ungerade Anzahl
{
    statistic.MedianBlue = bluePixels.ElementAt(middle);
    statistic.MedianGreen = greenPixels.ElementAt(middle);
    statistic.MedianRed = redPixels.ElementAt(middle);
```

Code 14: Berechnung des Medianwerts

#### 4.5.5 Varianz und Standardabweichung

Diese beiden Kennwerte sagen, aus wie stark der ‘Histogramm-Hügel’ gestreut ist. Es wird gemessen wie stark jeder Farbwert pro Kanal vom Mittelwert abweicht. Je grösser dieser Wert, desto grösser ist die Streuung und desto flacher ist der Hügel (Häufigkeitsverteilung der Farbwerte im Histogramm). Für die Varianz werden die Differenzen aller Farbwerte zum Mittelwert ( $\bar{g}$ ) berechnet, quadriert, aufsummiert und durch die Gesamtanzahl ( $N$ ) von Bildpunkten dividiert:

$$\text{Varianz: } var = \frac{1}{N} \sum_{x=0}^{n-1} \sum_{y=0}^{m-1} (g(x,y) - \bar{g})^2$$

Die Standardabweichung ( $\sigma$ ) ist definiert als Wurzel aus der Varianz:

$$\text{Standardabweichung: } \sigma = \sqrt{var}$$

Für die Berechnung der Standardabweichung wird die bereits erstellte eindimensionale Liste mit Farbwerten und der berechnete Mittelwert genutzt:

```
// Variance
statistic.VarianceBlue = bluePixels.Sum(i => Math.Pow(i - statistic.MeanBlue, 2))
    / (double)bluePixels.Count;
statistic.VarianceGreen = greenPixels.Sum(i => Math.Pow(i - statistic.MeanGreen, 2))
    / (double)greenPixels.Count;
statistic.VarianceRed = redPixels.Sum(i => Math.Pow(i - statistic.MeanRed, 2))
    / (double)redPixels.Count;

// Standard Deviation
statistic.StandardDeviationBlue = Math.Sqrt(statistic.VarianceBlue);
statistic.StandardDeviationGreen = Math.Sqrt(statistic.VarianceGreen);
statistic.StandardDeviationRed = Math.Sqrt(statistic.VarianceRed);
```

Code 15: Berechnung der Standardabweichung

#### 4.5.6 Kontrast, Minimum und Maximum

Der Kontrast ( $C$ ) pro Farbkanal ist definiert als Differenz zwischen Maximal- ( $g_{max}$ ) und Minimalwert ( $g_{min}$ ), dividiert durch die Summe von eben diesen Werten:

$$\text{Kontrast: } C = \frac{g_{max} - g_{min}}{g_{max} + g_{min}}$$

Diese Werte lassen sich ebenfalls einfach aus der eindimensionalen Liste von Farbwerten auslesen:

```
// Minimum
statistic.MinimumBlue = bluePixels.Min();
statistic.MinimumGreen = greenPixels.Min();
statistic.MinimumRed = redPixels.Min();

// Maximum
statistic.MaximumBlue = bluePixels.Max();
statistic.MaximumGreen = greenPixels.Max();
statistic.MaximumRed = redPixels.Max();

// Contrast
statistic.ContrastBlue = (statistic.MaximumBlue - statistic.MinimumBlue)
    / (statistic.MaximumBlue + statistic.MinimumBlue);
statistic.ContrastGreen = (statistic.MaximumGreen - statistic.MinimumGreen)
    / (statistic.MaximumGreen + statistic.MinimumGreen);
statistic.ContrastRed = (statistic.MaximumRed - statistic.MinimumRed)
    / (statistic.MaximumRed + statistic.MinimumRed);
```

Code 16: Berechnung des Minimums, des Maximums und des Kontrasts

#### 4.5.7 Berechnung der Kennzahlen ohne OpenCV-Methoden

Die Berechnung der statistischen Werte und des Histogramms findet ohne Hilfe von scheinbar äquivalenten OpenCV-Methoden statt. Die Implementation ist aus zwei Gründen so umgesetzt:

- Alle Daten können in einer einzigen verschachtelten For-Schleife erfasst werden.
- Das Histogramm ist als Liste vorhanden. Auf dieser lassen sich die statistischen Werte mit C# und LINQ einfach und nachvollziehbar auslesen.

#### 4.5.8 Statistik Modul

Für die schnelle Analyse der nun berechenbaren statistischen Werte ist ein weiteres Modul in der Applikation entstanden: Dieses ermöglicht ein Bild aus der Galerie für Segmentierung und statistische Analyse auszuwählen. Auf Knopfdruck werden alle Werte berechnet und in einer übersichtlichen Form dargestellt:

```
// Statistiken für das gesamte Bild berechnen
Statistic completeImageStatistic =
    openCVHelper.GetStatisticForImage(imageViewModel.Image.FileName);

// Statistiken und Bild verpacken und darstellen
PatchViewModel completeImagePatchViewModel =
    new PatchViewModel(completeImageStatistic, imageViewModel);
patches.Add(completeImagePatchViewModel);

// Für jedes Patch des Bildes
foreach (var polygon in polygons)
{
    // Polygon des Patch auslesen
    IEnumerable<Point> pointCollection =
        PolygonHelper.DeserializePointCollection(polygon.PolygonPointCollection);

    // Bild ausschneiden und Statistiken für das Patch berechnen
    Statistic patchStatistic =
        openCVHelper.GetStatisticForPatchFromImagePath(
            imageViewModel.Image.FileName, pointCollection);

    // Patch-Bild generieren
    BitmapImage patchImage =
        openCVHelper.GetPatchBitmapImage(imageViewModel.Image.FileName, pointCollection);
```

```
// Patch-Statistiken und Patch-Bild verpacken und darstellen
PatchViewModel patchViewModel =
    new PatchViewModel(patchStatistic, patchImage, imageViewModel, polygon);

patches.Add(patchViewModel);
}
```

Code 17: Berechnung der statistischen Werte für eine Bild uns dessen Patches

Nach der Berechnung der Werte werden die berechneten Werte tabellarisch dargestellt. Auf der linken Seite des Fensters ist das gewählte Bild überlagert mit den vordefinierten Patches dargestellt. Auf der rechten Seite sind die Bildausschnitte und deren Kennzahlen dargestellt.

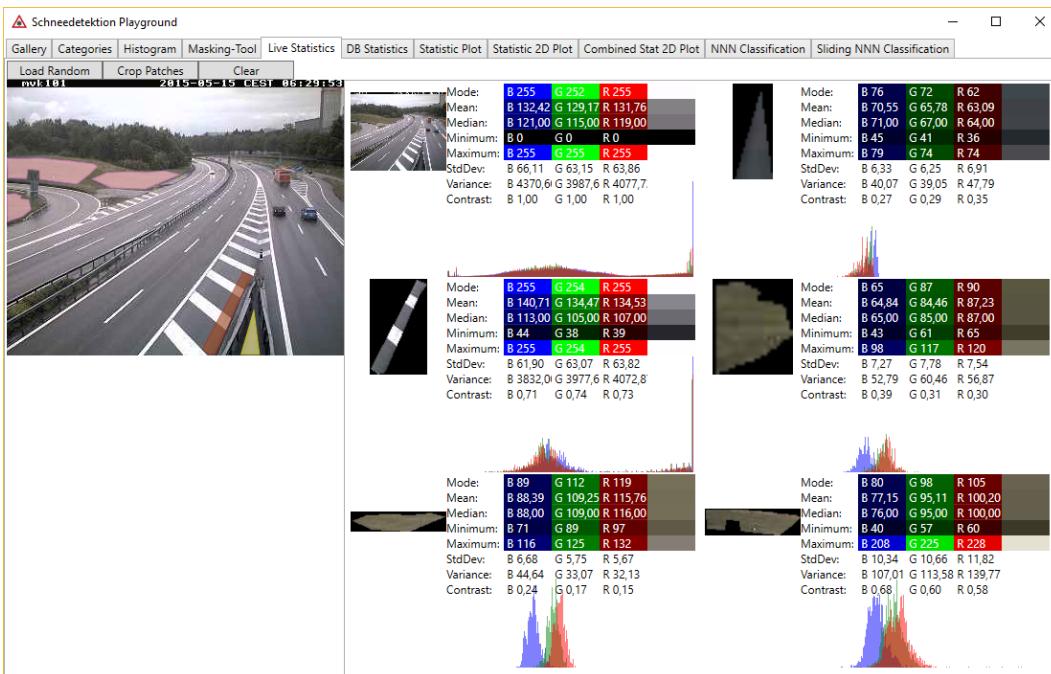


Abbildung 35: Bildschirmfoto des Statistikmoduls

Ein Listenelement besteht aus drei Teilen: Auf der linken Seite wird eine Vorschau des betrachteten Patch dargestellt. Links daneben ist eine vierstellige Tabelle der berechneten statistischen Werte. Die Zahlen in den ersten drei Spalten sind auf die drei Farbkanäle aufgeschlüsselt und unterlegt mit der Farbe, welche dem Farbwert entspricht. In der vierten Spalte ist eine Fläche eingefärbt; diese Farbe entspricht der Kombination der Werte in den ersten drei Spalten. Unterhalb der Tabelle ist das Histogramm des Bildausschnitts als kombiniertes Balkendiagramm dargestellt.

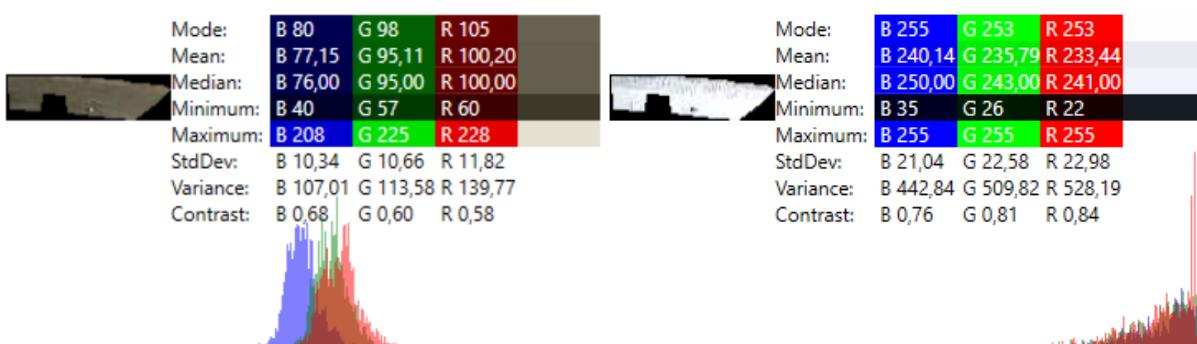


Abbildung 36: Berechnete statistische Werte eines Bildausschnitts ohne Schnee und mit Schnee

## 4.6 Statistische Werte in Datenbank ablegen

Die Kennzahlen der Bilder und Patches lassen sich nun schnell und einfach auslesen und vergleichen. Dies ist aber nur für einzeln ausgewählte Bilder aus der Galerie möglich. Um die Werte auf globaler Ebene genauer und systematischer betrachten zu können, müssen diese in einer anderen Form verfügbar sein. Aus diesem Grund soll ein grosser Teil der Daten in die Datenbank geladen werden. Hierzu wurde die Datenstruktur erweitert und ein Programm geschrieben, das systematisch Bilder durchgeht, die statistischen Werte der Patches ausliest und in die Datenbank ablegt.

### 4.6.1 Datenstruktur

Zu den bereits bestehenden Tabellen auf der Datenbank für Bilder (Images) und Patches (Polygons) kommen zwei weitere Strukturen hinzu: Die berechneten statistischen Werte können in die Tabelle ‘Statistics’ abgelegt werden: Diese beinhaltet jeweils für die drei Farbkanäle ein serialisiertes Histogramm und die statistischen Werte als Fliesskommazahlen. Eine weitere Tabelle (‘Entity\_Statistics’) stellt die Verbindung zwischen Bild, Patch und Statistiken her. Da sind die Laufnummern der drei verbundenen Strukturen abgelegt:

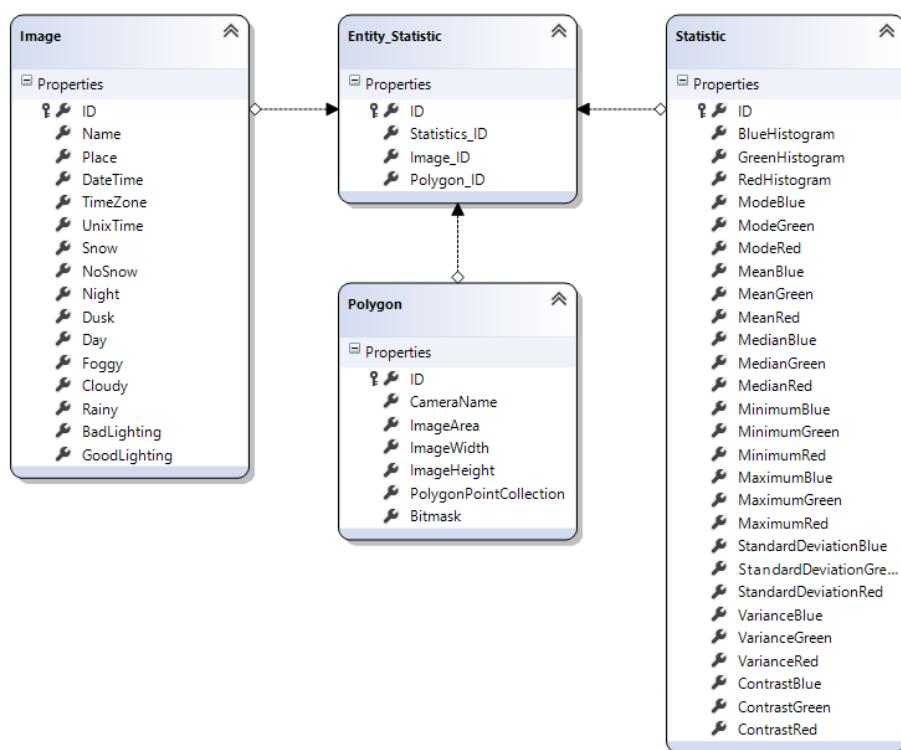


Abbildung 37: Klassendiagramm der erstellten Datenstrukturen

### 4.6.2 Programm

Das Auslesen und Abspeichern der statistischen Werte eines Patches ist unkompliziert und kann in vier Zeilen Code ausgeführt werden. Die eigentliche Herausforderung bei diesem Schritt ist es grosse Menge an Daten zu verarbeiten. Hierzu mussten zwei Massnahmen umgesetzt werden:

Als erstes werden Patches der Bilder in der Nacht und in der Dämmerung nicht berücksichtigt. Die Daten, die zu diesen Tageszeiten entstehen, sind nicht sehr aussagekräftig und würden die Daten teilweise verfälschen. Die Bilder in der Nacht sind nämlich nicht in Farbe, sondern nur in schwarz-weiss erfasst. Somit verkleinert sich die Menge der zu betrachtenden Bilder um über 50% auf knapp 200'000 Stück.

Die zweite Massnahme ist das schrittweise ablegen in die Datenbank. Bei Tests kam zum Vorschein, dass der Prozess stark beschleunigt werden kann, wenn nicht nach jeder Berechnung der statistischen Werte, sofort eine Zeile in die Datenbank geschrieben wird. Es

ist aber auch ineffizient alle Berechnungen für eine Kamera auszuführen und anschliessend alle Werte abzuspeichern. Dieses Vorgehen belastet die Datenbank stark und hält das Programm zu lange auf. Die Lösung hier ist die Bilder stapelweise zu verarbeiten. Als Stapelgrösse wurde 1000 Stück gewählt. Der Ablauf sieht so aus:

1. 1000 unverarbeitete Bilder laden
2. Statistische Werte für alle Patches der Bilder berechnen
3. In Datenbank ablegen

Pro Durchlauf werden so circa 4000 Zeilen geschrieben. Dies belastet die Datenbank nicht zu stark.

