Freie Universität Berlin

Bachelorarbeit am Institut für Informatik der Freien Universität Berlin

Dahlem Center for Intelligent Systems

Wetterklassifizierung mit Convolutional Neural Networks für das autonome Fahren

Lukas Felmy
Matrikelnummer: 5325120
lukas.felmy@fu-berlin.de

Betreuer: Nicolai Steinke, Prof. Dr. Daniel Göhring
Erstgutachter: Prof. Dr. Daniel Göhring
Zweitgutachter: Prof. Dr. Tim Landgraf

Berlin, 15. August 2022

Zusammenfassung

Extremwetter stellt für das autonome Fahren eine unverkennbare Herausforderung dar. Das Erkennen des Wetters erlaubt es, das Fahrverhalten entsprechend der Wetterbedingung automatisch anzupassen. Im Rahmen dieser Arbeit wurden verschiedene Convolutional Neural Networks zur Klassifizierung des Wetters auf Verkehrsbildern unter unterschiedlichen Trainingsumständen auf dem umfangreichen BDD100K-Datensatz trainiert und evaluiert. Als wesentlich für die Übertragbarkeit der Vorhersagefähigkeiten auf die Bilder des autonomen Fahrzeuges der Freien Universität Berlin, stellte sich dabei der Einsatz von Label Smoothing heraus. Die Evaluation zeigt, dass das System besondere Schwierigkeiten bei Bildern mit leichtem Regen und einer geringen Spiegelung auf der Straße hat und zu Fehlklassifikationen bei Tunnel-Bildern neigt. Die Vorhersagefähigkeit des Systems wird durch den Einsatz eines exponentiell gleitenden Durchschnitts gesteigert und weitere Verbesserungsvorschläge für zukünftige Arbeiten wurden gemacht.

Inhaltsverzeichnis

1 Einführung **7**
- 1.1 Motivation . . . 7
- 1.2 Verwandte Arbeiten . . . 7

2 Grundlagen **9**
- 2.1 Supervised Learning und Klassifizierung . . . 9
- 2.2 Neuronale Networks . . . 9
- 2.3 Convolutional Neural Networks . . . 10
- 2.4 Label-Kodierung . . . 11
- 2.5 Deep Transfer Learning . . . 12
- 2.6 Hyperparameter-Suche . . . 12
- 2.7 Metriken . . . 12
  - 2.7.1 Konfusionsmatrix . . . 13
  - 2.7.2 Genauigkeit . . . 13
  - 2.7.3 Recall . . . 14
  - 2.7.4 Precision . . . 14
  - 2.7.5 F1-Score . . . 14

3 Implementation **15**
- 3.1 Datensatz . . . 15
- 3.2 Alternative Datensätze . . . 16
- 3.3 Custom-Datensatz für Evaluation . . . 17
- 3.4 PyTorch . . . 17
- 3.5 Architekturen . . . 18
  - 3.5.1 ResNet . . . 18
  - 3.5.2 RegNetY . . . 18
- 3.6 Preprocessing-Schritte . . . 20
- 3.7 Training . . . 20

4 Evaluation **22**
- 4.1 Evaluation auf BDD-100K Testdatensatz . . . 22
- 4.2 Evaluation auf Custom-Testdatensatz . . . 23
- 4.3 Failure Cases . . . 25
- 4.4 Exponentiell gleitender Durchschnitt . . . 27

5 Zusammenfassung und Ausblick **29**

A Anhang **36**
- A.1 Ergänzende Information zu der Architektur . . . 36
- A.2 ROS-Node . . . 36
- A.3 Ergänzende Informationen zu den Datensätzen . . . 36
- A.4 Beispiele für nicht eindeutigem Wetter im Custom-Datensatz . . . 39
- A.5 Konfusionsmatrizen . . . 41
  - A.5.1 BDD100K-Datensatz . . . 41
  - A.5.2 Custom-Datensatz . . . 44

1 Einführung

1.1 Motivation

Eine Voraussetzungen von Level 5 der Fahrautomatisierung ist die Funktionsfähigkeit des autonomen Fahrzeuges unabhängig von den Wetterbedingungen, solange auch ein herkömmliches Fahrzeug unter diesen Bedingungen von einem menschlichen Fahrer vernünftig gefahren werden könnte[1, S. 32]. Doch noch immer stellt Extremwetter eine der größten Herausforderungen für das autonome Fahren dar.

Nebel reduziert die Sichtweite von LiDAR und die Bildqualität, was die Erkennbarkeit von beleuchteten Objekten, wie Ampeln, erschwert[2, S. 258-259]. Eine verschneite Straße führt zu Problemen mit der Selbstlokalisierung und Bahnplanung und Schneefall beeinträchtigt die Objekterkennung[2, S.259-260]. Auch Regen beeinträchtigt die Sichtweite von LiDAR und die Selbstlokalisierung[2, S. 258]. Durch Regentropfen auf der Kameralinse und Regenspritzer aus Regenpfützen wird die Objekterkennung gestört[2, S. 258].

Ein Grund für die Abnahme der Objekterkennungsqualität bei Regen und Schnee sind die scharfen Intensitätsschwankungen in den Bildern[3, S. 1].

Roh u. a.[4] zeigen in einem Experiment mit dem Fahrerassistenzsystem Mobile Eye 630, dass ein Niederschlag von 30 mm bei einer Geschwindigkeit ab $48\,\mathrm{km\,h^{-1}}$, aufgrund der Reduktion der Sichtweite, zu dessen Funktionsunfähigkeit führt. Bei geringeren Geschwindigkeiten ist das Ausmaß niedriger.

Das vorangegangene Beispiel verdeutlicht, dass das Wahrnehmen des Wetters für das autonome Fahren nützlich sein kann, um das Fahrverhalten an die Wetterbedingungen anzupassen.

Doch das Erkennen des vorliegenden Wetters ist in vielen Fällen nicht einfach. Die Fortschritte der vergangenen Jahre im Bereich der Computer Vision, ermöglichen jedoch heutzutage die erfolgreiche Klassifizierung des Wetters auf Bildern einer Kamera mittels state-of-art Convolutional Neural Networks.

1.2 Verwandte Arbeiten

Al-Haija u. a.[5] verwenden ResNet18 und Transfer-Learning für die Wetterklassifizierung für IOT-Anwendungen und Cyber-Physical-Systems. Sie erreichen eine Genauigkeit von 98.22% auf dem $1.1k$ Bilder großen MCWD Datensatz mit den Klassen *sunrise*, *shine*, *rain* und *cloudy*. Der Datensatz zeigt zwar Bilder mit verschiedenen Wetterarten, dennoch sind die Motive beliebig. Neben einigen Verkehrsbildern beinhaltet der Datensatz auch Zeichnungen von Personen, Graslandschaften, Videospiel-Charaktere und Sonnenuntergänge über dem Meer. Darunter befinden sich ebenfalls keine oder wenige Nachtaufnahmen. Daraus folgt eine geringe Eignung für das autonome Fahren.

Dhananjaya u. a.[6] stellen einen Datensatz mit 1.1*k* Bildern, für die Wetter- und Lichtlevelklassifizierung mit den Klassen *clear*, *rain*, *fog* und *snow* sowie den Klassen *bright*, *moderate* und *low* vor. Die Bilder sind aus der Perspektive eines Fahrzeuges und wurden zu verschiedenen Tageszeiten über einen Zeitraum von zwei Jahren gesammelt. Sie nutzen Transfer-Learning und ResNet18 und erreichen damit einen F1-Score von 77.2%. Der Datensatz beinhaltet jedoch ausschließlich Bilder aus ländlichen Regionen Deutschlands. Es ist fraglich, inwiefern die Ergebnisse auf große bevölkerungsreiche Städte mit einem anderen Landschaftsbild übertragbar sind.

Xia u. a.[7] schlagen ein neues vereinfachtes ResNet15-Modell vor, basierend auf ResNet-50, zur Klassifizierung des Wetters *foggy days*, *rainy days*, *snowy days* und *sunny days* auf Verkehrsbildern. Dabei erreicht das Modell eine durchschnittliche Genauigkeit von 96.03%. Der 4.9*k* Bilder große WeatherDataset-4 wurde hauptsächlich aus dem Internet zusammengestellt und ist nicht frei verfügbar.

Ibrahim u. a.[8] stellen das WeatherNet-Modell vor, bestehend aus 4 ResNet50-Modellen zur Klassifizierung von Verkehrsbildern mit den Klassen *day* und *night*, den Klassen *glare* und *no glare*, den Klassen *clear*, *rain* und *snow* sowie den Klassen *fog* und *no fog*. WeatherNet erreicht eine Genauigkeit von 91.6%, 94.8%, 93.2% und 95.6%. Der 23.8*k* Bilder große Datensatz wurde hauptsächlich über Google Images zusammengestellt, ist jedoch nicht frei verfügbar. Das vorgeschlagene Framework ist jedoch rechenintensiv[6, S. 2]. In autonomen Fahrzeugen teilen sich die verschiedenen Module begrenzte Ressourcen, demnach sollte jedes Modul der Aufgabe entsprechend angemessene Rechenleistung beanspruchen. Die vorangegangenen Arbeiten zeigen, dass auch kleinere Modelle hohe Performance bei der Wettererkennung erzielen können.

2 Grundlagen

2.1 Supervised Learning und Klassifizierung

Sei $X^{\text{tr}} = \{(x_1, y_1)...(x_n, y_n)\}$ ein Trainingsdatensatz mit Ein- und Ausgabepaaren. Die Aufgabe des Supervised Learning ist es, eine Funktion h zu finden, die eine unbekannte Funktion f approximiert, welche die Ausgabe $y_i = f(x_i)$ erzeugt[9, Kap. 19.1]. Die Leistung von h wird mittels eines Testdatensatz X^{te} evaluiert, welcher ausschließlich Elemente enthält, die nicht in X^{tr} enthalten sind[9, Kap. 19.1]. Wenn die Ausgabe der zu approximierenden Funktion Element einer endlichen Menge ist, nennt sich die Aufgabe Klassifizierung[9, Kap. 19.1].

2.2 Neuronale Networks

Für Supervised Learning und Klassifizierung eignen sich Artificial Neural Networks (ANN), welche Funktionen approximieren können, die als Tensor kodierte Eingaben x Ausgaben $\hat{y}$ zuordnen. Im Folgenden werden die Eigenschaften und Struktur von neuronalen Netzen und insbesondere die der Feedforward-Netze beschrieben. ANNs haben die Eigenschaft, von vornherein unbekannte Informationen in den Daten für die Vorhersage zu nutzen[10, S. 44]. Sie sind inspiriert vom Nervensystem und bestehen aus einer Vielzahl an künstlichen Neuronen, welche miteinander verbunden sind[11, S. 1][10, S. 44] und in übereinander liegenden Schichten angeordnet sind[10, S. 44]. Bei der ersten Schicht handelt es sich um die Eingabeschicht und die letzte Schicht ist die Ausgabeschicht[10, S. 44-45]. Die Schichten dazwischen nennen sich Hidden Layers[10, S. 45].

Ein künstliches Neuron akzeptiert üblicherweise stetige Eingaben und erzeugt eine stetige Ausgabe[9, Kap. 21.1], wobei einige dieser Eingaben Parameter des ANNs sind[9, Kap. 21.1].

Ein neuronales Netz operiert entweder im Trainingsmodus oder im Predictionmodus[10, S. 46]:

Die initialen Parameterwerte des ANNs können vor dem Training beliebig gewählt werden[10, S. 46]. Sie werden innerhalb mehrerer Iterationen während des Trainings so angepasst, dass ein Error[10, S. 46] bzw. eine Verlustfunktion minimiert wird. Das Ziel dabei ist es, dass das neuronale Netz generalisiert. Das bedeutet, dass unbekannten Eingaben nahezu korrekte Ausgaben zugeordnet werden[10, S. 47]. Eine Voraussetzung dafür, ist ein ausreichend großer Datensatz[10, S. 47]. Ist das nicht der Fall und die Vorhersagen auf Elemente des Testdatensatzes sind deutlich ungenauer als auf dem Trainingsdatensatz, dann kann es sich um eine Überanpassung des Modells an den Trainingsdatensatz handeln, welche als Overfitting bezeichnet wird[10, S. 47].

Im Predictionmodus fließt die Information durch das neuronale Netz von der Eingabe bis zur Ausgabe und das neuronale Netz erzeugt entsprechende Vorhersagen[10, S. 46-47].

Der Abbildung 1 ist ein reguläres fully-connected ANN zu entnehmen. Die künstlichen Neuronen einer Schicht sind jeweils mit sämtlichen künstlichen Neuronen der Vorgängerschicht verbunden.