```
// Alle Patch-Definitionen der Kamera auslesen
IEnumerable<Polygon> polygons = dataContext.Polygons.Where(p => p.CameraName == camera);

// Alle Bilder (dieser Kameras, am Tag) auslesen
// deren Patch-Statistiken noch nicht berechnet wurden
// Davon nur die ersten 1000 (take) auslesen
IEnumerable<Image> images = (from i in dataContext.Images
                             where i.Place == camera
                             where i.Day == true
                             where (from es in i.Entity_Statistics
                                   where es.Polygon_ID != null
                                   select es).Count() == 0
                             select i).Take(take);

// Für alle geladenen Bilder der Kamera
foreach (var image in images)
{
    // Für jedes Patch der Kamera
    foreach (var polygon in polygons)
    {
        // Polygon des Patches auslesen
        var polygonPoints =
            JsonConvert.DeserializeObject<Media.PointCollection>(
                polygon.PolygonPointCollection);

        // Bild ausschneiden und Statistiken für das Patch berechnen
        Statistic imageStatistic =
            openCVHelper.GetStatisticForPatchFromImagePath(image.FileName, polygonPoints);

        // Statistische Werte in Datenstruktur ablegen
        image.Entity_Statistics.Add(new Entity_Statistic())
        {
            Statistic = imageStatistic,
            Polygon = polygon
        });
    }
}

// In Datenbank abspeichern
dataContext.SubmitChanges();
```

Code 18: Statistische Werte in Datenbank abspeichern

Die Verarbeitungszeit für alle Patches eines Bildes beläuft sich so auf etwa eine Zehntelsekunde. Um alle Bilder aller Kameras durchzuarbeiten wurden insgesamt 8 Stunden Rechenzeit investiert.

#### 4.6.3 Grösse der Datenbank

Mit der Berechnung und dem Speichern der statistischen Werte wuchs die Datenbank stark an. Vor der Erfassung der Patch-Statistiken beinhaltete die Datenbank nur die eingelesenen Meta-Informationen zu den Bildern und nahm etwa 80 Megabyte in Anspruch. Während der Erfassung wuchs die Datei schnell auf über mehrere Gigabyte Grösse an. Dies hatte zur Folge, dass die Datei nicht mehr in der Quellcodeverwaltung (GitHub) hochgeladen werden konnte. Die Onlineplattform akzeptiert bei kostenfreier Nutzung Dateien mit einer Maximalgrösse von 100 Megabyte<sup>8</sup>.

<sup>8</sup> Maximale Dateigrößen auf GitHub: <https://help.github.com/articles/what-is-my-disk-quota/>

Um weiterhin ein aktuelles Backup der Datei zu Verfügung zu haben, wurde die Datei in einen OneDrive-Ordner verschoben. Dies ist ein Cloud-Dienst, welcher Dateien auf dem Computer und dem Online-Speicher synchron hält. So könnte die Datenbank bei einem Datenverlust auf dem Computer aus dem Online-Speicher wiederhergestellt werden. OneDrive zeigte im Gebrauch einige signifikante Schwächen: Die Datenbank konnte während des Synchronisierungsprozesses nicht beschrieben und gelesen werden. Somit musste jeweils entweder die Synchronisierung oder die Entwicklungsumgebung ausgeschaltet werden. Der zweite Nachteil des Diensts war die Synchronisierungsdauer: Um die 1 Gigabyte grosse Datei hochzuladen brauchte es jeweils mehrere Stunden. Dieser Umstand behinderte die Weiterentwicklung stark.

Aus diesem Grund wurde die Datenbank wieder aus dem OneDrive-Verzeichnis entfernt. Im weiteren Verlauf der Arbeit wurde die Datenbank täglich per Hand auf eine externe Festplatte kopiert. Dies ist nicht zeitaufwändig und bietet eine ähnliche Sicherheit wie der Cloud-Speicher.

## 4.7 Plot Modul

Um die nun erfassten Daten analysieren zu können wurde die Applikation um ein Modul erweitert, dass die Daten visualisieren kann:

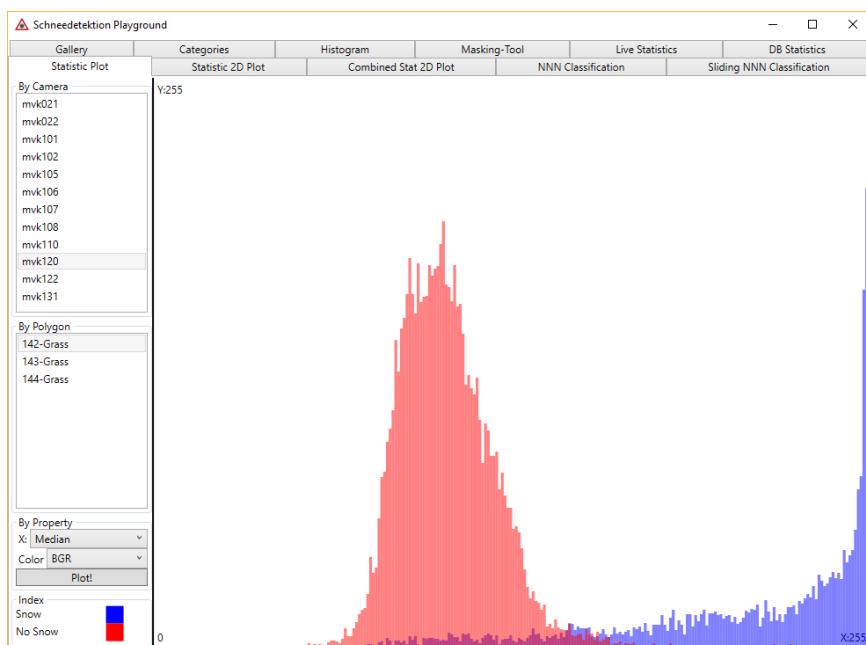


Abbildung 38: Bildschirmfoto des Plot Moduls.

Plot der Häufigkeitsverteilung Medianwerts (BGR) der Kamera mvk120 und des Patch 142

Auf der linken Seite der Benutzerschnittstelle lässt sich der gewünschte Datensatz für die Analyse auswählen: In der Kamera-Liste (oben) kann eine der Kameraperspektiven ausgewählt werden. Direkt unterhalb lädt die Applikation die Patches der gewählten Kamera. Hier muss ebenfalls eines ausgewählt werden. Unterhalb der Patch-Auswahl lässt sich auswählen welches Statistik-Attribut ins Diagramm geladen wird. Zur Auswahl stehen die gespeicherten statistischen Werte: Modalwert, Mittelwert, Median, Minimum, Maximum, Standardabweichung, Varianz und Kontrast. Weiterhin lässt sich bestimmen ob ein Farbkanal (blau, grün rot) gesondert oder alle drei Kanäle kombiniert betrachtet werden.

Das Resultat, links in der Zeichnungsebene, sind zwei Balkendiagramme, die die Häufigkeitsverteilungen des gewählten Attributs jeweils für Patches mit Schnee (blaue Balken) und Patches ohne Schnee (rote Balken) darstellen. Die Höhe der Balken ist abhängig von der Häufigkeit des Wertes (zwischen 0 und 256) des Statistik-Attributs. Die Balken sind zu fünfzig Prozent transparent, damit sichtbar wird, wie sich die beiden Verteilungen überlagern.

Im Programm sieht das so aus:

```
// Statistische Werte für die gewählte Kamera und den gewählten Patch mit Schnee auslesen
var statisticsWithSnow = from es in dataContext.Entity_Statistics
    where es.Image.Place == selectedCamera
    where es.Image.Snow == true
    where es.Polygon.ID == polygonID
    select es.Statistic;

// Balken für Statistiken mit Schnee zeichnen (Blau)
DrawBars(statisticsWithSnow, selectedAttribute, selectedColor, Brushes.Blue);

// Statistische Werte für die gewählte Kamera und den gewählten Patch ohne Schnee auslesen
var statisticsWithoutSnow = from es in dataContext.Entity_Statistics
    where es.Image.Place == selectedCamera
    where es.Image.Snow == false
    where es.Polygon.ID == polygonID
    select es.Statistic;

// Balken für Statistiken ohne Schnee zeichnen (Rot)
DrawBars(statisticsWithoutSnow, selectedAttribute, selectedColor, Brushes.Red);
```

Code 19: Generierung von Häufigkeitsverteilungen

In der Methode DrawBars werden die Werte des gewählten Attributs aus jedem Statistik-Objekt ausgelesen und auf 256 Balken akkumuliert:

```
// 256 leere Balken: Leere Liste der Länge 256 erstellen
List<double> bars = new List<double>();
bars.AddRange(new double[256]);

// Für alle Statistik-Objekte
foreach (var statistic in statistics)
{
    // 1. Wert aus dem Statistik-Objekt auslesen in der gewählten Farbe
    // 2. Werte bei Bedarf skalieren auf den Wertebereich [0,255]
    // 3. Auf die nächste Ganzzahl abrunden
    int flooredValue = (int)ScaleToCanvas(
        statistic.Get(selectedAttribute, selectedColor), selectedAttribute);
    // Den Balken in der Liste um 1 vergrößern,
    // dessen Position in der Liste dem Statistik-Wert entspricht
    bars[flooredValue]++;
}

// Die Breite eines Balkens ist abhängig von der aktuellen Breite der Zeichnungsebene.
double strokeWidth = plotCanvas.ActualWidth / 256;

// Für alle Balken in der Liste eine Linie in der Zeichnungsebene zeichnen
for (int i = 0; i < bars.Count; i++)
{
    // Balken zur Zeichnungsebene hinzufügen
    plotCanvas.Children.Add(new Line()
    {
        Margin = thickness,
        // Breite des Balkens
        StrokeThickness = strokeWidth,
        // Farbe des Balkens (Blau oder Rot)
        Stroke = brush,
        // 50% Transparent
        Opacity = 0.5,
        // Horizontale Startposition des Balkens ist abhängig von der Position in der Liste
        X1 = i * strokeWidth,
        // Vertikale Startposition: Grundlinie
        Y1 = plotCanvas.ActualHeight - 3,
        // Horizontale Endposition ist die gleiche wie die Startposition
        X2 = i * strokeWidth,
        // Vertikale Endposition der Linie: Höhe der Zeichnungsebene minus Balkenwert
        Y2 = plotCanvas.ActualHeight - bars[i] - 3
    });
}
```

Code 20: Zeichnen von Häufigkeitsverteilungen auf einer virtuellen Zeichnungsebene

#### 4.7.1 Erkenntnisse

Bei der stichprobenartigen Betrachtung einiger Balkendiagramme ist kein Statistik-Attribut aufgefallen, welches die beiden Kategorien besonders gut trennt. Es existiert immer eine gewisse Überschneidung der beiden Häufigkeitsverteilungen.

Bemerkenswert ist aber, dass Patches auf Rasenstücken die beiden Kategorien viel besser trennen als Patches in unmittelbarer Nähe der Strasse (Pannenstreifen, Mittelstreifen, Strassenmarkierungen).

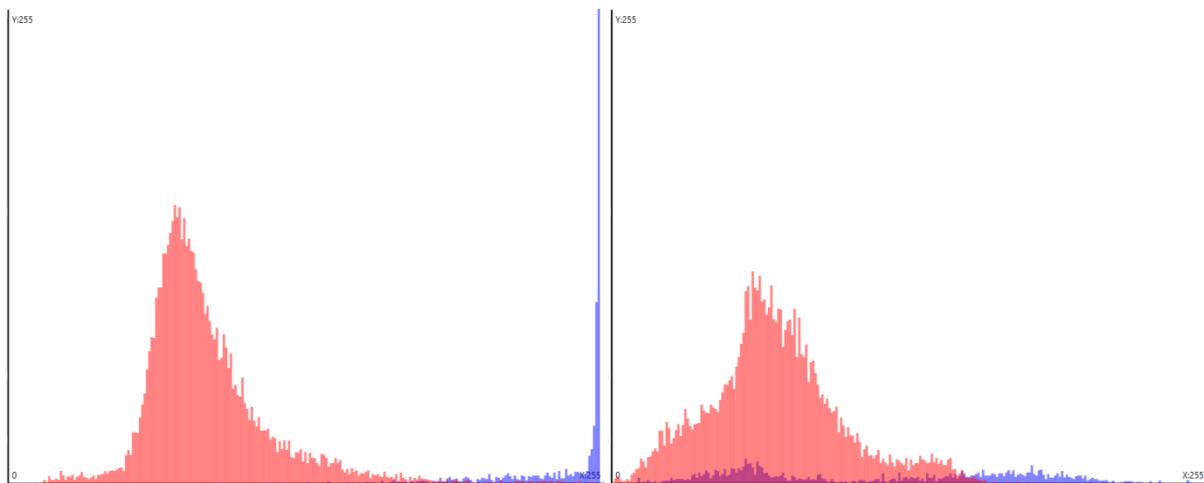


Abbildung 39: Häufigkeitsverteilungen des Medianwerts für die Kamera mvk101  
Links: Grass-Patch 110. Rechts Mittelstreifen-Patch 108

Der Grund dafür ist, dass der Schnee auf Rasenflächen länger liegenbleibt als auf Asphalt oder Flächen in Nähe der Strasse. So kann es passieren, dass auf einem Bild der Kategorie 'Schnee' ein Patch auf dem Pannenstreifen schneefrei ist und im gleichen Bild ein Patch mit Rasenfläche noch ganz 'weiss' ist.

#### 4.8 2D Plot Modul

Im eindimensionalen Plot hat sich kein Statistik-Attribut als klarer Gewinner bei der Trennung der Kategorien herausgestellt. Es gibt aber eine weitere Variante die Daten auf der Zeichnungsfläche zu betrachten: Die Werte zweier Statistik-Attribute können als X- und Y-Achse betrachtet werden. Die Schnittpunkte der beiden Werte auf der Zeichnungsebene können als Punkte markiert werden:

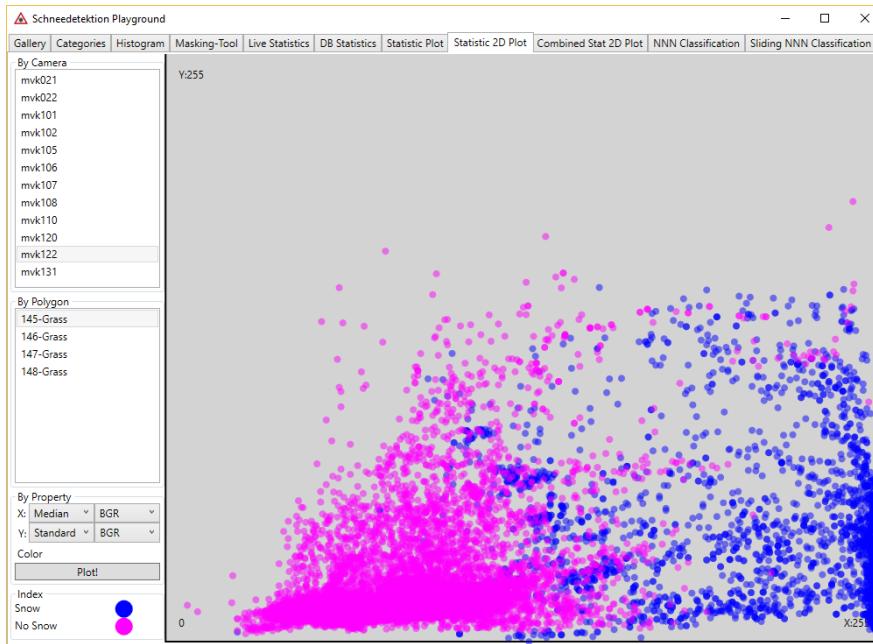


Abbildung 40: Bildschirmfoto des 2D Plot Moduls.

Plot der zweidimensionalen Darstellung zweier Kennzahlen der Kamera mvk112 (Grass-Patch 145)  
X-Achse: Medianwert (BGR). Y-Achse: Standardabweichung (BGR)

Dieses Modul baut auf dem vorangehenden Plot-Modul auf. Die Auswahl der Kameras und Patches funktioniert identisch. Der Unterschied bei den Statistik-Attributen gegenüber dem Plot Modul ist nun, dass deren zwei zur Auswahl stehen. Anstatt der akkumulierten Balken werden auf der Zeichnungsfläche blaue (Schnee) und Magenta (kein Schnee) Punkte gezeichnet. Jeder einzelne Punkt repräsentiert einen statistischen Wert.

```
// Für alle Statistik-Objekte
foreach (var statistic in statistics)
{
    // X-Position des Punktes aus dem Statistik-Objekt auslesen
    // Wert bei Bedarf skalieren auf den Wertebereich [0,255]
    double left = ScaleToCanvas(
        statistic.Get(selectedXAttribute, selectedXColor),
        selectedXAttribute, plotCanvas.ActualWidth);
    // Y-Position des Punktes aus dem Statistik-Objekt auslesen
    double top = (plotCanvas.ActualHeight) - ScaleToCanvas(
        statistic.Get(selectedYAttribute, selectedYColor),
        selectedYAttribute, plotCanvas.ActualHeight);

    // Eine Ellipse erstellen
    Ellipse e = new Ellipse()
    {
        // Die Ellipse soll so breit sein wie sie hoch ist
        Width = 8,
        Height = 8,
        // Farbe des Balkens (Blau oder Magenta)
        Fill = brush,
        // 50% Transparent
        Opacity = 0.5,
        // Position auf der Zeichnungsebene
        Margin = new Thickness(left, top, 0, 0),
        // ID des Statistik-Objekts in die Ellipse schreiben
        Tag = statistic.ID,
    };
    // Punkt zur Zeichnungsebene hinzufügen
    plotCanvas.Children.Add(e);
}
```

Code 21: Zeichnen von statistischen Werten als Punkte auf der Zeichnungsebene

#### 4.8.1 Erkenntnisse

Auch bei dieser Darstellung der Daten kristallisiert sich kein Statistik-Attribut-Paar heraus, dass die beiden Kategorien klar trennen. Die beiden Punkt-Wolken überschneiden sich bei jeder Kombination der Statistik-Attribute teilweise.

Auffallend bei dieser Betrachtungsperspektive ist, dass es auf der Ebene meistens einige Ausreisser hat. Da jeder gezeichnete Punkt einem Statistik-Objekt entspricht, kann zurückverfolgt werden warum dieser Datensatz ein solcher Ausreisser ist. Um das herauszufinden muss das 2D-Plot-Modul um eine Funktion erweitert werden: Bei einem Mausklick auf einen Punkt werden Bild und betrachteter Patch geladen und neben der Zeichnungsebene dargestellt:

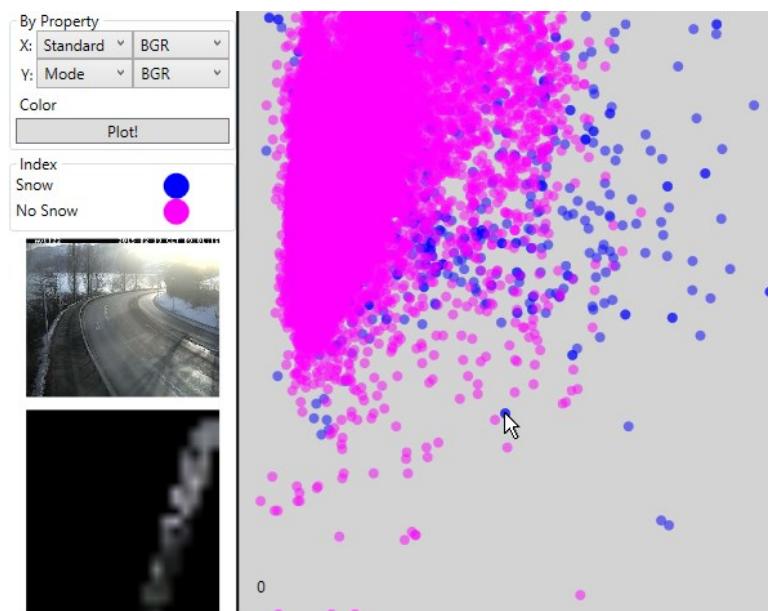


Abbildung 41: Ausgewählter Patch im 2d Plot Modul

Ursachen für solche Ausreisser sind in praktisch allen Fällen:

- Starker Nebel
- Schlechte Lichtverhältnisse: Die Sonne scheint in die Kamera, verändert die Farbbebalance im Bild und lässt den Patch dunkler erscheinen.
- Ein Auto ist im Patch abgebildet

#### 4.9 Statistische Werte zeitlich kombinieren

Da in den erfassten statistischen Werten kein klarer Kandidat herausgestochen ist, welcher die beiden Kategorien für die meisten Patches an einem Schwellwert gut trennt, musste ein anderer Ansatz verfolgt werden. Der gesammelte Datenstand soll als Vergleichsbasis für neue Bilder dienen. Hierzu muss dieser reduziert, bereinigt und angereichert werden:

- Der Datensatz muss reduziert werden, damit ein Klassifikator gegen eine kleinere Menge von Vergleichswerten prüfen kann. So kann die Verarbeitungszeit einer Klassifizierung in weniger Rechenzeit durchgeführt werden.
- Ausreisser in den Datensätzen müssen eliminiert werden. Es gibt Bilder, auf denen Fahrzeuge einen Bildausschnitt verbergen und damit die Farben stark verändern.
- Da die Aufnahmezeit eines Bildes bekannt ist, soll ermöglicht werden, diesen Datensatz mit Werten zu vergleichen die in einem ähnlichen Zeitraum entstanden sind.

#### 4.9.1 Statistische Werte in 2-Stunden-Zeitfenster kombinieren und abspeichern

Die Aufzeichnung der Bilder verlief über mehrere Monate im Winter. Während dieser Zeit verändert sich der Sonnenstand bemerkbar. Die Bildqualität kann von der Sonne stark beeinflusst werden, wenn sie tief steht und direkt in die Kamera scheint. Dieser Einfluss ist

unterschiedlich für Bilder die beispielsweise im Oktober und im Januar aufgenommen wurden. Weiterhin treten diese Einflüsse im Tagesverlauf immer zu einer ähnlichen Zeit auf. Während der Entwicklung und im Gespräch mit den Betreuern kam aus diesen Gründen immer wieder die Frage auf, wie ein ‘typischer’ statistischer Wert für ein Bild aussieht, dass in einer bestimmten Jahreswoche zu einer bestimmten Zeit aufgenommen wurde. Oder konkreter: «Wie sieht ein Bild aus in der ersten Jahreswoche um neun Uhr morgens, wenn Schnee liegt?»

#### 4.9.2 Erfasste Jahreswochen

Es sollen nun Referenzwerte entstehen, die wöchentlich gruppiert und innerhalb einer Woche weiter in Zeitfenster gruppiert sind. Im Bildarchiv sind Daten für folgende Jahreswochen erfasst:

Winter 2014 / 2015			Winter 2015 / 2016		
Datum Montag	Jahreswoche	Jahr	Datum Montag	Jahreswoche	Jahr
01.12.2014	49	2014	21.09.2015	39	2015
08.12.2014	50		28.09.2015	40	
15.12.2014	51		05.10.2015	41	
22.12.2014	52		12.10.2015	42	
29.12.2014	1		19.10.2015	43	
05.01.2015	2		26.10.2015	44	
12.01.2015	3		02.11.2015	45	
19.01.2015	4		09.11.2015	46	
26.01.2015	5		16.11.2015	47	
02.02.2015	6		23.11.2015	48	
09.02.2015	7		30.11.2015	49	
16.02.2015	8		07.12.2015	50	
23.02.2015	9		14.12.2015	51	
02.03.2015	10		21.12.2015	52	
09.03.2015	11	2015	28.12.2015	1	2016
16.03.2015	12		04.01.2016	2	
23.03.2015	13		11.01.2016	3	
30.03.2015	14		18.01.2016	4	
06.04.2015	15		25.01.2016	5	
13.04.2015	16		01.02.2016	6	
20.04.2015	17		08.02.2016	7	
27.04.2015	18		22.02.2016	8	
04.05.2015	19		29.02.2016	9	
11.05.2015	20		07.03.2016	10	
18.05.2015	21		14.03.2016	11	
25.05.2015	22		21.03.2016	12	
01.06.2015	23		28.03.2016	13	
08.06.2015	24		04.04.2016	14	
			11.04.2016	15	

Tabelle 3: Jahreswochen mit Bildmaterial für Winter 2014 / 2015 und Winter 2015 / 2016

#### 4.9.3 Zeitfenster

Pro Kamera und Jahreswoche werden die Bilder nun noch einmal in Zeitfenster unterteilt. Diese Zeitfenster umspannen jeweils zwei Stunden und beginnen um sechs Uhr morgens. Es ergeben sich sieben Zeitfenster, in welchen an einem Tag genügend beleuchtete Bilder entstehen:

- 6 bis 8 Uhr
- 8 bis 10 Uhr
- 10 bis 12 Uhr
- 12 bis 14 Uhr
- 14 bis 16 Uhr
- 16 bis 18 Uhr
- 18 bis 20 Uhr

#### 4.9.4 Kategorie-Permutationen

Innerhalb einer Woche ist es möglich, dass beispielsweise montags um acht Uhr noch Schnee neben der Strasse liegt, während am Freitag um die gleiche Zeit dieser Schnee bereits weggeschmolzen ist. Die Bilder dieser Kategorien gehören nicht in die selbe Gruppe und sollen getrennt voneinander betrachtet werden. Des Weiteren können auch Bilder mit schlechten Lichtverhältnissen, mit Nebel und mit Niederschlag getrennt werden, da diese Informationen beim Aufbau der Stichprobe alle erfasst wurden. Auch Permutationen dieser vier Kategorien kommen vor und können getrennt betrachtet werden:

Alle möglichen Kategorie-Permutationen			
Schnee	schlechte Lichtverhältnisse	kein Nebel	kein Niederschlag
Schnee	schlechte Lichtverhältnisse	Nebel	kein Niederschlag
Schnee	schlechte Lichtverhältnisse	kein Nebel	Niederschlag
Schnee	gute Lichtverhältnisse	kein Nebel	kein Niederschlag
Schnee	gute Lichtverhältnisse	Nebel	kein Niederschlag
Schnee	gute Lichtverhältnisse	kein Nebel	Niederschlag
kein Schnee	schlechte Lichtverhältnisse	kein Nebel	kein Niederschlag
kein Schnee	schlechte Lichtverhältnisse	Nebel	kein Niederschlag
kein Schnee	schlechte Lichtverhältnisse	kein Nebel	Niederschlag
kein Schnee	gute Lichtverhältnisse	kein Nebel	kein Niederschlag
kein Schnee	gute Lichtverhältnisse	Nebel	kein Niederschlag
kein Schnee	gute Lichtverhältnisse	kein Nebel	Niederschlag

Tabelle 4: Alle möglichen Kategorie-Permutationen

Während dem Aufbau der Stichprobe wurde keinem Bild die Kategorien 'Nebel' und 'Niederschlag' zugleich zugewiesen.

#### 4.9.5 Skript für die Kombination

Aus all diesen Vorbedingungen entstand ein Algorithmus, der die Berechnung der zeitlich kombinierten statistischen Werte Schritt für Schritt durchführt. Da der implementierte Code komplex ist und durch Vereinfachung an Bedeutung verliert steht hier eine Abbildung des Algorithmus in Pseudocode:

```
FOR EACH Camera
    FOR EACH Week
        FOR EACH Time slot
            FOR EACH Category permutation
                Load pictures
                IF Group consists of less than 3 pictures
                    Discard Group
                ELSE
                    Create average picture from group
                    FOR EACH Patch in combined picture
                        Calculate statistic values for Patch
                        Save statistic values to database
                    ENDFOR
                ENDIF
            ENDFOR
        ENDFOR
    ENDFOR
ENDFOR
```

Code 22: Statistische Werte zeitlich kombinieren in Pseudocode

Die gefundenen Bilder in der Kategorie-Permutation werden zu einem Bild kombiniert. Aus der Bildgruppe wird ein zuerst Durchschnittsbild hergestellt, dass in die Patches unterteilt wird. Dieses Durchschnittsbild entsteht, wenn für jeden Bildpunkt aus den Quellbildern der mittlere Farbwert berechnet und dem neuen Bild zugewiesen wird.

Wie bei dem ursprünglichen Abspeichern aller statistischen Werte, mussten die kombinierten Statistiken stapelweise in die Datenbank abgelegt werden. In diesem Fall wurden die Daten nach jeder Berechnung einer Jahreswoche in die Datenbank gespeichert.

#### 4.9.6 Combined Plot Modul

Um die kombinierten, statistischen Werte zu visualisieren, wurde die Applikation um ein Modul erweitert, dass diese auf einer zweidimensionalen Ebene darstellen kann. Funktional unterscheidet es sich nur geringfügig zum 2D Plot Modul. Die Menge der Datenpunkte ist aber viel kleiner. Aus diesem Grund bietet sich die Gelegenheit die Punkte grösser zu zeichnen und mit zusätzlichen Informationen anzureichern: Zusätzlich zur Farbe der Hauptkategorien 'Schnee' und 'kein Schnee' sind die statistischen Werte mit farbigen Rändern versehen:

- Türkis bei schlechten Lichtverhältnissen
- Grün bei Niederschlag
- Gelb bei Nebel

Bei der Gegenüberstellung eines 2D Plots und eines Combined Plots wird das Ausmass der Reduktion der Datenpunkte ersichtlich. Zusätzlich zeigt die Darstellung der kombinierten statistischen Werte, dass sich die Kategorien (schlechte Lichtverhältnisse / Niederschlag / Nebel) tendenziell in Gruppen sammeln.

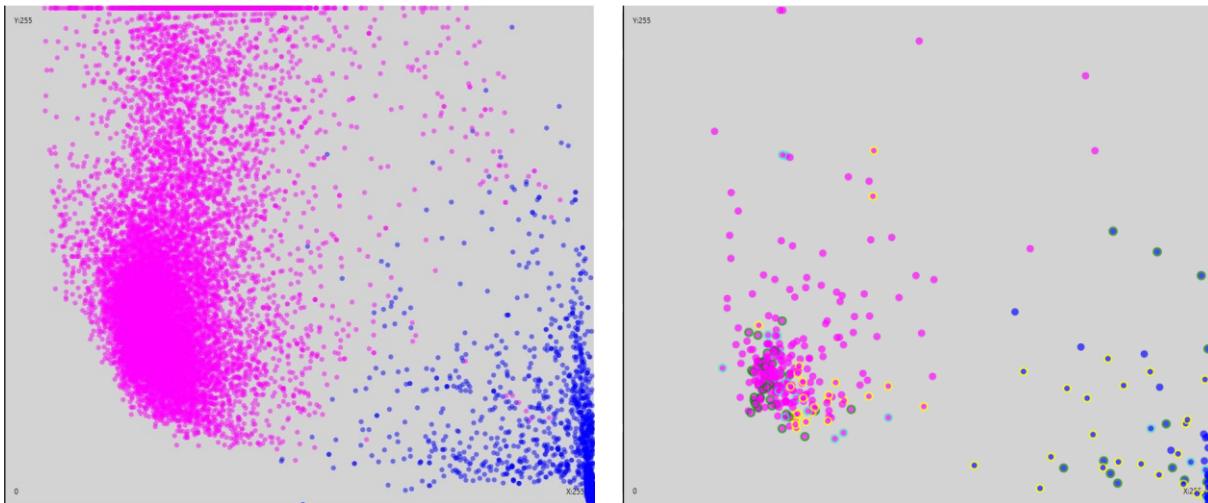


Abbildung 42: Plot der zweidimensionalen Darstellung zweier Kennzahlen der Kamera mvk101 (Grass-Patch 110)

x-Achse: Mittelwert (blau). Y-Achse: Kontrast (blau)

Linke Hälfte: Alle erfassten Datensätze für mvk101. Rechte Hälfte: Reduzierte Datensätze für mvk101

## 4.10 Klassifikation

Die kombinierten statistischen Werte können nun als Wissensbasis für eine Klassifikation dienen. Neue, noch nicht klassifizierte Bilder sollen mit den bestehenden Werten verglichen und auf Grund einer Regel, der einen oder anderen Klasse zugewiesen werden.

Weil sich die Klassengrenzen in der Datenbasis nicht linear trennen lassen, bietet sich der k-Nearest Neighbour Klassifikator (kNN) an (Hudritsch, 2016). Dieser ist einfach implementierbar und, da die Datenbasis im vorherigen Schritt reduziert wurde, es kann davon ausgegangen werden, dass eine Klassifizierung überschaubar viel Rechenzeit in Anspruch nimmt. Vor der Implementierung dieses Klassifikators muss in der Menge der gespeicherten statistischen Werte ein Distanzmaß definiert werden.

### 4.10.1 Distanzmaß

Ein gespeicherter statistischer Wert besteht aus 24 Kennzahlen: Modalwert, Mittelwert, Median, Minimum, Maximum, Standardabweichung, Varianz und Kontrast – jeweils für jeden der drei Farbkanäle. Diese Werte sind als Fließkommazahlen in einer Zeile der Datenbank abgelegt. Der erstbeste Ansatz ist nun eine Zeile als Vektor in einem 24-Dimensionalen Raum zu betrachten. Da Standardabweichung und Varianz voneinander abhängig sind, kann eine der Kennzahlen ohne grossen Informationsverlust ausgelassen werden. So verbleiben 21 Dimensionen.

In einem hochdimensionalen Raum bietet sich die Euklidische Distanz als Maß an. Zunächst werden die Differenzen komponentenweise gebildet, anschliessend werden die Resultate quadriert und aufsummiert. Die Wurzel der Summe ist die resultierende Distanz  $d(x, y)$ :

$$d(x, y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \dots + (x_n - y_n)^2}$$

Im Code wurde die Distanz so umgesetzt, dass jeder statistische Wert den ‘Abstand’ zu einem anderen statistischen Wert berechnen kann:

```

public double DistanceTo(Statistic other)
{
    return Math.Sqrt(
        Math.Pow(this.ModeBlue      - other.ModeBlue, 2) +
        Math.Pow(this.ModeGreen     - other.ModeGreen, 2) +
        Math.Pow(this.ModeRed       - other.ModeRed, 2) +
        Math.Pow(this.MeanBlue      - other.MeanBlue, 2) +
        Math.Pow(this.MeanGreen     - other.MeanGreen, 2) +
        Math.Pow(this.MeanRed       - other.MeanRed, 2) +
        Math.Pow(this.MedianBlue    - other.MedianBlue, 2) +
        Math.Pow(this.MedianGreen   - other.MedianGreen, 2) +
        Math.Pow(this.MedianRed     - other.MedianRed, 2) +
        Math.Pow(this.MinimumBlue   - other.MinimumBlue, 2) +
        Math.Pow(this.MinimumGreen  - other.MinimumGreen, 2) +
        Math.Pow(this.MinimumRed    - other.MinimumRed, 2) +
        Math.Pow(this.MaximumBlue   - other.MaximumBlue, 2) +
        Math.Pow(this.MaximumGreen  - other.MaximumGreen, 2) +
        Math.Pow(this.MaximumRed    - other.MaximumRed, 2) +
        Math.Pow((this.StandardDeviationBlue - other.StandardDeviationBlue) * 2, 2) +
        Math.Pow((this.StandardDeviationGreen - other.StandardDeviationGreen) * 2, 2) +
        Math.Pow((this.StandardDeviationRed - other.StandardDeviationRed) * 2, 2) +
        Math.Pow((this.ContrastBlue - other.ContrastBlue) * 255, 2) +
        Math.Pow((this.ContrastGreen - other.ContrastGreen) * 255, 2) +
        Math.Pow((this.ContrastRed - other.ContrastRed) * 255, 2));
}

```

Code 23: Berechnung der Euklidischen Distanz zwischen zwei statistischen Werten

Der Wertebereich der meisten Werte ist zwischen 0 und 255, da sie direkt aus Farbwerten in Bildern abgeleitet werden. Die Standardabweichung nimmt Werte zwischen 0 und etwa 120 an. Darum wird eine naive Skalierung der Dimension ausgeführt, indem die Komponente verdoppelt wird. Der Kontrast, welcher Werte zwischen 0 und 1 annimmt, wird ebenfalls auf den Wertebereich der Farbwerte skaliert.

#### 4.10.2 Algorithmus

Der Algorithmus ist sehr einfach aufgebaut. Als erstes werden alle kombinierten statistischen Werte geladen, die zur aktuell betrachteten Kameraperspektive gehören. Dann werden die Patches aus dem aktuell betrachteten Bild auseinander geschnitten und die statistischen Werte für die Patches berechnet.

Dann folgt für jeden Patch die Suche nach den nächsten Nachbaren: Die kombinierten statistischen Werte, die den gleichen Patch beschreiben, werden geladen. Die Distanz zu jedem geladenen Element wird berechnet und in einer Kollektion gespeichert. Anschliessend kann die Kollektion sortiert und auf die nächsten Nachbaren reduziert werden. Im letzten Schritt wird untersucht werden zu welcher Kategorie die gefundenen nächsten Nachbaren gehören: Es werden alle kombinierten statistischen Werte nach Kategorie aufsummiert. Nachdem dieser Prozess für jeden Patch ausgeführt wurde, können die Summen verglichen werden. Es gewinnt diejenige Kategorie, deren statistischen Werte sich häufiger unter den nächsten Nachbaren befunden haben.

```

private void FindNearestNeighbours(
    ClassificationViewModel classificationViewModel, int numberofNearestNeighbours)
{
    [...]

    // Für jedes Patch der Kamera des aktuell betrachteten Bildes
    foreach (var polygon in polygons)
    {
        // Alle Kombinierte Statistiken des des Polygons laden
        var combinedStatisticsForPolygon =
            this.combinedStatistics.Where(cs => cs.Polygon_ID == polygon.ID);

        // Leere Dictionary erstellen, die einen kombinierten statistischen Wert mit
        // der Distanz zu den statistischen Werten des aktuell betrachteten Patches
        // verbindet
        Dictionary<Combined_Statistic, double> distances =

```

```

        new Dictionary<Combined_Statistic, double>();

    // Für jede kombinierte Statistik
    foreach (var combinedStatistic in combinedStatisticsForPolygon)
    {
        // Distanz zu den statistischen Werten des aktuell betrachteten Patches
        // berechnen und in Kollektion ablegen
        distances.Add(combinedStatistic,
            sourceStatistic.DistanceTo(combinedStatistic.Statistic));
    }

    // Kollektion nach Distanz sortieren
    // Kollektion auf die ersten k Elemente reduzieren (take)
    var nearestNeighbours = distances
        .OrderBy(d => d.Value)
        .Take(numberOfNearestNeighbours);

    // Anzahl Schnee-Bilder unter nächsten Nachbaren bestimmen
    snow += nearestNeighbours.Where(nn => nn.Key.Snow.Value).Count();
    // Anzahl Nicht-Schnee-Bilder unter nächsten Nachbaren Bestimmen
    noSnow += nearestNeighbours.Where(nn => !nn.Key.Snow.Value).Count();
}