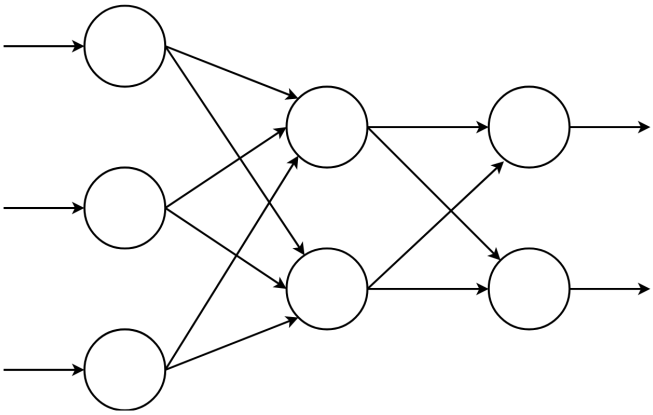

Abbildung 1: Beispiel eines regulären ANNs

2.3 Convolutional Neural Networks

Die rechnerische Komplexität, die bei Bildern benötigt wird, stellt für die regulären ANNs eine Herausforderung dar[11, S. 3]. Die Anzahl ihrer Parameter wächst substantiell mit der Größe der Eingabe. Die künstlichen Neuronen der ersten Schicht eines fully-connected ANN, welches Farbbilder, die als $x \in \mathbb{R}^{3\times224\times224}$ Tensor kodiert sind, akzeptiert, berechnen bereits aus jeweils über 150k Eingabewerte einen Ausgabewert.

Bei Convolutional Neural Networks (CNN) handelt es sich um eine Form von ANNs, welche hauptsächlich für die Mustererkennung in Bildern eingesetzt werden[11, S. 2]. Ihre Neuronen sind, anders als bei regulären ANNs, nur mit einer kleinen lokalen Region der vorherigen Schicht verbunden[11, S. 6]. Somit werden deutlich weniger Parameter benötigt, was die Wahrscheinlichkeit für Overfitting verringert[11, S. 3, 6].

Folgende Schichten finden üblicherweise Verwendung in CNNs:

Jedes **Convolutional Layer** arbeitet mit einer Anzahl an lernbaren Filtern, die in der Regel eine geringe räumliche Dimensionalität besitzen[11, S. 6]. Die Filter eines Convolutional Layers erzeugen jeweils eine 2D-Feature-Map, welche übereinandergestapelt werden[11, S. 6]. Derselbe Filter wird auf allen lokalen Regionen der Eingabe angewandt. Das erfolgt mit der Annahme, dass die Berechnung eines Features in einer lokalen Region auch für eine andere lokale Region nützlich sein kann[11, S. 7]. Die Filter in der ersten Schicht dienen der Erkennung einfacher Features und Filter in höheren Schichten erkennen abstraktere Features[12, S. 3].

Pooling Layer reduzieren die Dimensionalität der Feature Maps, um die Anzahl der nötigen Parameter weiter zu reduzieren[11, S. 8][12, S. 6].

Ähnlich wie in regulären ANNs, verbindet das **Fully Connected Layer** alle Neuronen

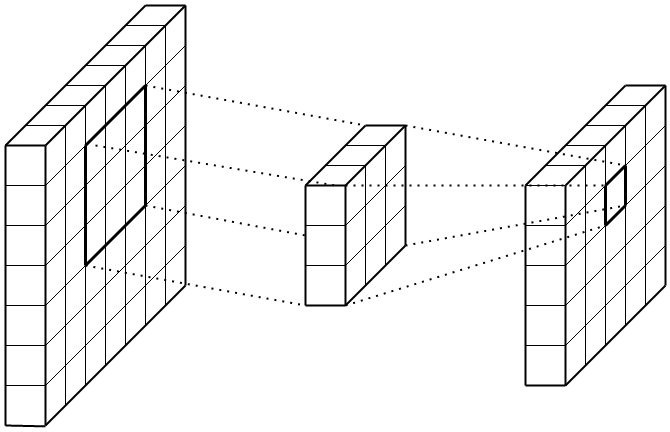

Abbildung 2: Convolution mit 3×3 großem rezeptiven Feld

der vorherigen Schicht mit allen Neuronen der aktuellen Schicht[11, S. 8]. Die Fully-Connected-Layer gehören üblicherweise zu den letzten Schichten eines Convolutional Neural Networks.

2.4 Label-Kodierung

Für die Mehrklassen-Klassifizierung bieten sich unterschiedliche Verlustfunktionen an. Üblicherweise wird der Cross Entropy Loss genutzt

$$\mathcal{L}(\hat{y}, y) = -\sum_{i=1}^{N} y_i * \log \hat{y}_i$$

in Kombination mit der Softmax-Funktion als letzte nichtlineare Aktivierungsfunktion.

$$\hat{y}_i = \text{softmax}(z)_i = \frac{\exp z_i}{\sum_{j=1}^{N} \exp z_j}$$

$\hat{y}$ stellt dann eine diskrete Wahrscheinlichkeitsverteilung dar mit $\hat{y}_i$ als Confidence (Zuversichtlichkeit) des Modells, dass die Eingabe x der Klasse i zugehörig ist. Dementsprechend ist dann $\underset{i}{\text{argmax}} \hat{y}_i$ die vorgeschlagene Klasse.

Sei i die tatsächliche Klasse der Eingabe x, dann wird dieses Label für das Training üblicherweise one hot-kodiert:

$$y_i^{\text{hot}} = 1 \text{ und } y_{j \neq i}^{\text{hot}} = 0$$

Das kann jedoch in einigen Fällen zu Overfitting führen[13, S. 7]. Alternativ kann die Klasse j mittels Label Smoothing[13] mit einem Label Smoothing-Wert $\epsilon \in [0, 1]$ kodiert werden:

$$y_i^{\text{sm}} = (1 - \epsilon) * y_i^{\text{hot}} + \frac{\epsilon}{N}$$

Diese Kodierung reduziert die Confidence des Modells. Außerdem kann Label Smoothing die Performance eines Modells steigern, das auf einem Datensatz mit inkorrekt gelabelten Elementen trainiert wird[14].

2.5 Deep Transfer Learning

Tan u. a.[15, S. 2] beschreiben Deep-Transfer-Learning als ein Werkzeug von Machine Learning, welches dazu dient, Wissen aus einer Quelldomäne in eine Zieldomäne zu übertragen. Als network based Deep Transfer Learning bezeichnen sie das Wiederverwenden eines Teilnetzes, welches in der Quelldomäne trainiert wurde[15, S. 6]. Transfer-Learning kann dann als Lösung eingesetzt werden, wenn die Trainingsdaten nicht ausreichend sind[15, S. 2].

In der vorliegenden Arbeit werden auf dem ImageNet-Datensatz vortrainierte neuronale Netze genutzt. ImageNet stellt die Quelldomäne dar und das Erkennen des Wetters die Zieldomäne.

Es ist üblich manuell einige Parameter vor dem Training in der Zieldomäne einzufrieren, jedoch zeigt die Arbeit von Gua u. a., dass das unter Umständen zu schlechteren Ergebnissen führen kann[16, S. 6]. Sie unterscheiden zwischen verschiedenen Fine-Tuning-Techniken u. a.: Standard-Fine-Tuning, bei dem sämtliche Netzwerk-Parameter auf die Zieldomäne abgestimmt werden und Feature Extractor, wobei ausschließlich eine trainierbare Klassifikationsschicht hinzugefügt wird[16, S. 5].

2.6 Hyperparameter-Suche

Vor dem Training eines ANN müssen eine Reihe von Hyperparameterwerten λ wie beispielsweise der Learning Rate-Wert festgelegt werden. Sie können sehr unterschiedliche Auswirkungen auf das resultierende Modell und seine Performance haben[17, S. 1]. Das Ziel der Hyperparameter-Suche ist es, die optimalen Hyperparameter λ^* zu finden, die den Verlust $\mathcal{L}(X^{\text{te}}; \mathcal{M})$ eines Modells $\mathcal{M}$ auf dem Testdatensatz X^{te} minimiert[17, S. 2]. Diese Funktion muss nicht der Verlustfunktion entsprechen, die beim Training eines Artificial Neural Networks minimiert wird.

Sei X^{tr} der Trainingsdatensatz und X^{te} der Testdatensatz und sei $\mathcal{M}$ das Modell, das durch das Lernverfahren $\mathcal{A}$ trainiert wird mit $\mathcal{M} = \mathcal{A}(X^{\text{tr}}; \lambda)$, dann lässt sich die Hyperparameter-Suche formal beschreiben mit[17, Gl. (1)]:

$$\lambda^* = \underset{\lambda}{\operatorname{argmin}} \mathcal{L}(X^{\text{te}}; A(X^{\text{tr}}; \lambda))$$

Das Suchen der Hyperparameter ist jedoch kostspielig, da das Training eines Modells $\mathcal{M}$ zeitaufwändig sein kann[17, S. 2] und üblicherweise wird ein neuronales Netz mit unterschiedlichen Hyperparametern für die Hyperparameter-Suche mehrmals trainiert.

2.7 Metriken

Es gibt verschiedene Metriken, die für die Evaluation eines Klassifizierers in Betracht kommen und dazu dienen dessen Performance zu messen. Im Folgenden werden Konfusionsmatrix, Genauigkeit, Recall, Precision und F1-Score vorgestellt, welche im Laufe der Arbeit Anwendung finden.

2.7.1 Konfusionsmatrix

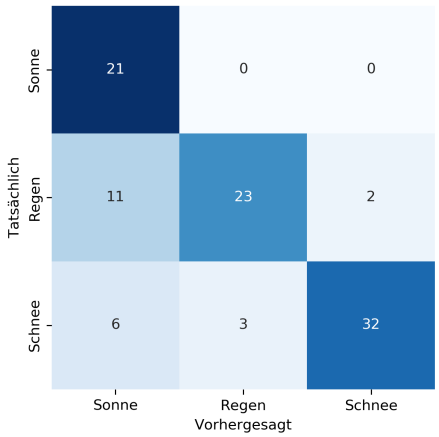

Abbildung 3: Beispiel einer Konfusionsmatrix mit den Klassen Sonne, Regen, Schnee

Die Konfusionsmatrix $C \in \mathbb{N}_0^{N \times N}$ für N Klassen gibt eine Übersicht über die Anzahl der vorgeschlagenen Klassen und die Anzahl der tatsächlichen Klassen. In Abbildung 3 ist beispielhaft eine Konfusionsmatrix zu sehen. Die Spalten sind vorgeschlagene Klassen und die Reihen die tatsächlichen Klassen. Das Feld $c_{i,j}$ in Zeile i und Spalte j ist die Anzahl der Fälle, in denen Klasse j vorgeschlagen wurde und die tatsächliche Klasse i ist. Daraus folgt, dass $c_{i,i}$ die Anzahl der Fälle ist, in denen Vertreter der Klasse i korrekt klassifiziert wurden. Die Genauigkeit, die Precision und der F1-Score können mittels der Konfusionsmatrix bestimmt werden. Die Formeln für den F1-Score, die Precision und den Recall sind aus den Beiträgen [18] und [19] abgeleitet. Die Konfusionsmatrizen werden in dieser Arbeit mit der Seaborn-Bibliothek[20] visualisiert.

2.7.2 Genauigkeit

Die Genauigkeit gibt an, welcher Anteil korrekt klassifiziert wurde. Das Bestimmen der Genauigkeit kann jedoch bei einem unausgewogenen Datensatz nicht ausreichend sein: Ein Klassifizierer, der immer die Mehrheitsklasse vorschlägt, kann so eine hohe Genauigkeit erzielen[21], die dennoch nicht für eine hohe Leistung des Klassifizierers spricht.

$$\text{Genauigkeit} = \frac{\sum_{i=1}^{N} c_{i,i}}{\sum_{i=0}^{N} \sum_{j=0}^{N} c_{i,j}}$$

2.7.3 Recall

Der Recall gibt an, welcher Anteil der tatsächlich Positiven korrekt klassifiziert wurde[18]. Bei dem Macro-Recall handelt es sich um das arithmetische Mittel der Recall-Werte der verschiedenen Klassen.

$$\text{Recall}_i = \frac{c_{i,i}}{\sum_{j=1}^{N} c_{i,j}}$$
$$\text{Macro-Recall} = \frac{\sum_{i=1}^{N} \text{Recall}_i}{N}$$

2.7.4 Precision

Die Precision gibt an, welcher Anteil der vorgeschlagenen Positiven tatsächlich positiv ist[18]. Bei der Macro-Precision handelt es sich um das arithmetische Mittel der Precision-Werte der verschiedenen Klassen.

$$\text{Precision}_i = \frac{c_{i,i}}{\sum_{j=0}^{N} c_{j,i}}$$
$$\text{Macro-Precision} = \frac{\sum_{j=1}^{N} \text{Precision}_j}{N}$$

2.7.5 F1-Score

In der Regel gibt es einen Kompromiss zwischen guter Präzision und gutem Recall, denn wird versucht, die eine Metrik zu verbessern, vermindert das häufig die andere[18][19]. Der F1-Score fasst beide Werte mittels des harmonischen Mittels zu einem einzigen Wert zusammen[19]. Bei dem Macro-F1-Score handelt es sich um das arithmetische-Mittel der F1-Score-Werte der verschiedenen Klassen.

$$\text{F1-Score}_j = 2 * \frac{\text{Precision}_j * \text{Recall}_j}{\text{Precision}_j + \text{Recall}_j}$$
$$\text{Macro-F1-Score} = \frac{\sum_{j=1}^{N} \text{F1-Score}_j}{N}$$

.