[...]
}

```

Code 24: kNN-Suche für statistische Werte

#### 4.10.3 Benutzerschnittstelle

Um die Resultate der Klassifikation zu visualisieren und die Parameter des Klassifikators zu verändern, wurde die Applikation um ein weiteres Modul erweitert.

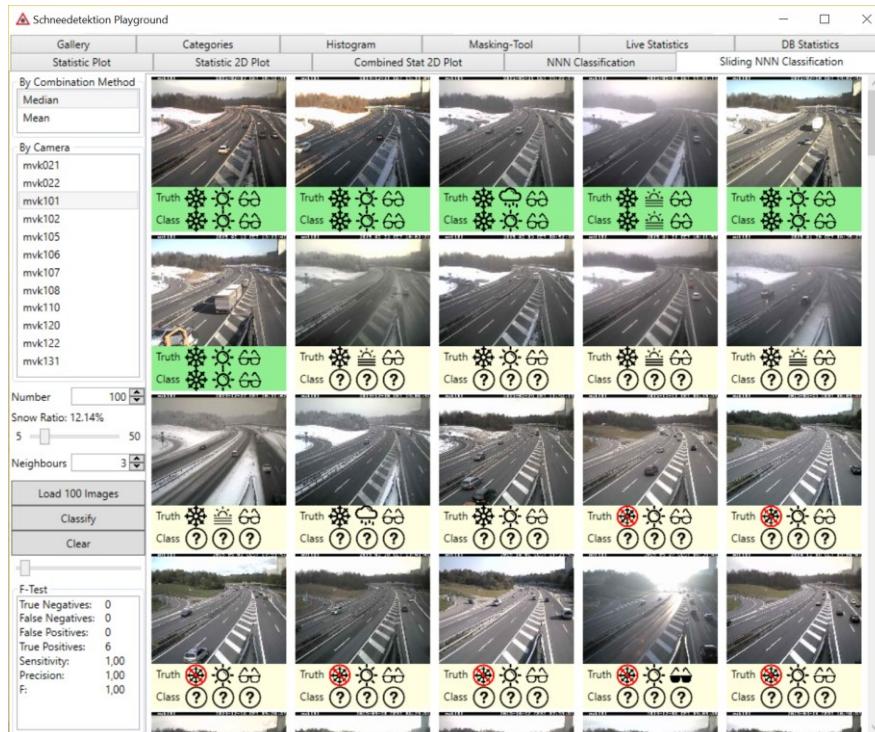


Abbildung 43: Bildschirmfoto des kNN Moduls.

In dieser Benutzerschnittstelle lässt sich einstellen welche Kameraperspektive getestet wird und wie viele Testbilder geladen werden. Bei der Menge der geladenen Bilder lässt sich konfigurieren in welchem Verhältnis die beiden Kategorien 'Schnee' und 'kein Schnee' stehen. Dieses Verhältnis lässt sich frei verschieben. Maximal können die Kategorien im

Verhältnis 0,5 stehen. Standardeinstellung ist das echte, beobachtete Verhältnis zwischen Schneebildern und schneefreien Bildern, dass nach Erfassen der Stichprobe ersichtlich wurde:

```
// Gewählte Kamera aus der Liste auslesen
string selectedCamera = cameraList.SelectedItem as String;

// Anzahl aller Bilder dieser Kamera am Tag auslesen
double total = dataContext.Images
    .Where(i => i.Day.Value && i.Place == selectedCamera).Count();
// Anzahl aller Schneebilder dieser Kamera am Tag auslesen
double snow = dataContext.Images
    .Where(i => i.Day.Value && i.Place == selectedCamera && i.Snow.Value).Count();

// Verhältnis zwischen Schneebildern und schneefreien Bildern
// ausrechnen und dem Slider zuweisen
ratio.Value = 100d / total * snow;
```

Code 25: Berechnung des Verhältnisses zwischen Schneebildern und schneefreien Bildern

Aus dem gesamten Bildarchiv einer Kameraperspektive wird für die Klassifizierung immer eine zufällige Auswahl von Bildern ausgewählt. Hierbei wird auf der Kollektion von Bildern ein Fisher-Yates-Shuffle<sup>9</sup> durchgeführt um die Reihenfolge zu randomisieren.

#### 4.10.4 Resultate

Während der Klassifizierung zählt die Applikation die korrekt und falsch zugewiesenen Bilder. Unten links in der Benutzerschnittstelle wird das Resultat des F-Tests dargestellt. Zudem kann bei jedem Bild abgelesen werden, ob die Klassifizierung erfolgreich war:



Wenn die korrekte Klasse zugeordnet wird, ist die Informationsleiste grün eingefärbt. In der ersten Zeile stehen die echten Kategorien für dieses Bild (Schnee / Niederschlag / gute Sichtverhältnisse). In der Zeile Darunter werden die zugewiesenen Kategorien aufgeführt.

Abbildung 44: Korrekt klassifiziertes Bild



Wenn die Klassifikation bezüglich ‘Schnee’ und ‘kein Schnee’ nicht erfolgreich war, ist die Informationsleiste rot eingefärbt. In diesem Beispiel liegt neben der Strasse Schnee. Die zugewiesene Kategorie ist aber ‘kein Schnee’. Zudem herrschen auf der Aufnahme schlechte Lichtverhältnisse (Sonnenbrillen-Symbol). Diese Kategorie wurde ebenfalls falsch zugeordnet. Die falsche Zuordnung bezüglich Lichtverhältnissen trägt nicht zum Resultat des F-Test bei. Die Anzeige der anderen Kategorien kann aber hilfreich sein bei der Untersuchung der Schwachstellen der Klassifizierung.

Abbildung 45: Falsch klassifiziertes Bild

<sup>9</sup> Hintergrund zum Fisher-Yates-Shuffle: [https://en.wikipedia.org/wiki/Fisher%20Yates\\_shuffle](https://en.wikipedia.org/wiki/Fisher%20Yates_shuffle)

In einem ersten Durchlauf wurde für die Kameras mvk106 und mvk108 eine Klassifizierung von 1000 Bildern durchgeführt. Das Verhältnis zwischen Schneebildern und schneefreien Bildern war 0,5. Es wurden also jeweils 500 Bilder der beiden Kategorien betrachtet. Beim kNN-Klassifikator wurden die 7 nächsten Nachbaren berücksichtigt.

	<b>mvk106</b>	<b>mvk108</b>
True Negatives	500	492
False Negatives	38	35
False Positives	0	8
True Positives	462	465
Sensitivity	0,92	0,93
Precision	1,00	0,98
F	0,96	0,96

Tabelle 5: F-Test für die kNN-Klassifikation

Der Klassifikator weist bereits in der ersten Ausprägung bemerkenswert viele Bilder der richtigen Kategorie zu. Von 1000 Bildern wurden deren 962 beziehungsweise 965 korrekt eingeschätzt. Der Klassifikator schwächelt noch bei der Erkennung der Schneebilder. Die Zahl der ‘False Negatives’ – Schneebilder denen fälschlicherweise die Kategorie ‘kein Schnee’ zugewiesen wurde – ist deutlich höher als die Zahl der ‘False Positives’. Bei diesen schneefreien Bildern ist die falsche Kategorie zugewiesen worden. Es wäre daher wünschenswert, wenn die ‘Sensitivität’ des Klassifikators höher wäre.

#### 4.10.5 Optimierung: Median-Kombination

Bei der ersten Reduktion der statistischen Werte wurden Bilder kombiniert um für ein Zeitfenster typische statistische Werte zu erhalten. Bei der Verschmelzung der Bilder wurden die Bildpunkte aus den Quellbildern über den Mittelwert berechnet. Eine andere Variante wäre diese Kombination über den Medianwert zu machen. Aus diesem Grund wurde für alle Kameras der Prozess aus Kapitel 4.9 erneut durchgeführt. Bei diesem zweiten Durchlauf wurde über den Median kombiniert.

Anschliessend konnte die Klassifikation mit derselben Konfiguration erneut getestet werden:

- Insgesamt 1000 Bilder. 500 Schneebilder und 500 schneefreie Bilder
- Berücksichtigung der 7 nächsten Nachbaren

	<b>mvk106</b>	<b>mvk108</b>
True Negatives	500	495
False Negatives	41	33
False Positives	0	5
True Positives	459	467
Sensitivity	0,92	0,93
Precision	1,00	0,99
F	0,96	0,96

Tabelle 6: F-Test für die kNN-Klassifikation mit Median-Kombination

Diese Optimierung brachte nicht den erwünschten Effekt. Die Sensitivität wurde nicht erhöht.

#### 4.10.6 Optimierung: Inputbild Kombination

Bei der genaueren Betrachtung der falsch klassifizierten Bilder fällt auf, dass häufig die Bilder betroffen sind, auf denen Fahrzeuge die Sicht auf einen Patch behindern. Da aber Bilder in regelmässigen Abständen von den Webcams heruntergeladen werden, ist es möglich das Inputbild aus mehreren Bildern herzustellen. Die Idee ist in einem zukünftigen Livebetrieb Bilder in kurzer Reihenfolge von der Webcam herunterzuladen und zu kombinierten. Mit diesem Vorgehen könnten Autos aus den Bildern bereinigt werden.

Der Klassifikator wurde darum um einen weiteren Parameter erweitert: Anzahl Quellen. Dieser Parameter steuert wie viele Inputbilder für die Klassifizierung herangezogen werden. Es werden zeitlich unmittelbar angrenzende (ältere und jüngere) Bilder aus dem Archiv geladen und mit dem bestehenden Klassifikator bearbeitet. Das Heraussuchen und Kombinieren der Bilder sieht so aus:

```
// Falls mehr als ein Bild für die Klassifizierung berücksichtigt werden soll
if (numberOfSources > 1)
{
    // Leere Liste für die Dateinamen der Bilder erstellen
    List<string> sourcesFileNames = new List<string>();
    // Das aktuell betrachtete Bild zur Liste hinzufügen
    sourcesFileNames.Add(classificationViewModel.Image.FileName);

    // Bestimmen wie viele ältere und jüngere Bilder geladen werden sollen
    int numberOfSourcesToFind = (numberOfSources - 1) / 2;

    // Nächst ältere Bilder suchen
    var olderImages = dataContext.Images
        // Alle Bilder deren Aufnahmzeitpunkt kleiner ist als des aktuellen
        .Where(i => i.UnixTime < classificationViewModel.Image.UnixTime)
        // Absteigend nach Aufnahmzeitpunkt sortieren
        .OrderByDescending(i => i.UnixTime)
        // Vorbestimmte Menge laden
        .Take(numberOfSourcesToFind);

    var newerImages = dataContext.Images
        // Alle Bilder deren Aufnahmzeitpunkt grösser ist als des aktuellen
        .Where(i => i.UnixTime > classificationViewModel.Image.UnixTime)
        // Aufsteigend nach Aufnahmzeitpunkt sortieren
        .OrderBy(i => i.UnixTime)
        // Vorbestimmte Menge laden
        .Take(numberOfSourcesToFind);

    // Ältere und jüngere in eine Kollektion aufnehmen
    var additionalSources = olderImages.Concat(newerImages);

    // Alle gefundenen Bilder in Liste aufnehmen
    foreach (var image in additionalSources)
    {
        sourcesFileNames.Add(image.FileName);
    }

    // Alle Bilder mit der Median-Methode kombinieren und in Bitmap umwandeln
    bitmap = OpenCVHelper.BitmapToBitmapImage(
        openCVHelper.CombineImagesMedian(sourcesFileNames).Bitmap);
}
```

Code 26: Inputbilder kombinieren für die kNN-Suche

Mit dieser Erweiterung wurde der Klassifikator erneut mit folgenden Parametern getestet:

- Insgesamt 1000 Bilder. 500 Schneebilder und 500 schneefreie Bilder
- Statistische Werte aus der Mittelwert-Kombination
- Berücksichtigung der 7 nächsten Nachbarn
- 3 Inputbilder: Berücksichtigung eines älteren und eines jüngeren Bildes

	<b>mvk106</b>	<b>mvk108</b>
True Negatives	493	492
False Negatives	43	26
False Positives	7	8
True Positives	457	474
Sensitivity	0,97	0,95
Precision	0,98	0,98
F	0,95	0,97

Tabelle 7: F-Test für die kNN-Klassifikation mit Input-Kombination

#### 4.10.7 Optimierung: Patches aus 3 Inputs auswählen

Eine weitere Variante die Inputwerte von Störfaktoren zu befreien ergibt sich, wenn Patches aus älteren und jüngeren Bildern miteinander verglichen werden. Die Idee ist, dass Bei einem neuen Inputbild zunächst die Patches ausgeschnitten und anschliessend mit Patches aus zeitlich benachbarten Bildern (ein älteres und ein jüngeres) verglichen werden. Haben die statistischen Werte eines Patches eine zu grosse Distanz zu den anderen zwei in der Gruppe, so kann der Ausreisser verworfen werden. Auf diese Weise werden automatisch Patches, auf denen Autos abgebildet sind, nicht in der Klassifikation berücksichtigt. Weil die statistischen Werte dieses Patches stark differenzieren von Patches auf denen kein Auto abgebildet ist.

Diese Optimierung sieht im Code so aus:

```
// Neue Kollektion für statistische Werte initialisieren
var imageStatistics = new Dictionary<Polygon, List<Statistic>>();
// Für jedes Patch des Bildes
foreach (var polygon in polygons)
{
    IEnumerable<Point> polygonPoints =
        PolygonHelper.DeserializePointCollection(polygon.PolygonPointCollection);
    // Statistische Werte für das aktuelle Bild Berechnen
    Statistic statistic = openCVHelper.GetStatisticForPatchFromImagePath(
        classificationViewModel.Image.FileName, polygonPoints);

    if (olderImage != null && newerImage != null)
    {
        // Statistische Werte für das ältere Bild berechnen
        Statistic olderStatistic = openCVHelper.GetStatisticForPatchFromImagePath(
            olderImage.FileName, polygonPoints);
        // Statistische Werte für das jüngere Bild berechnen
        Statistic newerStatistic = openCVHelper.GetStatisticForPatchFromImagePath(
            newerImage.FileName, polygonPoints);

        // Distanz zwischen aktuellem und älterem Patch berechnen
        double distanceToOlder = statistic.DistanceTo(olderStatistic);
        // Distanz zwischen aktuellem und jüngerem Patch berechnen
        double distanceToNewer = statistic.DistanceTo(newerStatistic);
        // Distanz zwischen älterem und jüngerem Patch berechnen
        double distanceBetweenSurrounding = olderStatistic.DistanceTo(newerStatistic);

        // Falls die Distanzen zwischen den drei beobachteten Patches klein ist,
        // werden alle berücksichtigt für die Klassifikation
        if (distanceToOlder < 50
            && distanceToNewer < 50
            && distanceBetweenSurrounding < 50)
        {
            imageStatistics.Add(polygon, (
                new Statistic[] { statistic, olderStatistic, newerStatistic }).ToList());
        }
        // Das jüngere Bild ist der Ausreisser und wird verworfen
        else if (distanceToOlder < distanceToNewer
            && distanceToOlder < distanceBetweenSurrounding)
        {
            imageStatistics.Add(polygon, (
                new Statistic[] { statistic, olderStatistic }).ToList());
        }
        // Das ältere Bild ist der Ausreisser und wird verworfen
        else if (distanceToNewer < distanceToOlder
            && distanceToNewer < distanceBetweenSurrounding)
        {
            imageStatistics.Add(polygon, (
                new Statistic[] { statistic, newerStatistic }).ToList());
        }
        // Das aktuelle Bild ist der Ausreisser und wird verworfen
        else if (distanceBetweenSurrounding < distanceToNewer
            && distanceBetweenSurrounding < distanceToOlder)
        {
    }
}
```