3 Implementation

3.1 Datensatz

Um ein neuronales Netz zu trainieren, ist ein ausreichend großer, diverser und relevanter Datensatz nötig. Da nur wenige Schneefahrten in den Bag-Files vorliegen und das manuelle Annotieren einer großen Datenmenge aufwändig ist, muss auf einen externen Datensatz wie BDD100K zurückgegriffen werden.

BDD100K[22] ist ein umfangreicher und diverser Datensatz mit über 100k Videoclips von Fahrszenen. Die Daten stammen vorwiegend aus vielen Städten und Regionen in den bevölkerungsreichen Gebieten der USA[22, Abb. 2]. Der Datensatz ist auf der Grundlage von Crowdsourcing entstanden und wurde in über 50k Fahrten gesammelt, um hohe Diversität in den Daten zu ermöglichen[22, S. 3]. Er enthält unterschiedliche Szenen, darunter Stadtstraßen, Wohngebiete und Autobahnen zu verschiedenen Tageszeiten. 70k Bilder mit einer Auflösung von 1280×720 wurden aus 70k Videos extrahiert und die 10te Sekunde u. a. für Klassifizierung der Wetterbedingung annotiert. Es existieren nach einer Überarbeitung von 2020 sieben Wetterklassen: *clear*, *overcast*, *snowy*, *rainy*, *cloudy*, *foggy* und *undefined*. Der Verteilung in Abbildung 4 und Abbildung 15 im Anhang ist zu entnehmen, dass der Datensatz unausgewogen ist.

BDD100K ist frei verfügbar, denn die Lizenz erlaubt die kostenfreie Nutzung, Modifikation und Verbreitung für Bildung, Forschung und gemeinnützige Zwecke.[23]

Die Verkehrsbilder des Datensatzes sind aus der Perspektive eines Fahrzeuges. Das macht den Datensatz besonders geeignet für die Wetterklassifizierung von Kamerabildern eines autonomen Fahrzeuges. Es ist jedoch fraglich, wie gut ein neuronales Netz, das auf diesem Datensatz trainiert wurde, für ländliche Regionen Deutschlands geeignet ist, denn er besteht hauptsächlich aus Bildern stark bewohnter Gegenden.

Da nur wenige Vertreter der *foggy*-Klasse existieren, werden sie genau wie die *undefined* Klasse aus dem Datensatz entfernt. Ob bewölktes oder bedecktes Wetter vorliegt, hat für das autonome Fahren zwar keine hohe Relevanz, dennoch kann beides als Indiz für bevorstehende Wetterwechsel verstanden werden.

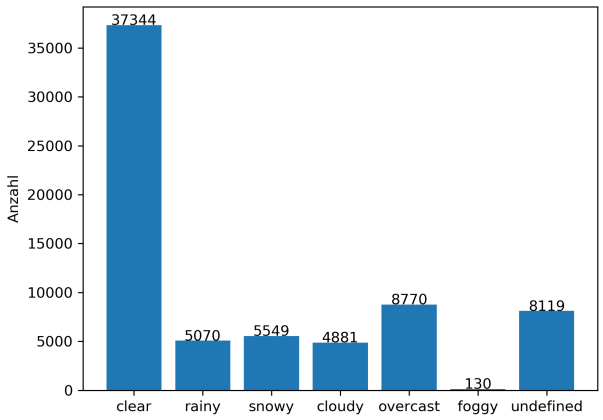

Abbildung 4: Verteilung der Klassen in BDD100K-Trainingsdatensatz

3.2 Alternative Datensätze

Im Folgenden werden weitere Datensätze, die für die Wetterklassifizierung auf Straßenszenen in Frage kommen, genannt:

Der Oxford RobotCar-Datensatz[24] stellt eine der Alternativen zum BDD100K-Datensatz dar. Er beinhaltet nahezu 20 Mio. Bilder, die innerhalb eines Jahres von verschiedenen Fahrten unter unterschiedlichen Wetterbedingungen aufgezeichnet wurden. Er fand Anwendung in einer Arbeit, die sich ebenfalls mit Wetter und dem autonomen Fahren befasst[25]. Es existiert jedoch nur eine Fahrt mit Schnee, was womöglich nicht ausreichend ist, um Schnee auf ungesehenen Bildern anderer Fahrten zu erkennen.

Ein weiterer Datensatz ist DAWN[26]. Er dient der Fahrzeugerkennung unter Extremwetter, dennoch kann er für die Erkennung von *snow*, *rain*, *fog* und *sand* in den 1000 Verkehrsbildern genutzt werden. Aufgrund der Tatsache, dass die Bilder kein sonniges bzw. klares Wetter enthalten, ist er für die Wetterklassifizierung in dieser Arbeit nicht geeignet.

Der 4006 Verkehrsbilder große ACDC-Datensatz[27] aus der Schweiz mit den Klassen *fog*, *nighttime*, *rain* und *snow* wurde für die semantische Segmentierung unter Extrembedingungen konzipiert. Es ist jedoch festzustellen, dass sämtliche Vertreter der *fog*-, *rain*- und *snow*-Klasse ausschließlich am Tag aufgenommen wurden. Dementsprechend ist er nicht geeignet für die Wetterklassifizierung von Nachtaufnahmen.

SeeingThroughFog[28] wurde als Datensatz für Objekterkennung in Extremwetter konzipiert. Die Daten stammen aus Fahrten in Deutschland, Schweden, Dänemark und Finnland mit einer Gesamtlänge von über 10000km und aus Nebelkammern. Das Wetter *clear*, *rain*, *light fog*, *dense fog* und *snow* wurde auf 28k Bildern annotiert.

3.3 Custom-Datensatz für Evaluation

Um die Performance der Modelle auf den Bildern der autonomen Fahrzeuge der Freien Universität Berlin testen zu können, wurden für die vorliegende Arbeit Bilder aus den Bag Files von über 20 Fahrten in Berlin extrahiert. Bei jeder Fahrt wurde alle 15 Sekunden das Bild der Front- und Rückkamera mit Zeitpunkt und Geokoordinaten gespeichert. Anders als im BDD100K-Datensatz besitzen die Kameras ein Fischaugenobjektiv und erzeugen Bilder mit einer Auflösung von 1280×800. Die Fahrten fanden ausschließlich am Tag statt; es gibt keine Aufzeichnungen von Nachtfahrten. Die Abbildung 16 im Anhang zeigt Beispielbilder für die Klassen *clear*, *rainy*, *snowy*, *cloudy* und *overcast*.

Statt jedes Bild einzeln einem Wetter zuzuordnen, teilen sich alle Bilder einer Fahrt dasselbe Wetter. Dieses Vorgehen ermöglicht das Labeln der Daten ohne immensen Zeitaufwand. Jedoch ist das Wetter nicht auf jedem Bild eindeutig zu erkennen: Zum Beispiel können bei der Durchfahrt eines Tunnels die Bilder keinem Wetter zugeordnet werden, denn der Himmel ist nicht zusehen. Weitere Beispiele sind den Abbildungen 17, 18, 19, 21 und 20 im Anhang zu entnehmen.

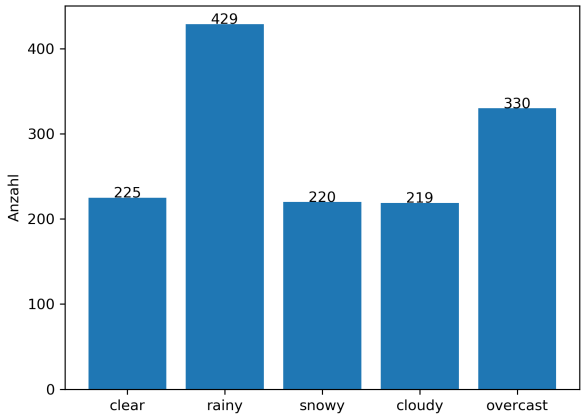

Abbildung 5: Verteilung der Klassen in Custom-Datensatz

3.4 PyTorch

Für die vorliegende Arbeit wird das Open-Source-Machine-Learning-Framework PyTorch[29] genutzt. Es reduziert den Aufwand der Implementierung und des Trainierens eines neuronalen Netzes und bietet Hardwarebeschleunigung auf der GPU. Ein weiterer Vorteil von PyTorch ist, dass bekannte Architekturen, wie ResNet und Architekturen des RegNetY-Design Spaces, implementiert sind und auf ImageNet vortrainierte Modelle vorliegen, die in dieser Arbeit für Transfer Learning genutzt werden.

3.5 Architekturen

3.5.1 ResNet

He u. a.[30] lösen mit ResNet ein Problem, das beim Training tiefer neuronaler Netze auftritt und bei steigender Tiefe zu abnehmender Genauigkeit führt: Sei $\mathcal{H}(x)$ die Abbildung, die durch eine Folge von Schichten approximiert werden soll. He u. a. empfehlen, anstatt $\mathcal{H}(x)$ die Abbildung $\mathcal{F}(x) := \mathcal{H}(x) - x$ zu approximieren und zeigen, dass dies dem neuronalen Netz leichter fällt. Aus diesem Grund nutzen sie für ihre ResNet18- und Resnet34-Architekturen einen Baustein mit zwei Schichten und Shortcut-Connections wie in Abbildung 6 zu sehen.

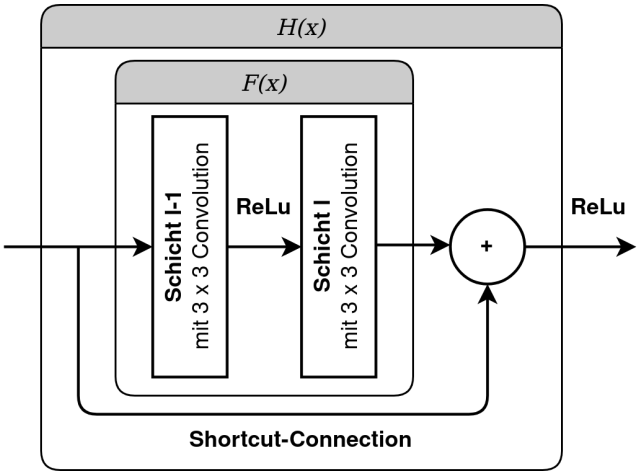

Abbildung 6: ResNet-Baustein ohne Flaschenhals - Darstellung basiert auf Abbildung [30, Abb. 2]

3.5.2 RegNetY

In der Arbeit von Radosavovic u. a.[31], wird das Verhalten eines Design Space untersucht und versucht, allgemeine Gestaltungsprinzipien, für einfachere und performantere Modelle zu finden.

Radosavovic u. a.[32] erklären den Unterschied zwischen Modellfamilien und Design Spaces wie folgt: Eine Modell-Familie, wie ResNet, ist eine Sammlung von verwandten ANN-Architekturen, die sich in der Regel einige übergeordnete architektonische Strukturen oder Konstruktionsprinzipien teilen, jedoch meist nicht vollständig spezifiziert sind[32, S. 3]. Ein Design Space ist eine konkrete Menge an Modellarchitekturen und besteht aus einer Parametrisierung einer Modellfamilie und einer Menge von validen Werten für die Parameter[32, S. 3].

Bei dem initialen Design Space handelt es sich um AnyNet mit einem Stamm, Körper und Kopf. AnyNet hat 16 Parameter: Jede der 4 Stufen des Körpers besteht aus einer Folge von d_i Bausteinen, die jeweils die Stufenbreite w_i, Bottleneck-Ratio b_i und Group

Width g_i besitzen. Nach jeder Stufe erfolgt die Reduktion der Auflösung der Feature-Maps.

Der RegNet-Design-Space wird durch die schrittweise Reduktion der Anzahl an Parameter des Design Spaces auf 6 unter Berücksichtigung der erklärten Absichten[31, S. 4] konstruiert:

- die Struktur des Design-Spaces zu vereinfachen
- die Interpretierbarkeit des Design-Spaces zu verbessern
- die Qualität des Design-Spaces zu verbessern oder zu erhalten
- die Modellvielfalt im Design-Spaces zu erhalten

Das erfolgt u. a. durch den Einsatz einer quantisierten linearen Funktion, welche die Stufentiefe d_i und Stufenbreite w_i mit nur 3 Parametern beschreibt. Der Design Space basierend auf RegNet mit Squeeze-and-Excitation Operationen ist RegNetY. Der Baustein einer RegNetY-Architektur ist der Abbildung 7 zu entnehmen. Unter vergleichbaren Trainingsumständen übertreffen Modelle aus diesem Design Space state-of-art Efficient Net-Modellen unter den meisten FLOP-Regimes und sind dabei bis zu fünf Mal schneller[31, Tab. 4].

Ein gut gestalteter Design Space wie RegNetY mit unterschiedlich großen Architekturen erleichtert es, eine Architektur mit angemessener Größe und Struktur zu finden, die gut geeignet ist für die entsprechende Klassifikationsaufgabe. Für die vorliegende Arbeit wurde RegNetY1.6GF verwendet, da sie eine vergleichbare Anzahl an Parametern wie die Baseline-Architektur ResNet18, die in verwandten Arbeiten verwendet wurde, besitzt.

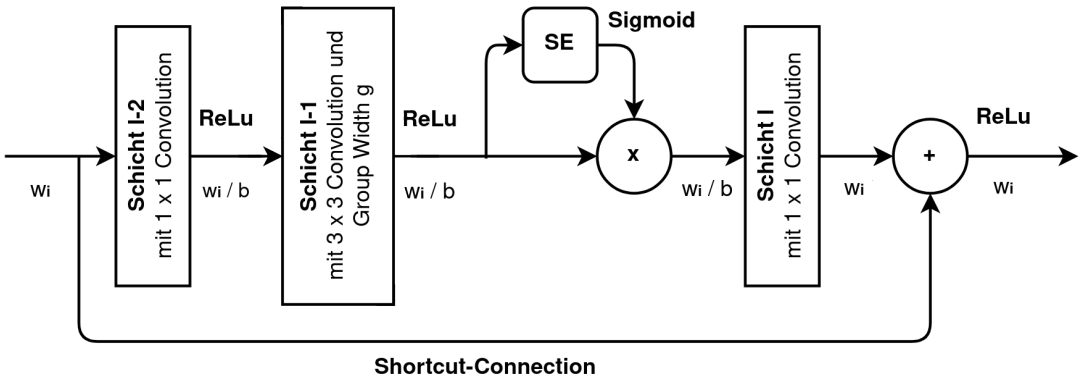

Abbildung 7: Baustein einer Architektur aus dem RegNetY-Design-Space - SE stellt die Verwendung von Squeeze-and-Excitation Operationen dar - Darstellung basiert auf Abbildung [31, Abb. 4]

3.6 Preprocessing-Schritte

Die Bilder des BDD100K-Datensatzes werden beim Training und bei der Evaluation asynchron aus dem Festplattenspeicher geladen.

Entweder wird ein Bildausschnitt mit 16:9-Verhältnis oder mit 1:1-Verhältnis genutzt. Die Bilder werden auf die für RegNet und ResNet benötigte Auflösung von 224×224 herunterskaliert. Einerseits weisen die Bilder mit vormals einem Verhältnis von 16:9 eine starke Verzerrung auf. Andererseits liegt bei dem Bildausschnitt mit vormals einem Verhältnis von 1:1 ein Informationsverlust vor, da Teile des Bildes nicht enthalten sind.

In der Arbeit zur Wetter- und Lichtlevelklassifizierung für das autonome Fahren, stellen die Autoren fest, dass die Verringerung der Auflösung nur einen minimalen Einfluss auf die Performance ihres Modells hat[6, S. 3].

Da der BDD100K-Datensatz ausnahmslos Frontkamerabilder enthält, dient eine zufällige Spiegelung dieser Bilder nicht nur als Regularisierung, sondern auch zur Erzeugung von Bildern, die der einer Rückkamera ähneln. Sämtliche Preprocessing-Schritte sind der Abbildung 8 zu entnehmen.

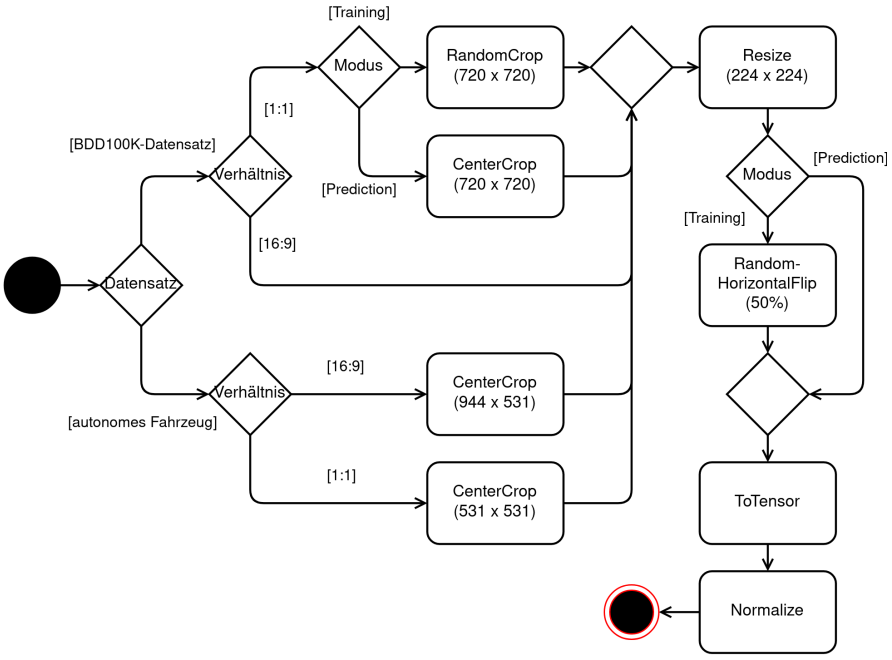

Abbildung 8: Preprocessing-Schritte

3.7 Training

Der BDD100K Datensatz bietet einen Trainingsdatensatz X^{tr} und einen Testdatensatz X^{te}, die für die Klassifizierung des Wetters genutzt werden können. Der Trainingsdatensatz wird in 2 Datensätze zufällige geteilt $X^{\text{tr}'} \cup X^{\text{te}'} = X^{\text{tr}}$, sodass sich 80% der Vertreter der Klasse i in $X^{\text{tr}'}$ befinden und die restlichen in $X^{\text{te}'}$. Die Batch Size ist

Nr.	Architektur	Fine-Tuning	Verhältnis	LR	LR-Decay[a]	Label-Smth.
(1)	RegNetY1.6GF	Std. FT.[b]	16:9	$2.03\text{E}-4$	$3.94\text{E}-1$	0.0
(2)	RegNetY1.6GF	—	16:9	$1.17\text{E}-3$	$7.20\text{E}-1$	0.0
(3)	ResNet18	Std. FT.	16:9	$1.23\text{E}-4$	$5.51\text{E}-1$	0.0
(4)	RegNetY1.6GF	Std. FT.	1:1	$3.54\text{E}-4$	$6.39\text{E}-1$	0.0
(5)	RegNetY1.6FG	Feature Extr.[c]	1:1	$3.47\text{E}-3$	$7.55\text{E}-1$	0.0
(6)	RegNetY800MF	Std. FT.	16:9	$3.32\text{E}-4$	$4.54\text{E}-1$	0.0
(7)	RegNetY1.6GF	Std. FT.	16:9	$2.79\text{E}-4$	$2.86\text{E}-1$	0.4
(8)	RegNetY1.6GF	teilweise[d]	16:9	$1.02\text{E}-3$	$6.88\text{E}-1$	0.0

Tabelle 1: Modelle und ihre Trainingsumstände

[a] Faktor um den die Learning Rate alle $\left\lceil \frac{|X^{\text{tr}'}|}{B} \right\rceil$ bzw. $\left\lceil \frac{|X^{\text{tr}}|}{B} \right\rceil$ Iterationen reduziert wird

[b] Standard Fine-Tuning

[c] Feature Extractor

[d] Die Parameter des Stammes und der ersten Stufe sind eingefroren.

64 und die Softmax-Funktion stellt die letzte nicht-lineare Aktivierungsfunktion dar. AdamW dient als Optimizer und Cross Entropy Loss als Verlustfunktion:

Für das Training wird Weighted Random Sampling genutzt, da es verhindern soll, dass das neuronale Netz bei einem unausgewogenen Datensatz einen Bias gegenüber einer Klasse entwickelt.

Es werden Modelle mit unterschiedlichen Architekturen und unter verschiedenen Trainingsumständen trainiert. Die Wetterklassen sind nicht klar und eindeutig unterscheidbar. Da Label Smoothing impliziert, dass eine Eingabe nicht in Gänze genau einer Klasse zugeordnet werden kann, wird ein Modell mit dieser Kodierung anstatt der One-Hot-Kodierung trainiert.

Mit dem Open-Source-Framework Optuna[33] werden innerhalb von 20 Wiederholungen mit jeweils $8\left\lceil \frac{|X^{\text{tr}'}|}{B} \right\rceil$ Iterationen die optimalen Hyperparameter auf dem Trainingsdatensatz $X^{\text{tr}'}$ und Testdatensatz $X^{\text{te}'}$ gesucht. Sie können der Tabelle 1 entnommen werden.

Daraufhin erfolgt das ausführliche Training innerhalb von $35\left\lceil \frac{|X^{\text{tr}}|}{B} \right\rceil$ Iterationen auf dem Trainingsdatensatz X^{tr} und dem Testdatensatz X^{te} mit den gefundenen optimalen Hyperparameterwerten.

4 Evaluation

Die Evaluation aller Modelle erfolgt sowohl auf dem BDD100K-Testdatensatz als auch auf dem Custom-Datensatz.

4.1 Evaluation auf BDD-100K Testdatensatz

Die Tabelle 2 zeigt die Performance der verschiedenen Modelle auf dem BDD100K-Testdatensatz. Das Modell mit der höchsten Genauigkeit und dem höchsten Macro-F1-Score ist Modell *(1)*. Die Abbildung 9 zeigt dessen Konfusionsmatrix und alle weiteren Konfusionsmatrizen können im Anhang gefunden werden. Das Modell *(1)*, welches auf Bildern mit Verzerrung und ohne Informationsverlust trainiert wurde, schneidet besser als Modell *(4)* ab.

Die unterschiedliche Performance von Modell *(1)* und Modell *(2)* zeigt, dass Fine-Tuning das Ergebnis verbessert. Aus den Experimenten geht Standard Fine-Tuning als effektivste Fine-Tuning-Technik hervor.

Auch wenn das ResNet18-Modell *(3)* mehr Parameter besitzt als das RegNetY1.6GF-Modell *(1)*, erzielt es eine geringere Performance.

Der Vergleich zwischen Modell *(1)* und Modell *(7)* zeigt, dass keine Performance-Steigerung auf dem Testdatensatz zu erkennen ist, wenn Label Smoothing mit einem hohen Label Smoothing-Wert genutzt wird.

Letztlich zeigt die Evaluation, dass das RegNetY1.6-Modell *(1)* mit Standard-Fine-Tuning auf dem 1280×720 großem Bildausschnitt ohne Label-Smoothing die messbar höchste Klassifizierungsqualität auf dem BDD100K-Testdatensatz hat.

Der Tabelle 3 ist zu entnehmen, dass die größte Schwierigkeit für das Modell *(1)* die *cloudy*-Klasse darstellt. In 22.4% der Fälle, in denen bewölktes Wetter klassifiziert wird, ist die tatsächliche Klasse *clear* und 22.0% der Bilder mit tatsächlich bewölktem Wetter, werden als bedeckt klassifiziert.

Nr.	Recall	Precision	F1-Score	Genauigkeit
(1)	78.4%	78.3%	**78.1%**	**85.1%**
(2)	70.2%	67.7%	68.7%	78.7%
(3)	75.9%	76.2%	76.0%	83.9%
(4)	76.0%	76.5%	76.0%	83.9%
(5)	68.1%	56.4%	59.9%	66.9%
(6)	76.9%	77.2%	76.8%	84.5%
(7)	**79.3%**	76.6%	77.7%	84.6%
(8)	73.9%	**79.1%**	76.1%	84.3%

Tabelle 2: Ergebnisse auf BDD100K-Testdatensatz: Macro-Recall, Macro-Precision, Macro-F1-Score, Genauigkeit

Der Tabelle 4 zeigt, dass das vorgeschlagene Modell *(1)* im Vergleich zu den ver-

clear	rainy	snowy	cloudy	overcast
92.0%	76.1%	80.9%	65.1%	76.4%

Tabelle 3: F1-Scores des Modells *(1)* auf dem BDD100K-Testdatensatz

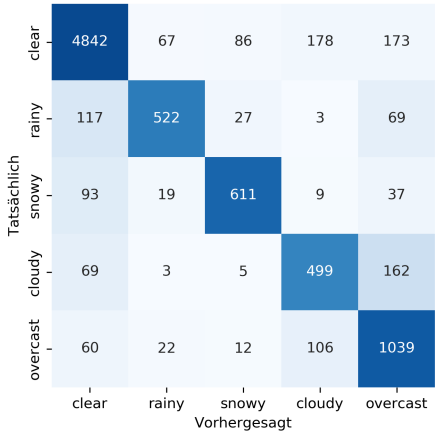

Abbildung 9: Konfusionsmatrix von Modell *(1)* mit dem höchsten F1-Score und der höchsten Genauigkeit auf BDD100K-Testdatensatz

wandten Arbeiten schlechtere Ergebnisse erzielt. Es übertrifft lediglich den F1-Score der in der Arbeit von Dhananjaya u. a.[6] erreicht wird. Ein neuronales Netz kann nur so gut sein, wie der Datensatz auf dem es trainiert wurde. Es liegt nahe, dass der BDD100K-Datensatz nicht optimal für die Wetterklassifizierung auf Verkehrsbildern ist.

System	Recall	Precision	F1-Score	Genauigkeit
Modell *(1)*	78.4%	78.3%	78.1%	85.1%
Al-Haija u. a.[5]	—	—	—	**98.2%**
Dhananjaya u. a.[6]	—	—	77.2%	—
Xia u. a.[7]	**96.0%**	—	—	
Ibrahim u. a. (PrecipitationNet)[8]	93.2%	**95.9%**	**94.7%**	93.2%

Tabelle 4: Vergleich der Ergebnisse mit verwandten Arbeiten. Die verwandten Arbeiten wurden auf anderen Datensätzen mit anderen Klassen trainiert und evaluiert.

4.2 Evaluation auf Custom-Testdatensatz

Die Evaluation auf dem Custom-Testdatensatz erfolgt auf einem Ausschnitt der 1280 × 800 großen Bilder des Custom-Datensatzes. Die Abbildung 10 zeigt die Größte und Position dieses Ausschnitts. Da beim Testdatensatz zu jedem Zeitpunkt sowohl Front- als auch Rückkamerabilder mit demselben Wetter zur Verfügung stehen, kann das arithmetische Mittel der beiden Vorschläge bestimmt werden, um die Vorhersagen zu

Nr.	Kamera	Recall	Precision	F1-Score	Genauigkeit
(1)	Frontkamera	63.2%	76.1%	61.0%	60.4%
(1)	Rückkamera	67.8%	77.1%	65.5%	63.5%
(1)	arithm. Mittel	68.1%	79.6%	66.3%	64.6%
(2)	arithm. Mittel	62.2%	72.9%	59.9%	59.7%
(3)	arithm. Mittel	66.7%	78.2%	64.5%	63.2%
(4)	arithm. Mittel	65.1%	79.7%	61.7%	60.2%
(5)	arithm. Mittel	63.3%	65.8%	62.5%	63.2%
(6)	arithm. Mittel	71.7%	81.6%	69.1%	67.5%
(7)	arithm. Mittel	**78.2%**	**85.7%**	**78.5%**	**77.8%**
(8)	arithm. Mittel	61.6%	75.8%	58.9%	58.3%

Tabelle 5: Ergebnisse der Modelle auf dem Custom-Datensatz: Macro-Recall, Macro-Precision, Macro-F1-Score, Genauigkeit

präzisieren:

$$\hat{y}_i = \frac{\hat{y}_i^{\text{front}} + \hat{y}_i^{\text{rear}}}{2}$$

Die Resultate der Evaluation sind der Tabelle 5 zu entnehmen. Das Modell, das auf dem BDD100K-Testdatensatz die beste Performance erzielt *(1)*, weist auf den Rückkamerabildern eine messbar höhere Leistung auf als auf den Frontkamerabildern. Es liegt nahe, dass der Grund in der durchschnittlich stärkeren und häufigeren Verschmutzung der Frontkamera liegt. Das Mittel der Vorschläge der Front- und Rückkamerabilder erhöht den Macro-Recall, die Macro-Precision, den Macro-F1-Score und die Genauigkeit des Modells *(1)*. Aus diesen Gründen werden alle folgenden Modelle ausschließlich mit dem arithmetischen Mittel der Front- und Rückkameravorschläge evaluiert.

Im Schnitt schneiden die Modelle auf dem BDD100K deutlich besser ab als auf den Bildern der Bag Files. Die Ausnahme stellt das Modell *(7)* mit der Label Smoothing-Kodierung dar.

Es lässt sich nicht mit Sicherheit feststellen, weshalb dieses Modell das Wetter auf dem Custom-Datensatz mit großem Abstand am besten erkennen kann, aber nicht auf dem BDD100K-Testdatensatz. Eine mögliche Erklärung ist, dass das Modell *(7)* BDD100K-spezifische Features für die Klassifizierung niedriger gewichtet bzw. weniger Overfitting aufweist. Da diese Features nicht auf Bildern des Custom-Datensatzes zu finden sind, führt das zur Reduktion von Fehlklassifikationen. Der BDD100K-Datensatz enthält, anders als der Custom-Datensatz, Bilder mit regnerischem Wetter und deutlich zu erkennende Regentropfen auf der Frontscheibe und auf der Motorhaube. Des Weiteren ist nicht auszuschließen, dass einige Bilder des Custom-Datensatz anders gelabelt sind, als vergleichbare Bilder des BDD100K-Datensatzes. Die niedrigere Confidence des Modells *(7)* könnte dann in einigen Grenzfällen dazu führen, dass diese Bilder dennoch korrekt klassifiziert werden. Ob das tatsächlich zutrifft, ist schwer festzustellen. Wie in Abschnitt 2.4 erwähnt, kann Label Smoothing helfen, wenn der Trainingsdatensatz inkorrekt gelabelte Elemente enthält, jedoch ist fraglich, ob das auch für den Testdatensatz der Fall ist.

Abbildung 10: Die weiße Umrandung zeigt den Bildausschnitt auf dem die Modelle evaluiert werden. Die Größe und Position ist identisch für die Front- und Rückkamerabilder. Er stellt sicher, dass die Ränder des Kameragehäuses nicht zu sehen sind.

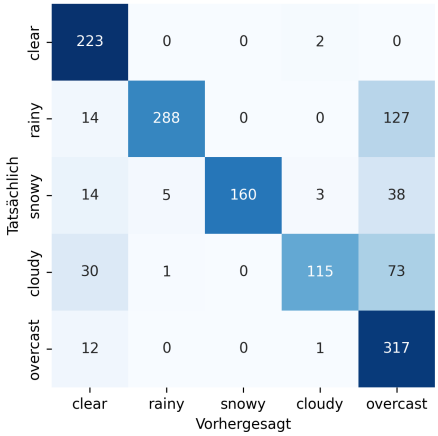

Abbildung 11: Konfusionsmatrix von Modell *(7)* auf dem Custom-Dataset durch Bildung des arithmetischen Mittels der Front- und Rückkameravorschläge

4.3 Failure Cases

In diesem Abschnitt erfolgt die Betrachtung der Fehlerfälle des Modells *(7)* auf dem Custom-Datensatz. Dabei handelt es sich um jene Bilder, welche mit der höchsten Confidence $\max_i \hat{y}_i$ inkorrekt klassifiziert wurden. Sie sind der Abbildung 12 zu entnehmen und können drei Gruppen zugeordnet werden: Fälle, in denen Regen als bedeckt klassifiziert wird, Tunnel-Bilder und bewölktes Wetter, welches als bedeckt erkannt wird.

Der Großteil der Fehler ergibt sich in Fälle, in denen regnerisches Wetter als bedeckt klassifiziert wird. Dabei handelt es sich hauptsächlich um Bilder auf der Stadtautobahn mit leichtem Regen und einer Straße, welche kaum oder wenig spiegelt. Es liegt nahe, dass das Modell Regen an einem bedeckten Himmel und Reflexion auf der

Straße erkennt. Die Anzahl der Fehlklassifikationen ließe sich reduzieren, indem der Trainingsdatensatz mit mehr Bildern von leichtem Regen angereichert werden würde.

Befindet sich das Fahrzeug in einem Tunnel, werden die Bilder als klar erkannt. Dass das Wetter auf diesen Bildern nicht korrekt klassifiziert werden kann, ist nachvollziehbar, denn der Himmel ist nicht zu erkennen. Es ist anzunehmen, dass das Modell diese Bilder als Nachtaufnahmen fehlinterpretiert und die Tunneldecke als klaren Nachthimmel ausmacht. Das ließe sich beheben, indem Tunnelszenen als solche identifiziert werden und entweder als eigene Wetterklasse behandelt werden oder den Tunnel-Bildern kein Wetter zugeordnet würde.

Einige Bilder mit bewölktem Wetter werden als bedeckt klassifiziert. Ein ähnliches Verhalten konnte bei dem Modell *(7)* auf dem BDD100K-Testdatensatz im Abschnitt 4.1 festgestellt werden. Es ist anzunehmen, dass diese Fehlklassifikationen auftreten, da die Trennung der Klassen im Trainingsdatensatz nicht ausreichend definiert ist und vergleichbare Vertreter mal der einen und mal der anderen Klasse zugeordnet sind.

(a) cloudy
vorherg.: overcast
confidence: 76.5%

(b) rainy
vorherg.: overcast
confidence: 74.3%

(c) cloudy
vorherg.: clear
confidence: 72.6%

(d) cloudy
vorherg.: overcast
confidence: 72.3%

(e) rainy
vorherg.: overcast
confidence: 72.1%

(f) rainy
vorherg.: overcast
confidence: 72.1%

(g) cloudy
vorherg.: overcast
confidence: 70.5%

(h) overcast
vorherg.: clear
confidence: 70.4%

(i) cloudy
vorherg.: overcast
confidence: 70.3%

(j) rainy
vorherg.: overcast
confidence: 70.3%

(k) rainy
vorherg.: overcast
confidence: 69.1%

(l) rainy
vorherg.: overcast
confidence: 68.5%

(m) rainy
vorherg.: overcast
confidence: 68.5%

(n) rainy
vorherg.: overcast
confidence: 68.3%

(o) overcast
vorherg.: clear
confidence: 68.3%

(p) cloudy
vorherg.: overcast
confidence: 68.2%

Abbildung 12: Fehlklassifikationen mit höchster Confidence des Modells *(7)* auf Custom-Dataset

4.4 Exponentiell gleitender Durchschnitt

Die Wetterklassifizierung erfolgt im autonomen Fahrzeug während einer Fahrt nicht auf einzelnen Bildern, sondern auf einer Folge von Bildern $(x_t)_{t\in\{1\ldots T\}}$ mit überwiegend identischem Wetter: Die Kameras nehmen stetig neue Bilder auf und ein Wetter-

wechsel ist dabei eine seltene Ausnahme. Um den Vorschlag $\hat{y}_T \in \mathbb{R}^N$ des Modells $\mathcal{M}$ zum Zeitpunkt T zu stabilisieren und zu verbessern, können die vorherigen Vorschläge $(\hat{y}_t)_{t\in\{1..T-1\}}$ in die Bestimmung der aktuellen Wetterbedingung einfließen. Eine Möglichkeit dafür bietet der exponentiell gleitende Durchschnitt mit $q \in [0,1]$:

$$\hat{y}^{\text{avg}}_{i,T} = \begin{cases} \hat{y}_{i,T} * q + (1-q) * \hat{y}^{avg}_{i,T-1}, & T \geq 1 \\ \frac{1}{N}, & T = 0 \end{cases}$$

Kleinere Werte für q bedeuten einen größeren Einfluss der vorherigen Vorhersagen und größere Werte bedeuten einen höheren Einfluss der aktuellen Vorschläge. Ein Kompromiss zwischen verbesserten und stabilisierten Vorhersagen und der rechtzeitigen Anpassung bei Wetterwechseln muss gefunden werden.

Um zu zeigen, dass dieses Verfahren die Qualität der Wetterklassifizierung steigert, wird das Modell *(7)* erneut auf dem Custom-Datensatz mit dem exponentiell gleitenden Durchschnitt evaluiert. Die Fahrten werden 10-mal in zufälliger Reihenfolge durchlaufen, die Bilder jeder einzelnen Fahrt jedoch in korrekter zeitlicher Abfolge. Der Übergang zwischen Fahrten mit unterschiedlichem Wetter simuliert dabei einen Wetterwechsel.

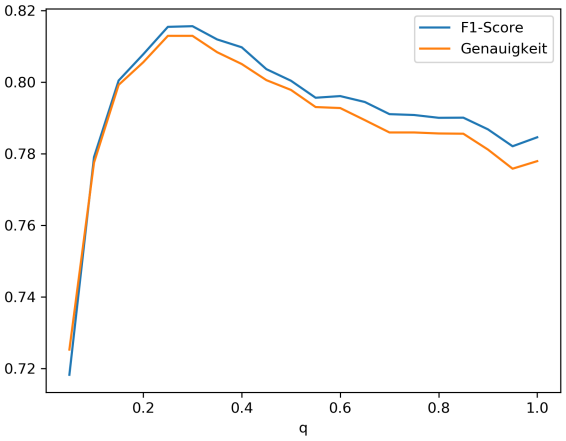

Abbildung 13: Macro F1-Score und Genauigkeit von Modell *(7)* mit exponentiell gleitenden Durchschnitt bei verschiedenen Werten für q auf dem Custom-Datensatz

Die Abbildung 13 zeigt den F1-Score und die Genauigkeit bei unterschiedlichen Werten für q. Die höchste Genauigkeit und F1-Score wird für $q = 30\%$ erreicht. Im Vergleich zu $q = 100\%$ ist eine leichte Verbesserung der Genauigkeit von 3.5% und des F1-Scores von 3.4% zu vermerken. Es muss jedoch beachtet werden, dass die Bilder im Custom-Datensatz alle 15s aufgenommen wurden. In dem selbstfahrenden Fahrzeug könnten die Vorhersagen mit einer höheren Frequenz erzeugt werden, was mutmaßlich mit einem geringeren q-Wert eine höhere Verbesserung zur Folge hätte. Ebenfalls ist anzunehmen, dass ein Wetterwechsel seltener stattfindet als dieses Experiment suggeriert. Des Weiteren wurde angenommen, dass sämtliche Wetterübergänge gleich wahrscheinlich sind. Das ist in der Realität nicht der Fall.

5 Zusammenfassung und Ausblick

Für diese Arbeit wurde ein System implementiert, welches das Wetter auf Verkehrsbildern als *clear*, *rainy*, *snowy*, *cloudy* oder *overcast* klassifiziert.

Das Training der Convolutional Neural Networks mit verschiedenen Architekturen und unter unterschiedlichen Trainingsumständen erfolgte auf dem umfangreichen BDD100K-Trainingsdatensatz.

Die Modelle wurden sowohl auf dem Testdatensatz evaluiert als auch auf den Bag Files der autonomen Fahrzeuge der Freien Universität Berlin. Sie offenbaren, dass die Modelle dieser Arbeit eine meist geringere Performance aufweisen als die Modelle verwandter Arbeiten, welche auf anderen Datensätzen trainiert wurden.

Als wesentlich für die Übertragbarkeit der Vorhersagefähigkeiten auf die Bilder des autonomen Fahrzeuges stellt sich dabei der Einsatz von Label Smoothing heraus.

Besondere Schwierigkeiten hat dieses Modell in Situationen, in denen nur leichter Regen vorliegt und die Straße eine geringe Spiegelung aufweist, das Fahrzeug durch einen Tunnel fährt oder in Situationen, in denen der Himmel stärker bewölkt ist und das Wetter fälschlicherweise als bedeckt statt bewölkt klassifiziert wird.

Das vorgeschlagene System ist aufgrund der mittelmäßigen Genauigkeit von 77.8% auf den Bildern der autonomen Fahrzeuge der Freien Universität Berlin, nicht für den realen Einsatz und als allein als Entscheidungsgrundlage in kritischen Situationen geeignet. Dennoch hat sich die zugrundeliegende Architektur als effektiv erwiesen, das Beste aus einem unzureichenden Datensatz zu holen.

In zukünftigen Arbeiten sollte daher ein anderer Datensatz als der BDD100K-Datensatz für die Wetterklasifizierung auf Verkehrsbildern in Betracht gezogen werden. Es bietet sich an stattdessen entsprechende Bilder aus dem Internet zusammenzustellen und manuell zu labeln, was jedoch zeitintensiv ist. Alternativ käme SeeingThroughFog[28] oder eine Kombination aus Datensätzen, welche in Abschnitt 3.2 genannt wurden, in Frage.

Die Verwendung eines exponentiell gleitenden Durchschnitts hat sich positiv auf die Performance des Systems ausgewirkt. Dieser Effekt kann durch frequentere Vorschläge mutmaßlich weiter gesteigert werden. Alle Wetterübergänge wurden als gleich wahrscheinlich betrachtet. Der Einsatz von Hidden-Markov-Modellen mit dem tatsächlichen Wetter als hidden States und dem vorgeschlagenen Wetter als observed States, könnte in zukünftigen Arbeiten die Vorhersagen bei unwahrscheinlichen Wetterübergängen als Fehlklassifikationen identifizieren und somit die Performance des Systems im Betrieb weiter verbessern.

Da sich die Verwendung von Fine-Tuning als gute Wahl herausgestellt hat, könnte die Qualität der Klassifikation mutmaßlich gesteigert werden durch andere Fine-Tuning Algorithmen, wie SpotTune[16] o.ä.

Literatur

[1] On-Road Automated Driving (ORAD) committee, *Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles*, Apr. 2021. DOI: https://doi.org/10.4271/J3016_202104. Adresse: https://doi.org/10.4271/J3016_202104.

[2] K. Yoneda, N. Suganuma, R. Yanase und M. Aldibaja, "Automated driving recognition technologies for adverse weather conditions," *IATSS Research*, Jg. 43, Nr. 4, S. 253–262, 2019, ISSN: 0386-1112. DOI: https://doi.org/10.1016/j.iatssr.2019.11.005. Adresse: https://www.sciencedirect.com/science/article/pii/S0386111219301463.

[3] K. Garg und S. Nayar, "When does a camera see rain?" In *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*, Bd. 2, 2005, 1067–1074 Vol. 2. DOI: 10.1109/ICCV.2005.253.

[4] C.-G. Roh, J. Kim und I.-J. Im, "Analysis of Impact of Rain Conditions on ADAS," *Sensors*, Jg. 20, Nr. 23, 2020, ISSN: 1424-8220. DOI: 10.3390/s20236720. Adresse: https://www.mdpi.com/1424-8220/20/23/6720.

[5] Q. A. Al-Haija, M. A. Smadi und S. Zein-Sabatto, "Multi-Class Weather Classification Using ResNet-18 CNN for Autonomous IoT and CPS Applications," in *2020 International Conference on Computational Science and Computational Intelligence (CSCI)*, 2020, S. 1586–1591. DOI: 10.1109/CSCI51800.2020.00293.

[6] M. M. Dhananjaya, V. R. Kumar und S. Yogamani, *Weather and Light Level Classification for Autonomous Driving: Dataset, Baseline and Active Learning*, 2021. DOI: 10.48550/ARXIV.2104.14042. Adresse: https://arxiv.org/abs/2104.14042.

[7] J. Xia, D. Xuan, L. Tan und L. Xing, "Resnet15: Weather recognition on traffic road with deep convolutional neural network," *Advances in Meteorology*, Jg. 2020, S. 1–11, 2020. DOI: 10.1155/2020/6972826.

[8] M. R. Ibrahim, J. Haworth und T. Cheng, *WeatherNet: Recognising weather and visual conditions from street-level images using deep residual learning*, 2019. DOI: 10.48550/ARXIV.1910.09910. Adresse: https://arxiv.org/abs/1910.09910.

[9] S. Russell und P. Norvig, *Artificial Intelligence: A Modern Approach*, 4. Aufl. Pearson Education, 2020.

[10] D. Svozil, V. Kvasnicka und J. Pospichal, "Introduction to multi-layer feed-forward neural networks," *Chemometrics and Intelligent Laboratory Systems*, Jg. 39, Nr. 1, S. 43–62, 1997, ISSN: 0169-7439. DOI: https://doi.org/10.1016/S0169-7439(97)00061-0. Adresse: https://www.sciencedirect.com/science/article/pii/S0169743997000610.

[11] K. O'Shea und R. Nash, *An Introduction to Convolutional Neural Networks*, 2015. DOI: 10.48550/ARXIV.1511.08458. Adresse: https://arxiv.org/abs/1511.08458.

[12] J. Gu, Z. Wang, J. Kuen u. a., *Recent Advances in Convolutional Neural Networks*, 2015. DOI: 10.48550/ARXIV.1512.07108. Adresse: https://arxiv.org/abs/1512.07108.

[13] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens und Z. Wojna, *Rethinking the Inception Architecture for Computer Vision*, 2015. DOI: 10.48550/ARXIV.1512.00567. Adresse: https://arxiv.org/abs/1512.00567.

[14] M. Lukasik, S. Bhojanapalli, A. K. Menon und S. Kumar, *Does label smoothing mitigate label noise?* 2020. DOI: 10.48550/ARXIV.2003.02819. Adresse: https://arxiv.org/abs/2003.02819.

[15] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang und C. Liu, *A Survey on Deep Transfer Learning*, 2018. DOI: 10.48550/ARXIV.1808.01974. Adresse: https://arxiv.org/abs/1808.01974.

[16] Y. Guo, H. Shi, A. Kumar, K. Grauman, T. Rosing und R. Feris, *SpotTune: Transfer Learning through Adaptive Fine-tuning*, 2018. DOI: 10.48550/ARXIV.1811.08737. Adresse: https://arxiv.org/abs/1811.08737.

[17] M. Claesen und B. De Moor, *Hyperparameter Search in Machine Learning*, 2015. DOI: 10.48550/ARXIV.1502.02127. Adresse: https://arxiv.org/abs/1502.02127.

[18] B. Shmueli, *Multi-class metrics made simple, part I: Precision and recall*, Juli 2019. Adresse: https://towardsdatascience.com/multi-class-metrics-made-simple-part-i-precision-and-recall-9250280bddc2.

[19] B. Shmueli, *Multi-class metrics made simple, part II: The F1-score*, Juli 2019. Adresse: https://towardsdatascience.com/multi-class-metrics-made-simple-part-ii-the-f1-score-ebe8b2c2ca1 (besucht am 01. 07. 2022).

[20] M. L. Waskom, "seaborn: statistical data visualization," *Journal of Open Source Software*, Jg. 6, Nr. 60, S. 3021, 2021. DOI: 10.21105/joss.03021. Adresse: https://doi.org/10.21105/joss.03021.

[21] J. Mohajon, *Accuracy on imbalanced datasets and why, You need confusion matrix!* Mai 2020. Adresse: https://medium.com/analytics-vidhya/accuracy-on-imbalanced-datasets-and-why-you-need-confusion-matrix-937613bf89bf (besucht am 01. 07. 2022).

[22] F. Yu, H. Chen, X. Wang u. a., *BDD100K: A Diverse Driving Dataset for Heterogeneous Multitask Learning*, 2018. DOI: 10.48550/ARXIV.1805.04687. Adresse: https://arxiv.org/abs/1805.04687.

[23] BDD100K-Lizens. Adresse: https://doc.bdd100k.com/license.html (besucht am 29. 06. 2022).

[24] W. Maddern, G. Pascoe, C. Linegar und P. Newman, "1 Year, 1000km: The Oxford RobotCar Dataset," *The International Journal of Robotics Research (IJRR)*, Jg. 36, Nr. 1, S. 3–15, 2017. DOI: 10.1177/0278364916679498. eprint: http://ijr.sagepub.com/content/early/2016/11/28/0278364916679498.full.pdf+html. Adresse: http://dx.doi.org/10.1177/0278364916679498.

[25] V. Mușat, I. Fursa, P. Newman, F. Cuzzolin und A. Bradley, "Multi-weather city: Adverse weather stacking for autonomous driving," in *2021 IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*, 2021, S. 2906–2915. DOI: 10.1109/ICCVW54120.2021.00325.

[26] M. Kenk, *DAWN*, V3, März 2020. DOI: 10.17632/766ygrbt8y.3.

[27] C. Sakaridis, D. Dai und L. Van Gool, *ACDC: The Adverse Conditions Dataset with Correspondences for Semantic Driving Scene Understanding*, 2021. DOI: 10.48550/ARXIV.2104.13395. Adresse: https://arxiv.org/abs/2104.13395.

[28] M. Bijelic, T. Gruber, F. Mannan u. a., "Seeing Through Fog Without Seeing Fog: Deep Multimodal Sensor Fusion in Unseen Adverse Weather," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Juni 2020.

[29] A. Paszke, S. Gross, F. Massa u. a., *PyTorch: An Imperative Style, High-Performance Deep Learning Library*, 2019. DOI: 10.48550/ARXIV.1912.01703. Adresse: https://arxiv.org/abs/1912.01703.

[30] K. He, X. Zhang, S. Ren und J. Sun, *Deep Residual Learning for Image Recognition*, 2015. DOI: 10.48550/ARXIV.1512.03385. Adresse: https://arxiv.org/abs/1512.03385.

[31] I. Radosavovic, R. P. Kosaraju, R. Girshick, K. He und P. Dollár, *Designing Network Design Spaces*, 2020. DOI: 10.48550/ARXIV.2003.13678. Adresse: https://arxiv.org/abs/2003.13678.

[32] I. Radosavovic, J. Johnson, S. Xie, W.-Y. Lo und P. Dollár, *On Network Design Spaces for Visual Recognition*, 2019. DOI: 10.48550/ARXIV.1905.13214. Adresse: https://arxiv.org/abs/1905.13214.

[33] T. Akiba, S. Sano, T. Yanase, T. Ohta und M. Koyama, *Optuna: A Next-generation Hyperparameter Optimization Framework*, 2019. DOI: 10.48550/ARXIV.1907.10902. Adresse: https://arxiv.org/abs/1907.10902.

Abbildungsverzeichnis

1 Beispiel eines regulären ANNs 10

2 Convolution mit 3×3 großem rezeptiven Feld 11

3 Beispiel einer Konfusionsmatrix mit den Klassen Sonne, Regen, Schnee 13

4 Verteilung der Klassen in BDD100K-Trainingsdatensatz 16

5 Verteilung der Klassen in Custom-Datensatz 17

6 ResNet-Baustein ohne Flaschenhals - Darstellung basiert auf Abbildung [30, Abb. 2] 18

7 Baustein einer Architektur aus dem RegNetY-Design-Space 19

8 Preprocessing-Schritte 20

9 Konfusionsmatrix von Modell *(1)* mit dem höchsten F1-Score und der höchsten Genauigkeit auf BDD100K-Testdatensatz 23

10 Bildausschnitte für Evaluation auf Custom-Datensatz 25

11 Konfusionsmatrix von Modell *(7)* auf dem Custom-Dataset durch Bildung des arithmetischen Mittels der Front- und Rückkameravorschläge 25

12 Fehlklassifikationen mit höchster Confidence des Modells *(7)* auf Custom-Dataset 27

13 Macro F1-Score und Genauigkeit von Modell *(7)* mit exponentiell gleitenden Durchschnitt bei verschiedenen Werten für q auf dem Custom-Datensatz 28

14 RegNetY1.6GF-Architektur 36

15 Verteilung der Klassen in BDD100K-Testdatensatz 37

16 Beispielbilder des Custom-Datasets 38

17 Beispiel für nicht eindeutiges Wetter: *snow* 39

18 Beispiel für nicht eindeutiges Wetter: *clear* 39

19 Beispiel für nicht eindeutiges Wetter: *overcast* 40

20 Beispiel für nicht eindeutiges Wetter: *cloudy* 40

21 Beispiel für nicht eindeutiges Wetter: *rainy* 40

22 Konfusionsmatrix von Modell *(2)* auf BDD100K-Testdatensatz 41

23 Konfusionsmatrix von Modell *(3)* auf BDD100K-Testdatensatz 41

24 Konfusionsmatrix von Modell *(4)* auf BDD100K-Testdatensatz 42

25 Konfusionsmatrix von Modell *(5)* auf BDD100K-Testdatensatz 42

26 Konfusionsmatrix von Modell *(6)* auf BDD100K-Testdatensatz 42

27 Konfusionsmatrix von Modell *(7)* auf BDD100K-Testdatensatz 43

28 Konfusionsmatrix von Modell *(8)* auf BDD100K-Testdatensatz 43

29 Konfusionsmatrix von Modell *(1)* auf dem Custom-Dataset durch Bildung des arithmetischen Mittels der Front- und Rückkameravorschläge 44

30 Konfusionsmatrix von Modell *(1)* auf den Frontkamerabildern des Custom-Datensatzes 44

31 Konfusionsmatrix von Modell *(1)* auf den Rückkamerabildern des Custom-Datensatzes 45

32 Konfusionsmatrix von Modell *(2)* auf dem Custom-Dataset durch Bildung des arithmetischen Mittels der Front- und Rückkameravorschläge 45

33 Konfusionsmatrix von Modell *(3)* auf dem Custom-Dataset durch Bildung des arithmetischen Mittels der Front- und Rückkameravorschläge 46

34 Konfusionsmatrix von Modell *(4)* auf dem Custom-Dataset durch Bildung des arithmetischen Mittels der Front- und Rückkameravorschläge 46

35 Konfusionsmatrix von Modell *(5)* auf dem Custom-Dataset durch Bildung des arithmetischen Mittels der Front- und Rückkameravorschläge 47

36 Konfusionsmatrix von Modell *(6)* auf dem Custom-Dataset durch Bildung des arithmetischen Mittels der Front- und Rückkameravorschläge 47

37 Konfusionsmatrix von Modell *(8)* auf dem Custom-Dataset durch Bildung des arithmetischen Mittels der Front- und Rückkameravorschläge 48

38 Konfusionsmatrix von Modell *(7)* auf dem Custom-Dataset durch Bildung des arithmetischen Mittels der Front- und Rückkameravorschläge und dem exponentiell gleitenden Durchschnitt 48

Tabellenverzeichnis

1 Modelle und ihre Trainingsumstände 21

2 Ergebnisse auf BDD100K-Testdatensatz: Macro-Recall, Macro-Precision, Macro-F1-Score, Genauigkeit 22

3 F1-Scores des Modells *(1)* auf dem BDD100K-Testdatensatz 23

4 Ergebnisse verwandter Arbeiten 23

5 Ergebnisse der Modelle auf dem Custom-Datensatz: Macro-Recall, Macro-Precision, Macro-F1-Score, Genauigkeit 24

6 Bag-Files mit Wetterbedingung und Anzahl der Front- und Rückkamera-Bilderpaare 37

A Anhang

A.1 Ergänzende Information zu der Architektur

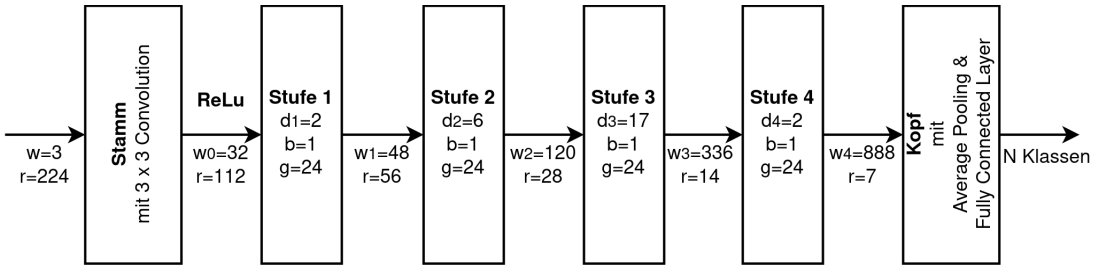

Abbildung 14: RegNetY1.6GF-Architektur

A.2 ROS-Node

Die entsprechende ROS-Node für die Wetterklassifizierung ist in Python implementiert und kann als Modul im autonomen Fahrzeug eingesetzt werden. Sie subscribt auf die Topics auf denen die Front- und Rückkamerabilder gepublisht werden. Sämtliche Preprocessing-Schritte werden durch die Node berechnet: darunter Cropping, Resizing und Normalization. Die Node klassifiziert die Bilder mit dem Modell *(7)* und bildet das arithmetische Mittel der Vorschläge auf den beiden Bildern, wie in Abschnitt 4.2 und nutzt einen exponentiell gleitenden Durchschnitt, wie aus Abschnitt 4.4 zu entnehmen. Insgesamt benötigt die Berechnung auf einem System mit einer Nvidia RTX 3060-Grafikkarte, dem Intel i5 12400F-Prozessor und 16GB DDR4 Arbeitsspeicher 0.045s, was einer Bildrate von $22.2s^{-1}$ entspricht.

A.3 Ergänzende Informationen zu den Datensätzen

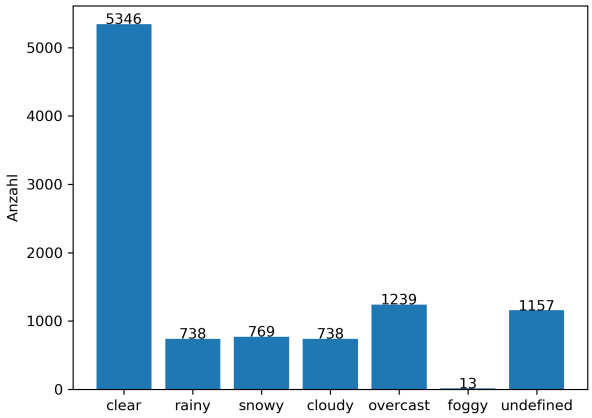

Abbildung 15: Verteilung der Klassen in BDD100K-Testdatensatz

Bag File	**Wetter**	**Anzahl**
20190909-181528 0200–regen	rainy	127
20200312-132818 0100–tegel	cloudy	71
20201127-120300 0100–thielallee_autonom2	cloudy	69
20201203-123102 0100–autonom_berliner	overcast	77
20211130-140335 0100–send_pole2	rainy	77
20210210-110725 0100–thiel_1	snowy	50
20210210-115420 0100–thiel_2	snowy	14
20210210-120755 0100–thiel_3	snowy	61
20210112-111850 0100–second_lap	rainy	38
2020210824-150259 0200	clear	11
20210824-170010 0200–2FU	cloudy	30
20191205-130317 0100–auto	clear	35
20200715-144048 0200–reinickendorf2fu	rainy	117
2020210211-113745 0100–to_reinickendorf	snowy	95
20210419-134221 0200–counterclockwise_small_3	cloudy	49
20200715-120340 0200–fu2reinickendorf	rainy	70
20210421-101252 0200–2reinickendorf	clear	92
20210917-133256 0200–big-lap-4	overcast	89
20220304-134924 0100–back	overcast	133
20220311-103513 0100	clear	87
20220616-105839 0200	overcast	31

Tabelle 6: Die Tabelle zeigt die Bag Files, deren Bilder für den Custom-Datensatz genutzt wurden, das vorliegende Wetter und die Anzahl der Bilderpaare. Die Ausnahme ist 20210210-120755 0100–thiel_3, bei der nur Rückkamerabilder vorliegen. Sie werden dennoch für die Evaluation genutzt, da es an Schneefahrten mangelt.

(a) clear

(b) rainy

(c) snowy

(d) cloudy

(e) overcast

Abbildung 16: Beispielbilder des Custom-Datasets

A.4 Beispiele für nicht eindeutigem Wetter im Custom-Datensatz

Abbildung 17: Der Fahrt ist zwar eindeutig verschneites Wetter zuzuordnen, aber in diesem Teil der Fahrt ist kein Schnee zu sehen und die Linse ist beschmutzt.

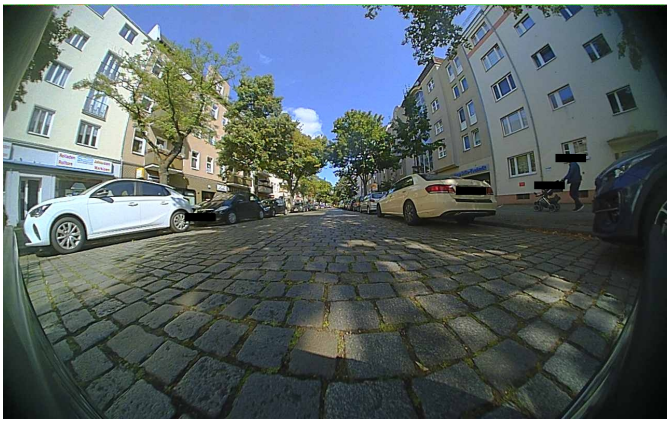

Abbildung 18: Der Fahrt ist zwar eindeutig klares Wetter zuzuordnen, aber in diesem Teil der Fahrt sind wenige Wolken zu sehen.

Abbildung 19: Der Fahrt ist zwar eindeutig bedecktes Wetter zuzuordnen, aber in diesem Abschnitt fährt das Fahrzeug durch einen Tunnel und kein Wetter ist zu erkennen.

Abbildung 20: Der Fahrt ist zwar eindeutig bewölktes Wetter zuzuordnen, aber in diesem Abschnitt der Fahrt könnte ebenfalls bedecktes Wetter erkannt werden.

Abbildung 21: Der Fahrt ist zwar eindeutig regnerisches Wetter zuzuordnen, aber in diesem Abschnitt der Fahrt ist keine Spiegelung auf der Straße zu erkennen.

A.5 Konfusionsmatrizen

A.5.1 BDD100K-Datensatz

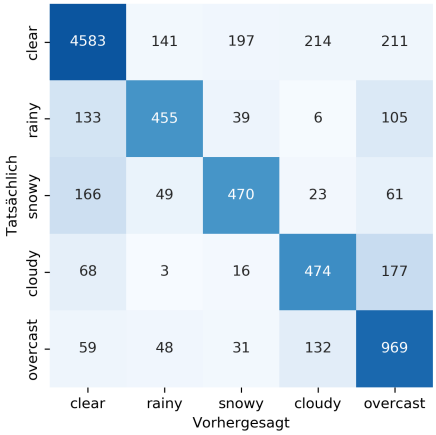

Abbildung 22: Konfusionsmatrix von Modell *(2)* auf BDD100K-Testdatensatz

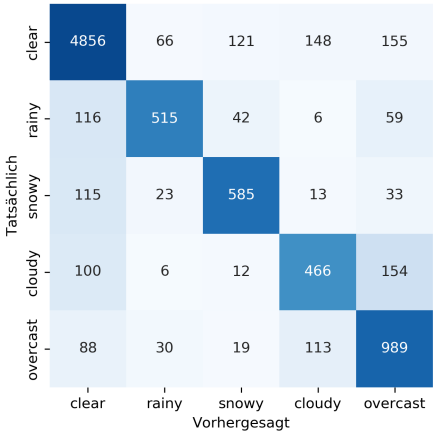

Abbildung 23: Konfusionsmatrix von Modell *(3)* auf BDD100K-Testdatensatz

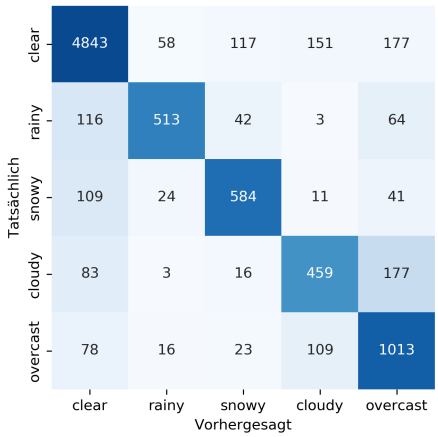

Abbildung 24: Konfusionsmatrix von Modell *(4)* auf BDD100K-Testdatensatz

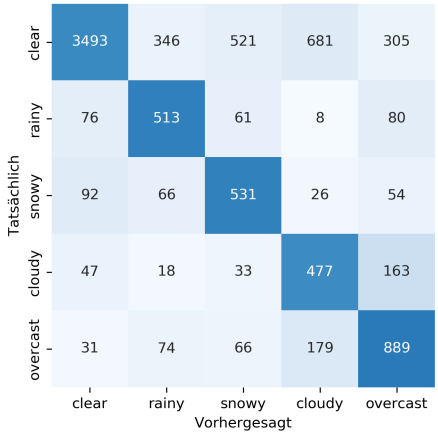

Abbildung 25: Konfusionsmatrix von Modell *(5)* auf BDD100K-Testdatensatz

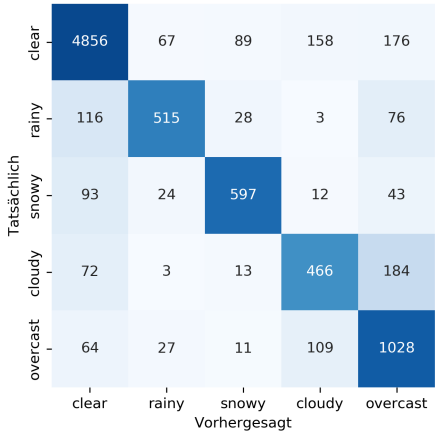

Abbildung 26: Konfusionsmatrix von Modell *(6)* auf BDD100K-Testdatensatz

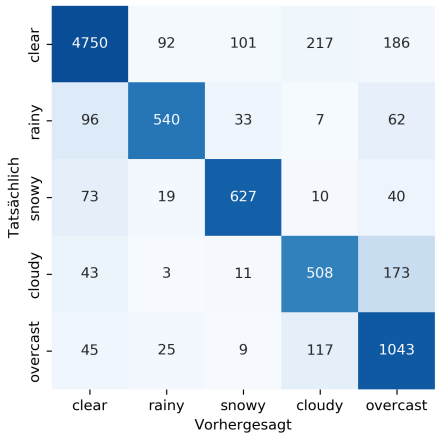

Abbildung 27: Konfusionsmatrix von Modell *(7)* auf BDD100K-Testdatensatz

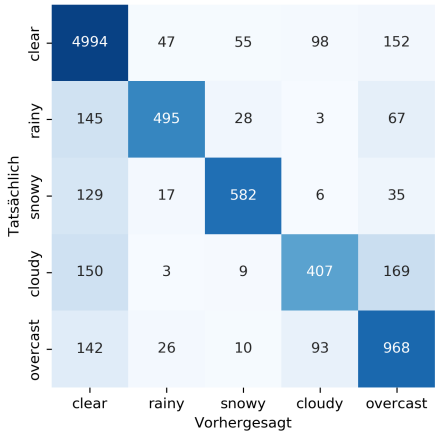

Abbildung 28: Konfusionsmatrix von Modell *(8)* auf BDD100K-Testdatensatz

A.5.2 Custom-Datensatz

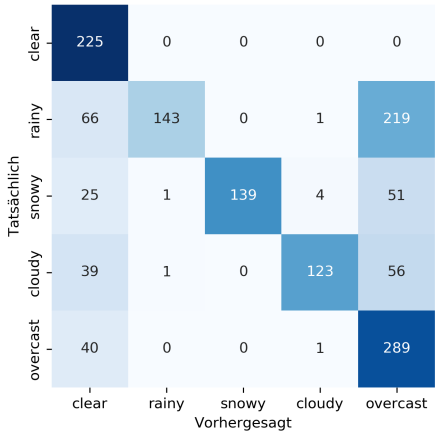

Abbildung 29: Konfusionsmatrix von Modell *(1)* auf dem Custom-Dataset durch Bildung des arithmetischen Mittels der Front- und Rückkameravorschläge

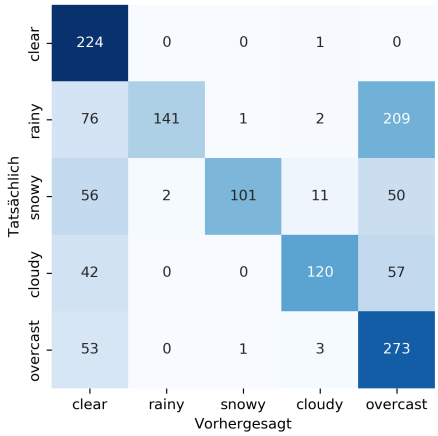

Abbildung 30: Konfusionsmatrix von Modell *(1)* auf den Frontkamerabildern des Custom-Datensatzes

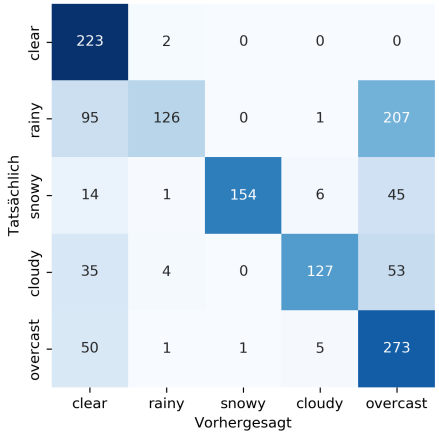

Abbildung 31: Konfusionsmatrix von Modell *(1)* auf den Rückkamerabildern des Custom-Datensatzes

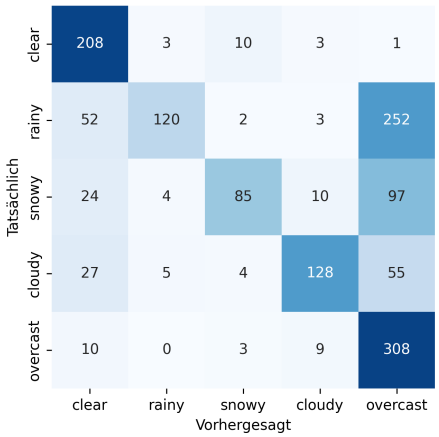

Abbildung 32: Konfusionsmatrix von Modell *(2)* auf dem Custom-Dataset durch Bildung des arithmetischen Mittels der Front- und Rückkameravorschläge

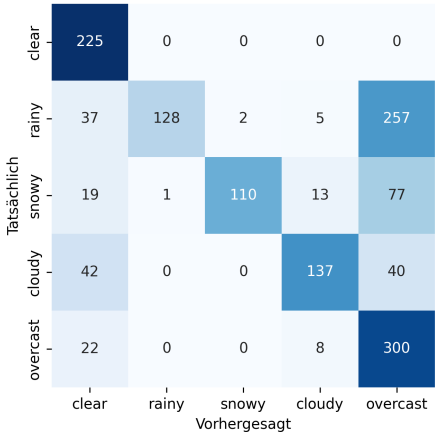

Abbildung 33: Konfusionsmatrix von Modell *(3)* auf dem Custom-Dataset durch Bildung des arithmetischen Mittels der Front- und Rückkameravorschläge

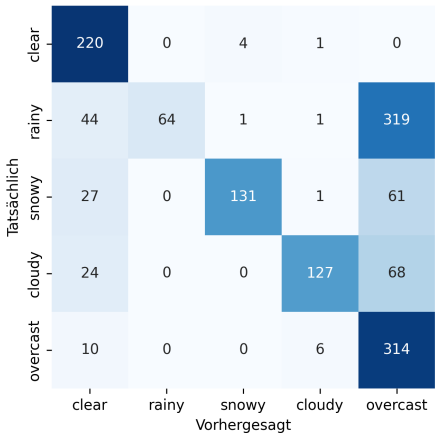

Abbildung 34: Konfusionsmatrix von Modell *(4)* auf dem Custom-Dataset durch Bildung des arithmetischen Mittels der Front- und Rückkameravorschläge

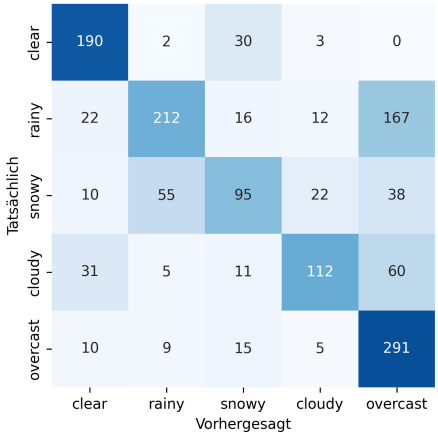

Abbildung 35: Konfusionsmatrix von Modell *(5)* auf dem Custom-Dataset durch Bildung des arithmetischen Mittels der Front- und Rückkameravorschläge

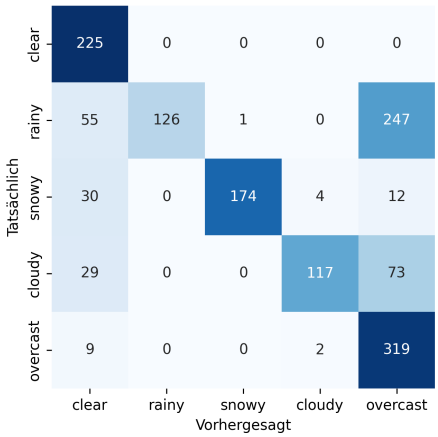

Abbildung 36: Konfusionsmatrix von Modell *(6)* auf dem Custom-Dataset durch Bildung des arithmetischen Mittels der Front- und Rückkameravorschläge

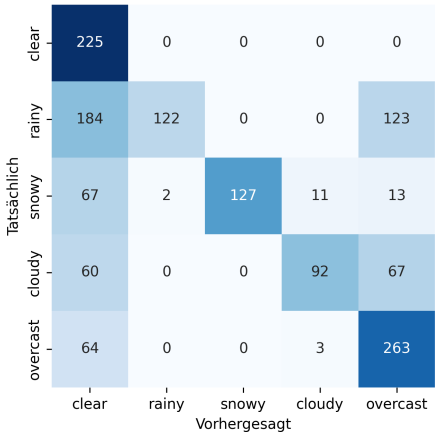

Abbildung 37: Konfusionsmatrix von Modell *(8)* auf dem Custom-Dataset durch Bildung des arithmetischen Mittels der Front- und Rückkameravorschläge

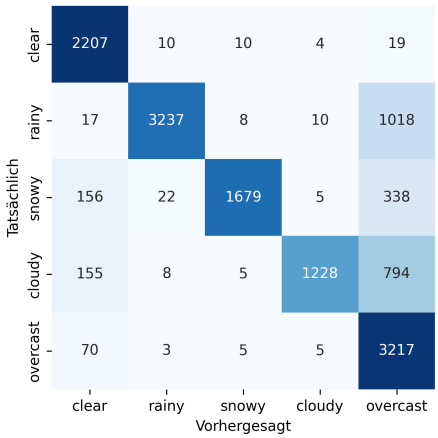

Abbildung 38: Konfusionsmatrix von Modell *(7)* auf dem Custom-Dataset durch Bildung des arithmetischen Mittels der Front- und Rückkameravorschläge und dem exponentiell gleitenden Durchschnitt