```

        imageStatistics.Add(polygon, (
            new Statistic[] { olderStatistic, newerStatistic }).ToList());
    }
}

```

Code 27: Ausreisser-Patches aus der Inputmenge vorzeitig eliminieren

Auch mit dieser Erweiterung wurde der Klassifikator getestet. Die Parameter sind die gleichen wie bei der Kombination der Inputbilder:

- Insgesamt 1000 Bilder. 500 Schneebilder und 500 schneefreie Bilder
- Statistische Werte aus der Mittelwert-Kombination
- Berücksichtigung der 7 nächsten Nachbarn
- 3 Inputbilder: Berücksichtigung eines älteren und eines jüngeren Bildes

	<b>mvk106</b>	<b>mvk108</b>
True Negatives	429	490
False Negatives	54	35
False Positives	2	10
True Positives	446	465
Sensitivity	0,85	0,93
Precision	1,00	0,98
F	0,94	0,95

Tabelle 8: F-Test für die kNN-Klassifikation mit Input-Anreicherung aus mehreren Patches

#### 4.10.8 Ergebnisse

Es stellt sich heraus, dass es sehr schwierig ist einen bereits guten Klassifikator noch signifikant zu verbessern: Die drei verfolgten Optimierungsstrategien konnten den F-Wert nicht mehr steigern. Die Sensitivität der Detektion konnte mit der Inputbild-Kombination deutlich gesteigert werden. Es scheint aber, dass diese Verbesserung auf Kosten der Präzision erzielt wurde. Der resultierende F-Wert veränderte sich nur wenig.

### 4.11 Weitere Themen

Wie bei der Projektarbeit wurden auch während der Bachelorthesis Themen und Ansätze verfolgt, die keine Resultate geliefert haben. Hierbei lag es nicht an der Qualität der Ideen. Der Hauptgrund für den Verzicht die Themen weiter zu verfolgen war, dass die Zeit im Semester knapp war und der Fokus eher auf konkrete Ergebnisse gelegt wurde.

#### 4.11.1 Sonneneinstrahlungs-Patch

Zusätzlich zu den Patches neben der Strasse könnte ein weiterer Patch definiert werden. Dieser deckt die Region im Bild ab, welche bei schlechten Lichtverhältnissen von der Sonne beleuchtet wird. Die statistischen Werte dieses Patches sollen mit den bestehenden gespeichert werden. Die Idee dabei ist, dass Bilder mit schlechten Lichtverhältnissen besser detektiert werden. Bei der Detektion eines solchen Bildes könnten entweder Spezialregeln die Detektion verfeinern oder die Bilder könnten grundsätzlich verworfen werden. Bei der zweiten Variante würde der Algorithmus so lange Bilder von der Webcam herunterladen und wieder verwerfen, bis der Sonnenstand sich genügend verändert hat und die schlechten Lichtverhältnisse nicht mehr gegeben sind.

#### 4.11.2 Logit Regression<sup>10</sup>

Bei dem gegebenen Wissensstand und dem Ziel neue Inputs einem der beiden Klassen zuzuordnen, wäre eine Klassifikation per Logit-Regression besonders geeignet. Die Logit-Regression (oder logistische Regression) ist eine Art es überwachten Lernens. Die Vorteile

<sup>10</sup> Logistische Regression: [https://de.wikipedia.org/wiki/Logistische\\_Regression](https://de.wikipedia.org/wiki/Logistische_Regression)

der Methode sind: Beim Training der Logit-Regression wird die Effektstärke und die Streuung eines jeden Parameters zurückgegeben. Die Parameter wären im Fall der Schneedetektion die einzelnen Kennzahlen aus den statistischen Werten. Es könnte so in Erfahrung gebracht werden welche Kennzahlen einen grossen Einfluss auf die Klassifizierung haben und solche ohne Einfluss könnten aussortiert werden.

Ein weiterer Vorteil der Logit-Regression ist, dass die Klassifizierung unscharf ist: Bei Inputs, bei denen die Klassenzugehörigkeit nicht klar ist, wird ein Wert zwischen 0 und 1 zurückgegeben. Auf diese Information könnte dann entsprechend reagiert werden. Der kNN-Klassifikator weist immer genau eine Klasse zu, ohne auszusagen wie sicher die Einordnung ist.

Erste Gehversuche wurden während des Semesters mit der Logit-Regression unternommen. Leider war die Lernkurve etwas zu steil und es fehlte an Vorwissen im Umgang mit MatLab.

#### 4.11.3 Weitere Klassifikatoren

Am Anfang der Projektarbeit stand die Idee im Raum mehrere verschiedene Klassifikatoren einzusetzen und diese zu beurteilen. Anschliessend können diese untereinander verglichen werden. Besprochen wurden mehrere hochinteressante Ansätze, die aus Zeitgründen nicht mehr berücksichtigt wurden:

- Training eines neuronalen Netzes für die Klassifikation
- Die Multivariate Regression
- Eine Support Vector Machine
- Einsatz einer Cloudbasierten Machine Learning Plattform wie Microsoft Azure

## 5 Schlussfolgerungen / Fazit

Die Arbeit an der Bachelorthesis lässt sich grob in drei Kapitel unterteilen:

1. Aus dem Bildarchiv sollte eine Stichprobe hergestellt werden. Hierzu wurde jedes Bild aus dem Archiv per Hand Klassifiziert.

Bevor dies geschehen konnte musste das Bildarchiv zunächst auf dem Dateisystem und in der Datenbank in eine handhabbare Form gebracht werden. Die Dauer die vielen Bilddateien von einem Verzeichnis in ein anderes zu verschieben ist beträchtlich. Der Windows Explorer sowie der FTP-Client kamen bei diesen Operationen oft an ihre Grenzen. Es musste erlernt werden, dass Operationen auf dem gesamten Bildarchiv häufig lange dauerten und nur häppchenweise erfolgreich durchgeführt werden konnten.

Auch die Grösse der Datenbank geriet während der Erfassung der statistischen Werte ausser Kontrolle. Sie wuchs von 80 Megabyte schnell auf den aktuellen Stand von 8 Gigabyte an. Diese Dateigrösse ist problematisch beim Verschieben zwischen Computern, da ein handelsüblicher Memorystick nur Dateien bis zu einer gewissen Maximalgrösse abspeichern kann.

Und schliesslich dauerte das Erfassen der Kategorien zu den Bildern einen Teil der Arbeitszeit. In 45 Arbeitsstunden wurden die Daten erhoben. Das Ergebnis dieser Bemühungen ist eine Datenbasis hoher Qualität, die für zukünftige Bearbeitung und Forschung nützlich sein kann.

2. Der zweite Teil der Arbeit fokussierte sich auf die statistische Auswertung der erfassten Daten. In einem agilen Prozess wurden verschiedene Wege der Datenverarbeitung implementiert, visualisiert und geprüft. Aus den Daten wurden schliesslich Kennzahlen und Referenzwerte extrahiert.
3. Im letzten Abschnitt ging schliesslich um die eigentliche Detektion. Es entspross ein einfacher Klassifikator, welcher aus dem Stand sehr gute Detektions-Ergebnisse lieferte. Auf Grund der guten Resultate könnte der Algorithmus bereits jetzt in einer produktiven Umgebung eingesetzt werden.

Geschmälert wird der Erfolg nur von zwei Umständen: Die Ergebnisse des Algorithmus konnten trotz verschiedener Optimierungsversuche nicht weiter verbessert werden. Und weiterhin konnte aus Zeitgründen kein weiterer Klassifikator implementiert werden.

## 6 Abbildungsverzeichnis

Abbildung 1: Schematischer Ablauf einer Detektion	2
Abbildung 2: Mobile Video Kamera 021	7
Abbildung 3: Mobile Video Kamera 022	7
Abbildung 4: Mobile Video Kamera 101	7
Abbildung 5: Mobile Video Kamera 102	8
Abbildung 6: Mobile Video Kamera 105	8
Abbildung 7: Mobile Video Kamera 106	8
Abbildung 8: Mobile Video Kamera 107	9
Abbildung 9: Mobile Video Kamera 108	9
Abbildung 10: Mobile Video Kamera 110	9
Abbildung 11: Mobile Video Kamera 120	10
Abbildung 12: Mobile Video Kamera 122	10
Abbildung 13: Mobile Video Kamera 131	10
Abbildung 14: Standorte der 12 Kameras auf Google Map	11
Abbildung 15: Bildschirmfoto des Galerie Moduls der Applikation	14
Abbildung 16: Bildschirmfoto des Segmentierungsmoduls der Applikation	14
Abbildung 17: Differenz aus zwei Kamerabildern und resultierende Bitmaske	16
Abbildung 18: Kandidatenbild mit schwarzen Löchern	16
Abbildung 19: Auffüllen von schwarzen Löchern im Kandidatenbild	18
Abbildung 20: Ausgeschnittene Bildsegmente von mvk110	19
Abbildung 21: Kategorie 'kein Schnee'	23
Abbildung 22: Kategorie 'Schnee'	23
Abbildung 23: Kategorie 'Tag'	23
Abbildung 24: Kategorie 'Dämmerung' (Morgen oder Abend)	23
Abbildung 25: Kategorie 'Nacht'	23
Abbildung 26: Kategorie 'Nebel'	23
Abbildung 27: Kategorie 'Niederschlag'	23
Abbildung 28: Kategorie 'schlechte Lichtverhältnisse' (Reflektionen oder direkte Sonneneinstrahlung)	23
Abbildung 29: Bildschirmfoto des Kategorieerfassungsmoduls	24
Abbildung 30: Bildschirmfoto des Galeriemoduls mit Kategorieerweiterungen	25
Abbildung 31: Neue Segmentierung für mvk110	28
Abbildung 32: Alle möglichen 8 Bit Farben (rot, blau und grün)	28
Abbildung 33: Ausgeschnittenes Bildsegment von mvk110 ohne Schnee	29
Abbildung 34: Ausgeschnittenes Bildsegment von mvk110 mit Schnee	29
Abbildung 35: Bildschirmfoto des Statistikmoduls	33
Abbildung 36: Berechnete statistische Werte eines Bildausschnitts ohne Schnee und mit Schnee	33
Abbildung 37: Klassendiagramm der erstellten Datenstrukturen	34
Abbildung 38: Bildschirmfoto des Plot Moduls. Plot der Häufigkeitsverteilung Medianwerts (BGR) der Kamera mvk120 und des Patch 142	36
Abbildung 39: Häufigkeitsverteilungen des Medianwerts für die Kamera mvk101 Links: Grass-Patch 110. Rechts Mittelstreifen-Patch 108	38

Abbildung 40: Bildschirmfoto des 2D Plot Moduls. Plot der zweidimensionalen Darstellung zweier Kennzahlen der Kamera mvk112 (Grass-Patch 145) X-Achse: Medianwert (BGR). Y-Achse: Standardabweichung (BGR)	39
Abbildung 41: Ausgewählter Patch im 2d Plot Modul	40
Abbildung 42: Plot der zweidimensionalen Darstellung zweier Kennzahlen der Kamera mvk101 (Grass-Patch 110) x-Achse: Mittelwert (blau). Y-Achse: Kontrast (blau) Linke Hälfte: Alle erfassten Datensätze für mvk101. Rechte Hälfte: Reduzierte Datensätze für mvk101	44
Abbildung 43: Bildschirmfoto des kNN Moduls.	46
Abbildung 44: Korrekt klassifiziertes Bild	47
Abbildung 45: Falsch klassifiziertes Bild	47

## 7 Tabellenverzeichnis

Tabelle 1: Mögliche Resultate einer Klassifizierung	22
Tabelle 2: Verhältnis zwischen Schneebildern und schneefreien Bildern pro Kameraperspektive	26
Tabelle 3: Jahreswochen mit Bildmaterial für Winter 2014 / 2015 und Winter 2015 / 2016	41
Tabelle 4: Alle möglichen Kategorie-Permutationen	42
Tabelle 5: F-Test für die kNN-Klassifikation	48
Tabelle 6: F-Test für die kNN-Klassifikation mit Median-Kombination	48
Tabelle 7: F-Test für die kNN-Klassifikation mit Input-Kombination	49
Tabelle 8: F-Test für die kNN-Klassifikation mit Input-Anreicherung aus mehreren Patches	51

## 8 Programmcodeausschnitte

Code 1: Bash-Skript für das Herunterladen der Bilder	6
Code 2: Auszug aus dem Header-Teil der Bilddatei mvk101_20141530_134001.jpg	12
Code 3: Serialisierung und Skalierung eines Polygons	15
Code 4: Berechnung der Absoluten Differenz zweier Bilder	16
Code 5: Bild Maskieren mit Bitmaske	16
Code 6: Ausfüllen von Löchern im Kandidatenbild mit Inhalten	17
Code 7: Ausschneiden eines Bildaussegments	19
Code 8: Berechnung der Durchschnittsfarbe einer Region mit Bitmaske	19
Code 9: Berechnung der Euklidischen Distanz zu den Referenzfarbwerten	19
Code 10: Erweiterung der Images-Tabelle um Kategorien	25
Code 11: Berechnung eines Histogramms	29
Code 12: Berechnung des Modalwerts	30
Code 13: Berechnung des Mittelwerts	30
Code 14: Berechnung des Medianwerts	31
Code 15: Berechnung der Standardabweichung	31
Code 16: Berechnung des Minimums, des Maximums und des Kontrasts	32
Code 17: Berechnung der statistischen Werte für eine Bild uns dessen Patches	33
Code 18: Statistische Werte in Datenbank abspeichern	35
Code 19: Generierung von Häufigkeitsverteilungen	37
Code 20: Zeichnen von Häufigkeitsverteilungen auf einer virtuellen Zeichnungsebene	37

Code 21: Zeichnen von statistischen Werten als Punkte auf der Zeichnungsebene	39
Code 22: Statistische Werte zeitlich kombinieren in Pseudocode	43
Code 23: Berechnung der Euklidischen Distanz zwischen zwei statistischen Werten	45
Code 24: kNN-Suche für statistische Werte	46
Code 25: Berechnung des Verhältnisses zwischen Schneebildern und schneefreien Bildern	47
Code 26: Inputbilder kombinieren für die kNN-Suche	49
Code 27: Ausreisser-Patches aus der Inputmenge vorzeitig eliminieren	51

## 9 Glossar

<b>Cron-Job</b>	Ein Computerprozess der in regelmässigen Intervallen eine Aufgabe erfüllt
<b>.NET</b>	Von Microsoft herausgegebene Software-Plattform, die der Entwicklung von Anwendungsprogrammen dient
<b>ASTRA</b>	Schweizer Strassenverkehrsamt
<b>Bash-Skript</b>	Sammlung von Kommandozeilen Befehlen für die automatisierte Ausführung auf dem Computer
<b>Bit</b>	Binäre Ziffer im Computer.
<b>Bitmaske</b>	Zweidimensionale Liste von Wahrheitswerten
<b>C#</b>	Von Microsoft entwickelte, typsichere, objektorientierte Programmiersprache
<b>Cloud (-Computing)</b>	Ist die Bereitstellung von IT-Infrastruktur und IT-Leistungen wie beispielsweise Speicherplatz, Rechenleistung oder Anwendungssoftware als Service über das Internet.
<b>Deserialisierung</b>	Umwandlung von Informationen in Textform in Programm-Objekte
<b>DPI</b>	Dots per Inch: Maßeinheit für die Auflösung im Druck und anderen Wiedergabesystemen
<b>Emgu CV</b>	Quelloffene .NET Programmierschnittstelle für OpenCV
<b>Exif</b>	Format fürs abspeichern von Meta-Information in Bilddateien
<b>GitHub</b>	Webbasierte Quellcodeverwaltung für Software-Entwicklungsprojekte
<b>JPG</b>	Dateiformat für komprimierte Bilddaten
<b>Json</b>	JavaScript Object Notation ist ein kompaktes Datenformat in einer einfach lesbaren Textform zum Zweck des Datenaustauschs zwischen Anwendungen.
<b>LINQ to SQL</b>	Language integrated Queries: Zuordnung eines Datenmodells einer relationalen Datenbank zu einem Objektmodell, dass in der Programmiersprache des Entwicklers ausgedrückt ist.
<b>MatLab</b>	Software zur Lösung mathematischer Probleme und grafischer Darstellung der Ergebnisse
<b>MDF</b>	Main Data File: Dateiformat für Datenbanken
<b>MVK</b>	Mobile Video Kamera
<b>OneDrive</b>	

## Cloudbasierter Speicher für Dateien von Microsoft

---

### **OpenCV**

Eine quelloffene Programmbibliothek mit Algorithmen für die Bildverarbeitung und maschinelles Sehen.

---

### **Serialisierung**

Umwandlung von Programm-Objekten in Textform

---

### **SQL**

Die Structured Query Language ist eine Datenbanksprache zur Definition von Datenstrukturen in relationalen Datenbanken sowie zum Bearbeiten (Einfügen, Verändern, Löschen) und Abfragen von darauf basierenden Datenbeständen.

---

### **SQL-Server**

Ein relationales Datenbankmanagementsystem von Microsoft.

---

### **Visual Studio**

Programmier-Entwicklungsumgebung von Microsoft

---

### **WGET**

Kommandozeilenprogramm zum Herunterladen von Dateien aus dem Internet

---

### **WPF**

Windows Presentation Foundation ist ein Grafik-Framework und Teil des .NET Frameworks von Microsoft

---

### **XAML**

Extensible Application Markup Language ist eine von Microsoft entwickelte allgemeine Beschreibungssprache für die Oberflächengestaltung von Anwendungen

---

## 10 Literaturverzeichnis

Hudritsch, M. (2016). *Grundlagen der Computer Vision*. Sutz: Berner Fachhochschule.

## 11 Selbständigkeitserklärung

Ich bestätige, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der im Literaturverzeichnis angegebenen Quellen und Hilfsmittel angefertigt habe. Sämtliche Textstellen, die nicht von mir stammen, sind als Zitate gekennzeichnet und mit dem genauen Hinweis auf ihre Herkunft versehen.

Ort, Datum:

Unterschrift: