HAW HAMBURG

MASTERTHESIS

Kai König, Alexander Maßmann

Entwicklung eines Deep Learning-Verfahrens zur semantischen Segmentierung von Kameraaufnahmen für die Ermittlung des Schneebedeckungsgrades

FAKULTÄT TECHNIK UND INFORMATIK
Department Informations- und Elektrotechnik

Faculty of Computer Science and Engineering
Department of Information and Electrical Engineering

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Kai König, Alexander Maßmann

Entwicklung eines Deep Learning-Verfahrens zur semantischen Segmentierung von Kameraaufnahmen für die Ermittlung des Schneebedeckungsgrades

Masterarbeit eingereicht im Rahmen der Masterprüfung
im Studiengang *Master of Science Automatisierung*
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. -Ing. Jörg Dahlkemper
Zweitgutachter: Prof. Dr. -Ing. Florian Wenck

Eingereicht am: 19. April 2022

Kai König, Alexander Maßmann

Thema der Arbeit

Entwicklung eines Deep Learning-Verfahrens zur semantischen Segmentierung von Kameraaufnahmen für die Ermittlung des Schneebedeckungsgrades

Stichworte

Schneeerkennung, Wolkenerkennung, semantische Segmentierung, Bildverarbeitung, Klassifizierung, neuronale Netze, künstliche Intelligenz, Faltungsnetz, Deep Learning

Kurzzusammenfassung

Diese Arbeit beschäftigt sich mit der semantischen Segmentierung von Kameraaufnahmen zur anschließenden automatischen Berechnung des Schneebedeckungsgrades. Die Umsetzung erfolgt unter Zuhilfenahme tiefer neuronaler Netze. Dabei wird zum einen die standortunabhängige Auswertung von Schneeaufnahmen und zum anderen die Übertragbarkeit der Erkenntnisse auf die Wolkenerkennung ausgewertet.

Kai König, Alexander Maßmann

Title of Thesis

Development of a deep learning method for semantic segmentation of camera images for snow coverage determination

Keywords

snow detection, cloud detection, semantic segmentation, image processing, classification, neural networks, artificial intelligence, convolutional network, deep learning

Abstract

This work deals with the semantic segmentation of camera images for subsequent automatic calculation of snow coverage. The implementation is done with the help of deep neural networks. In the process, location-independent evaluation of snow images and the transferability of the findings to cloud detection are evaluated.

Danksagung

An dieser Stelle möchten wir uns bei allen bedanken, die uns bei der Erstellung dieser Arbeit direkt oder indirekt zur Seite standen. Zunächst möchten wir uns besonders bei unserem betreuenden Prüfer Prof. Dr. -Ing. Jörg Dahlkemper für die umfangreiche und kompetente Beratung sowie die Flexibilität bei den Besprechungen bedanken. Auch bei Prof. Dr. -Ing. Florian Wenck möchten wir uns für die Bereitschaft, das Amt des Zweitgutachters anzutreten, bedanken.
Ein großer Dank geht auch an Herrn Lenkeit, Herrn Zimmermann und Herrn Mergardt vom Deutschen Wetterdienst, die durch die Bereitstellung der Bilddaten sowie der Informationen zum Thema Wetterbeobachtung maßgeblich zum Erfolg dieser Arbeit beigetragen haben.
Darüber hinaus möchten wir den Mitarbeiter*innen des Creative Space for Technical Innovations für die Bereitstellung der IT-Infrastruktur und die technische Unterstützung danken.
Zuletzt möchten wir uns bei unseren Partnerinnen, Freund*innen und Familienmitgliedern für ihre Unterstützung bei der Korrektur und ihre Geduld in den letzten Monaten bedanken.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

Acc Accuracy.

ADSC Atrous Depthwise Separable Convolution.

ASPP Atrous Spatial Pyramid Pooling.

CAM Channel Attention Module.

CNN Convolutional Neural Network.

CSAM Channel and Spatial Attention Module.

DATL Domain Adversarial Transfer Learning.

DCNN Deep Convolutional Neural Network.

DWD Deutscher Wetterdienst.

EFCN Evidential Fully Convolutional Network.

FCL Fully Connected Layer.

FCN Fully Convolutional Network.

GAN Generative Adversarial Network.

GRL Gradient Reversal Layer.

HAW Hochschule für Angewandte Wissenschaften Hamburg.

IoU Intersection over Union.

IR Infrarot.

kNN k-Nearest Neighbour.

MAE Mean Absolute Error.

MFFM Multi-level/scale Feature Fusion Module.

mIoU mean Intersection over Union.

MLP Multi-Layer Perceptron.

MSE Mean Square Error.

ReLU Rectified Linear Unit.

SAM Spatial Attention Module.

SGD Stochastic Gradient Descent.

SVM Support Vector Machine.

1 Einleitung

Bildverarbeitung ist ein wichtiger Bestandteil moderner Industrie. Sie wird vielseitig angewendet, um Produktionsprozesse zu unterstützen und zu überwachen. Vor allem die Einführung künstlicher neuronaler Netze und die stetig wachsenden Rechenkapazitäten machen die Bildverarbeitungsalgorithmen zu leistungsstarken Partnern. Nach [Koeppe u. a., 2019] können in der Industrie „numerische Simulationsverfahren beschleunigt, dynamisches Verhalten vorhergesagt und Zustände von Strukturen überwacht werden". Seit einigen Jahren findet die Bildverarbeitung auch in anderen Bereichen Einzug. So ist sie mittlerweile einer der wichtigsten Bestandteile verschiedener Fahrassistenzsysteme oder wird in der Medizin zur Krebsdiagnostik verwendet (vgl. [Winkler u. a., 2019]). Ein weiteres Anwendungsgebiet der von künstlichen neuronalen Netzen gestützten Bildverarbeitung ist die Erkennung von Wetterereignissen. [Verma u. a., 2017] nutzen diese Art der Bildverarbeitung beispielsweise für Wetterprognosen zur Vorhersage der Verfügbarkeit von nachhaltigen Energien. Der Erkennung von Wetterereignissen widmet sich auch die vorliegende Arbeit. Nachfolgend werden Ziel, Motivation und Aufbau der Arbeit näher erläutert.

1.1 Ziel der Arbeit

Das Ziel dieser Arbeit teilt sich in zwei Bereiche. Das erste Ziel ist es, den Grad der Schneebedeckung von Landschaften aus Bildaufnahmen von beliebigen Bodenkameras unabhängig vom Aufnahmeort zu ermitteln. Dabei werden Aufnahmen verschiedener Tageszeiten, Standorte und Wetterlagen semantisch segmentiert. Darüber hinaus wird analysiert, ob sich der gefundene Ansatz auf ähnlich strukturierte Aufgaben übertragen lässt. In der Bildverarbeitung sind Schneeflächen und Wolken oft schwer voneinander zu unterscheiden (vgl. [Hongcai u. a., 2019]). Dies legt nahe, die Erkenntnisse und Konzepte dieser Arbeit auch auf dieses Gebiet zu transferieren. Folglich wird die Bestimmung des Wolkenbedeckungsgrades im Zuge dieser Arbeit ausgehend von ortsunabhängigen

Bodenkameraaufnahmen des oberen Halbraums ebenfalls untersucht. Um die semantische Segmentierung mittels Deep Learning zu ermöglichen, sind außerdem geeignete Trainingsdaten für die Schnee- und Wolkenerkennung in ausreichender Menge und Qualität zu erstellen.

1.2 Motivation der Arbeit

Die Erfassung von Wetterdaten ist ein Thema, welches bereits mit verschiedenen Ansätzen verfolgt wird. Dafür werden nach [Marquardt u. Reigber, 2002] beispielsweise unterschiedliche Fernerkundungssysteme sowie Satelliten zur Erfassung von Wetterdaten verwendet. Viele Parameter im Bereich der Wetterbeobachtung werden manuell erfasst und sollen zunehmend automatisiert werden. Die Messung der Schneebedeckung stellt dabei eine wesentlich größere Herausforderung dar als beispielsweise die Temperaturmessung. Durch aktuelle wissenschaftliche Probleme wie den Klimawandel gewinnen Parameter wie die Schneebedeckung zunehmend an Relevanz. So sagen auch [Lozán u. a., 2015] zu Schneebedeckung im Bezug auf den Klimawandel: „Veränderungen von Schneeflächen sind daher wichtige Vorgänge für den Strahlungshaushalt der Atmosphäre und beeinflussen durch zahlreiche Rückkopplung die Temperatur [...]“. Somit ist die Motivation dieser Arbeit, diese Daten zugänglicher zu machen, damit sie beispielsweise in der Klimaforschung effektiver genutzt werden können. Die Arbeit von [Perkovic, 2022] behandelt ebenfalls die kamerabasierte Ermittlung des Schneebedeckungsgrades. Dort werden Convolutional Neural Networks (CNNs) zur Klassifikation ohne semantische Segmentierung verwendet. Die Genauigkeit der von [Perkovic, 2022] getroffenen Vorhersagen sinkt deutlich bei der Klassifikation von Bildern unbekannter Standorte. Dieser Aspekt soll durch die Methoden dieser Arbeit relativiert werden, um die standortunabhängige Nutzung des Systems gewährleisten zu können.
Mit der automatisierten Ermittlung des Wolkenbedeckungsgrades wird ein weiterer Parameter zugänglich gemacht. Durch Ceilometer[1] lässt sich zwar bereits punktuell die Wolkenhöhe und das Vorhandensein von Wolken erfassen, eine ganzheitliche Betrachtung des aktuellen Wolkenbedeckungsgrades ist mit ihnen jedoch nicht möglich.

[1] Das Ceilometer ist ein Gerät zur automatischen Messung der Wolkenhöhe bzw. der Wolkenuntergrenze.

1.3 Aufbau der Arbeit

In dieser Arbeit werden zunächst jene relevanten Grundlagen aufgearbeitet, die für das Verständnis dieser Arbeit und der angewandten Verfahren notwendig sind. Dabei geht es um das Verfahren der semantischen Segmentierung im Rahmen der Bildverarbeitung und um die Theorie zu künstlichen neuronalen Netzen. Darauffolgend wird der Stand der Technik zur Nutzung von künstlicher Intelligenz für semantische Segmentierung und speziell für die Erkennung von Wetterereignissen erläutert. Im dritten Teil der Arbeit wird eine detaillierte Anforderungsanalyse erstellt, die sowohl Anforderungen an Funktionen und Genauigkeit des zu erstellenden Systems als auch an Schnittstellen für Endnutzer*innen enthält. Das Kapitel zur Konzeptphase enthält Konzepte sowohl für die Erstellung und Aufbereitung von Trainingsdaten für künstliche neuronale Netze als auch für die semantische Segmentierung im Rahmen der Schnee- und Wolkenerkennung. Darüber hinaus werden Konzepte für die Berechnung der Bedeckungsgrade, den Domain Transfer und die Auswertung der Ergebnisse geliefert. In den anschließenden Kapiteln zur Entwicklung der technischen Lösungen und der Ergebnisvalidierung werden jene zuvor ausgearbeiteten Konzepte umgesetzt und ausgewertet.
Schließlich werden in einem Fazit mit Ausblick die wichtigsten Erkenntnisse zusammengefasst und Anregungen für zukünftige Erweiterungen und Verbesserungen für die hier erarbeiteten technischen Lösungen gegeben.

2 Grundlagen

Dieses Kapitel behandelt die für die vorliegende Arbeit relevanten Grundlagen. Dabei werden zunächst die verschiedenen Formen maschinellen Lernens sowie künstliche neuronale Netze thematisiert. Darüber hinaus wird das Verfahren der semantischen Segmentierung von Bildern ohne und mit künstlichen neuronalen Netzen thematisiert.

2.1 Maschinelles Lernen

Künstliche neuronale Netze und Deep Learning werden für die Erfüllung von Aufgaben des maschinellen Lernens genutzt. Nach [Géron, 2020] wird grundsätzlich zwischen vier Arten des maschinellen Lernens unterschieden:

1) Überwachtes Lernen (supervised Learning)
2) Reinforcement Learning
3) Unüberwachtes Lernen (unsupervised Learning)
4) Halbüberwachtes Lernen (semi-supervised Learning)

Diese Arten des maschinellen Lernens werden nachfolgend am Beispiel künstlicher neuronaler Netze erläutert.

2.1.1 Überwachtes Lernen (supervised Learning)

Beim überwachten Lernen wird dem künstlichen neuronalen Netz während des Trainings zu jedem Eingabemuster ein vollständig spezifiziertes Ausgabemuster bereitgestellt (vgl. [Géron, 2020]). Nach [Paaß u. Hecker, 2020] kann das Netz dadurch so beeinflusst werden, dass es in der Lage ist, die Eingabemuster bei erneuter Vorlage selber zu erkennen oder

das Erlernte sogar auf ähnliche Muster zu übertragen. Die Erkennung ähnlicher Muster wird Generalisierung genannt (vgl. [Aggarwal, 2018]).

2.1.2 Reinforcement Learning

Beim Reinforcement Learning wird dem neuronalen Netz kein vollständig spezifiziertes Ausgabemuster während des Trainings bereitgestellt (vgl. [Aggarwal, 2018]). Das Netz erhält für seine Ausgaben lediglich eine Aussage darüber, ob das Ergebnis richtig ist oder nicht. Gegebenenfalls bekommt das Netz noch die Information über den Grad der Richtigkeit. Nach [Paaß u. Hecker, 2020] lernt das Netz demnach über Belohnungen beziehungsweise Bestrafungen.

2.1.3 Unüberwachtes Lernen (unsupervised Learning)

Das künstliche neuronale Netz erhält bei dem unüberwachten Lernen nach [Paaß u. Hecker, 2020] keine Rückmeldung über die Qualität seiner Ausgabe. Der Lernalgorithmus muss hier Ähnlichkeiten bei den verschiedenen Eingabemustern erkennen und diese in sinnvolle Gruppen sortieren. Diese erkannten Muster können dann wieder auf andere ähnliche Muster abgebildet werden. Letztlich werden hier aus Trainingsdaten statistische Eigenschaften extrahiert, die in neuen Eingabemustern wiedergefunden werden können.

2.1.4 Halbüberwachtes Lernen (semi-supervised Learning)

Das halbüberwachte Lernen stellt eine Mischung aus überwachtem und unüberwachtem Lernen dar. In diesem Fall werden für den überwachten Teil des Lernens den Trainingsdaten spezifizierte Ausgabemuster bereitgestellt, wodurch das neuronale Netz im unüberwachten Teil des Lernens bei der Einordnung von Mustern in die korrekten Gruppen unterstützt wird (vgl. [Géron, 2020]).

2.2 Künstliche neuronale Netze

Ein wesentliches Werkzeug dieser Arbeit sind künstliche neuronale Netze. Dieses Kapitel behandelt die relevanten Grundlagen zu deren Aufbau und Training. Schließlich werden die wichtigsten Funktionen von Faltungsnetzen erläutert.

2.2.1 Aufbau künstlicher neuronaler Netze

Eine Möglichkeit zur Umsetzung maschinellen Lernens sind künstliche neuronale Netze. Sie haben ihren Ursprung in den Neunzehnhundertfünfziger Jahren mit der Ausarbeitung von [Rosenblatt, 1958] zum Perzeptron. Ein künstliches neuronales Netz kann aus einer oder mehreren Schichten bestehen. Nachfolgend wird zunächst das einfachste einschichtige Netz, das sogenannte Perzeptron, mit einer Eingangsschicht und einem Ausgang erläutert. Darüber hinaus werden Aktivierungsfunktionen sowie mehrschichtige künstliche neuronale Netze thematisiert.

Perzeptron

Das Perzeptron aus Abbildung 2.1 ist ein künstliches neuronales Netz, welches aus einer Zelle besteht, die auch als Neuron bezeichnet wird. Die n-Eingänge $\mathbf{x} = (x_1, \ldots, x_n)$ des Neurons werden über die Gewichte $\mathbf{w} = (w_1, \ldots, w_n)$ mit dem Neuron verbunden. Das Ausgangssignal $y \, \epsilon \, \{+1, -1\}$ wird auf Basis der gewichteten Summe des Eingangs durch die Signum-Aktivierungsfunktion gebildet. Ein solcher Übergang wird als Schicht bezeichnet. Eine Schicht kann ein oder mehrere Neuronen enthalten. Mathematisch lässt sich das Perzeptron folgendermaßen beschreiben (vgl. [Aggarwal, 2018]):

$$y = sign \left\{ \sum_{i=1}^{n} w_i x_i \right\} \tag{2.1}$$

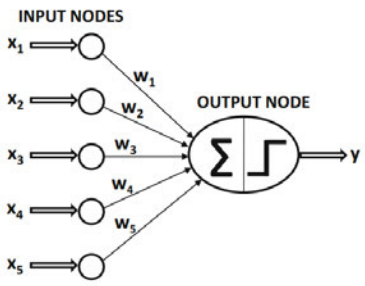


Abbildung 2.1: Aufbau eines Perzeptrons (vgl. [Aggarwal, 2018])

Auf diese Weise werden die Eingangssignale binär klassifiziert. Das Training eines neuronalen Netzes beeinflusst die Gewichte w_i. Neben der Signum-Aktivierungsfunktion existieren zahlreiche weitere Aktivierungsfunktionen (Φ), die nachfolgend erläutert werden.

Aktivierungsfunktionen

Eine Aktivierungsfunktion Φ bildet nach [Aggarwal, 2018], wie in Abbildung 2.2 dargestellt, den Ausgang der Neuronen – den sogenannten ‚Post-Activation Value' $h = \Phi(a_h)$. Der Faktor a_h stellt den inneren Funktionswert (‚Pre-Activation Value') $a_h = \mathbf{W} \cdot \mathbf{X}$ des Neurons vor der Aktivierung dar.
Dabei spielt der Gradient der Aktivierungsfunktion eine entscheidende Rolle, welcher für die Anpassung der Gewichte genutzt wird. Der Gradient wird über das Backpropagation-Verfahren, welches im weiteren Verlauf erläutert wird, errechnet.

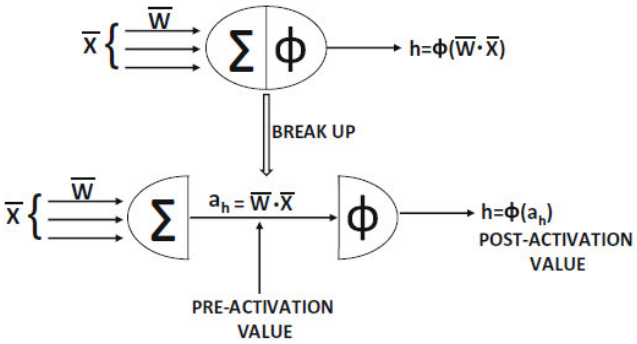


Abbildung 2.2: Schematischer Aufbruch eines Neurons (vgl. [Aggarwal, 2018])

Die Abbildungen 2.3 und 2.4 zeigen verbreitete Beispiele für Aktivierungsfunktionen und deren Gradienten. Die Wahl einer geeigneten Aktivierungsfunktion ist letztlich problemabhängig.

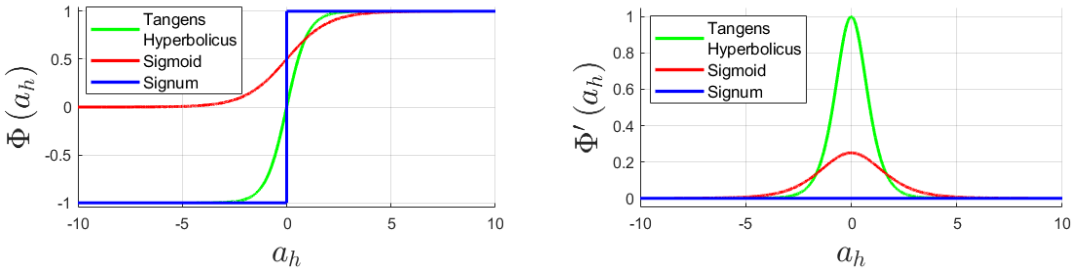


Abbildung 2.3: Verlauf von Aktivierungsfunktionen (links) und deren Gradienten (rechts)

Die Nutzung bestimmter Aktivierungsfunktionen kann durch ihre Gradienten beim Training neuronaler Netze auch Probleme mit sich bringen. So haben die Sigmoidfunktion

und der Tangens Hyperbolicus – für große positive und negative Eingangswerte – Gradienten, die gegen Null gehen.
Eine weitere verbreitete Aktivierungsfunktion ist die Rectified Linear Unit (ReLU). Abgeleitet aus Abbildung 2.4 ergibt sich Gleichung 2.2 für die ReLU-Aktivierungsfunktion und deren Gradienten (vgl. [Brandenbusch, 2018]).

$$\begin{aligned} \Phi_{ReLU}(a_h) &= max\{0, a_h\} \\ \Phi'_{ReLU}(a_h) &= \begin{cases} 1 & a_h > 0 \\ 0 & sonst \end{cases} \end{aligned} \tag{2.2}$$

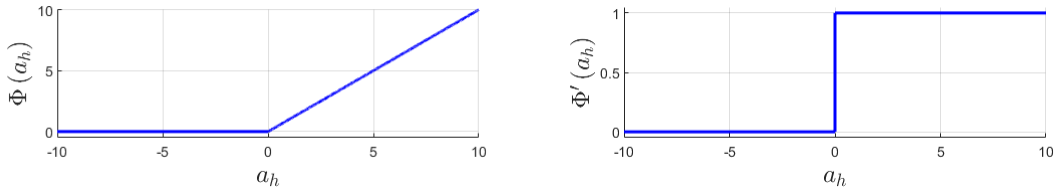


Abbildung 2.4: Verlauf der Rectified Linear Unit-Aktivierungsfunktion (links) und deren Gradient (rechts)

Multi-Layer Perceptron

Künstliche neuronale Netze können auch aus mehreren Schichten bestehen. Ein Beispiel für ein solches mehrschichtiges Netz ist das Multi-Layer Perceptron (MLP). Abbildung 2.5 zeigt ein einfaches MLP mit fünf Eingabeneuronen, zwei Hidden-Layer mit jeweils drei Neuronen und einem Ausgabeneuron. Als Hidden-Layer werden die Schichten des Netzes bezeichnet, die weder Input-, noch Output-Layer sind. Zusätzlich zu den Neuronen der Layer des MLP können sogenannte Bias-Neuronen genutzt werden, um die Schwellwerte der Eingangssignale der Neuronen zu beeinflussen (vgl. [Aggarwal, 2018]). Der Bias hat den Wert +1 und ist mit jeweils einem trainierbaren Gewicht an die Neuronen der zugehörigen Schicht angebunden. Bei sehr geringen Eingangssignalen der vorangehenden Schicht kann so eine anhaltende Aktivierung oder Deaktivierung des Neurons sichergestellt werden.
Das MLP nutzt eine feed forward Architektur, bei der alle Neuronen eines Layers mit allen Neuronen des darauffolgenden Layers verbunden sind. Diese Layer werden auch

als Fully Connected Layer (FCL) oder Dense Layer bezeichnet. Informationen werden in diesem Netz nur in eine Richtung übertragen. Bei künstlichen neuronalen Netzen zur Klassifikation entspricht die Anzahl der Ausgabeneuronen in der Regel der Anzahl der zu prädizierenden Klassen. Die Tiefe des Netzes k wird über die Anzahl der Layer definiert, wobei der Input-Layer in der Regel nicht mitgezählt wird, da dort keine Datenverarbeitung stattfindet (vgl. [Aggarwal, 2018]).

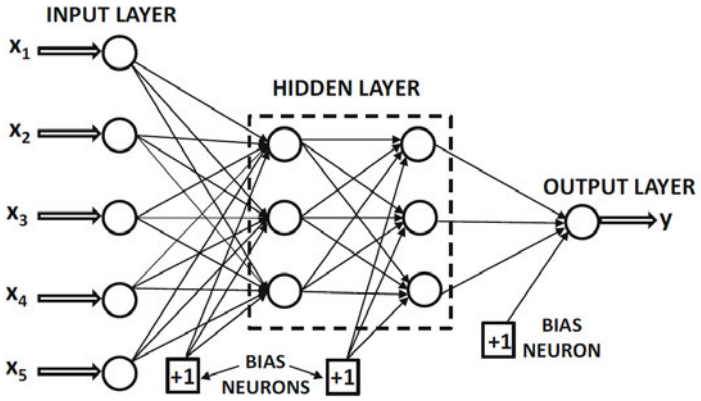


Abbildung 2.5: Multi-Layer Perceptron mit zwei versteckten Schichten (vgl. [Aggarwal, 2018])

Die Verbindungen zwischen den Layern des MLP können auch als Gewichtsmatrizen $\mathbf{W}^{(l)}$ dargestellt werden (siehe Abb. 2.6), wobei das hochgestellte l den jeweiligen Layer angibt. Die Ausgaben $h^{(l)}$ der einzelnen Layer werden nach [Aggarwal, 2018] wie in Gleichung 2.3 bis 2.5 berechnet.

Input- zu Hidden-Layer[2]

$$\mathbf{h}^{(1)} = \Phi\left(\mathbf{W}^{(0)} \cdot \mathbf{x}\right) \tag{2.3}$$

Hidden- zu Hidden-Layer[2]

$$\mathbf{h}^{(l+1)} = \Phi\left(\mathbf{W}^{(l)} \cdot \mathbf{h}^{(l)}\right) \quad \forall l \in \{1, \ldots, k-1\} \tag{2.4}$$

Hidden- zu Output-Layer[2]

$$\mathbf{y} = \Phi\left(\mathbf{W}^{(k)} \cdot \mathbf{h}^{(k)}\right) \tag{2.5}$$

[2]Die Notation wird für eine einheitliche Schreibweise abweichend zu [Aggarwal, 2018] dargestellt.

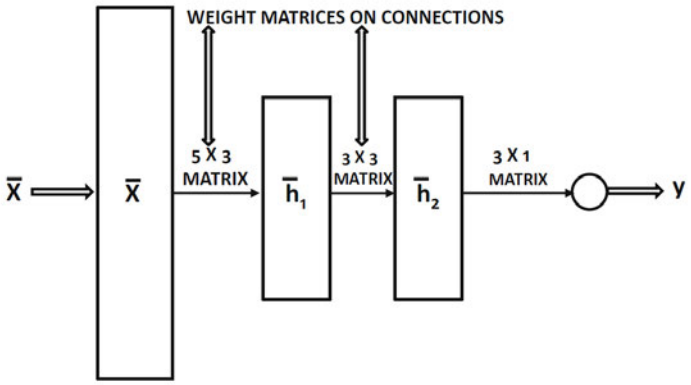


Abbildung 2.6: Multi-Layer Perceptron in Vektor-Darstellung (vgl. [Aggarwal, 2018])

Ein Spezialfall des MLP ist das shared MLP. Dabei teilen sich mehrere Eingangssignale die Gewichte und Layer des MLP. Abbildung 2.7 zeigt ein abstrahiertes shared MLP mit zwei unabhängigen Eingangssignalen. Folglich wird das MLP so trainiert, dass es allen Eingangssignalen genügt.

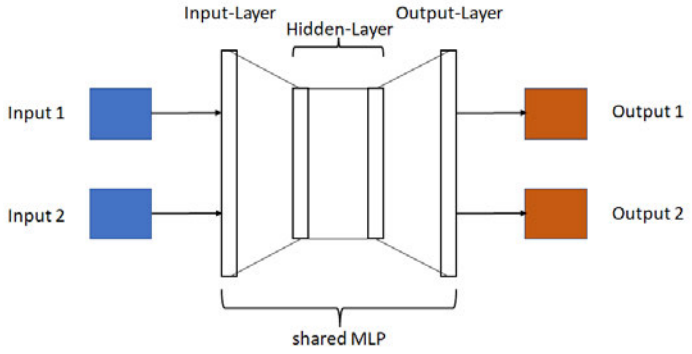


Abbildung 2.7: Shared Multi-Layer Perceptron mit zwei Eingangssignalen

2.2.2 Training künstlicher neuronaler Netze

Die Aufgabe des Trainings künstlicher neuronaler Netze ist, die Gewichte des neuronalen Netzes mit Hilfe der annotierten Trainingsdaten, welche auch als Ground Truth bezeichnet werden und während des Trainings als Referenz dienen, für die vorgesehene Aufgabe anzupassen (vgl. [Traeger u. a., 2003]). In diesem Kapitel werden die verschiedenen Phasen, die notwendigen Funktionen und die Probleme bei dem Training künstlicher neuronaler Netze erläutert.

Allgemeiner Trainingsablauf

Nach [Aggarwal, 2018] ist das Training in eine Vorwärts- und eine Rückwärtsphase aufgeteilt. In der Vorwärtsphase werden Trainingspaare, bestehend aus den Originaldaten und den annotierten Soll-Ausgangsdaten, in das Netz gespeist. Das Netz erzeugt basierend auf den aktuellen Gewichten eine Vorhersage für die Originaldaten, welche mit den Ausgangsdaten der Trainingspaare verglichen werden, indem der resultierende Fehler mit einer Loss-Function berechnet wird.
In der Rückwärtsphase werden die Gewichte des neuronalen Netzes mit Hilfe der Gradienten der Loss-Function aus der Vorwärtsphase trainiert. Im Optimalfall ist das Netz in der nächsten Vorwärtsphase genauer bei der Berechnung einer Vorhersage.

Loss-Function

Die Loss-Function bildet die Berechnungsgrundlage für die Anpassung der Gewichte des neuronalen Netzes während des Trainings. Diese ist allgemein definiert als $\varepsilon = L(\mathbf{y}, \hat{\mathbf{y}})$, wobei $\mathbf{y} = (y_1, \ldots, y_n)$ die annotierten Trainingsdaten und $\hat{\mathbf{y}} = (\hat{y}_1, \ldots, \hat{y}_n)$ die vom neuronalen Netz berechneten Vorhersagen darstellen. Als Loss-Function kann beispielsweise der Mean Square Error (MSE) nach Gleichung 2.6 genutzt werden. Der Faktor n stellt die Anzahl der möglichen Ausgangsklassen beziehungsweise Ausgabeneuronen dar. (vgl. [Aggarwal, 2018]).

$$L(\mathbf{y}, \hat{\mathbf{y}}) = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n} \tag{2.6}$$

Ein anderes verbreitetes Verfahren ist die Kreuzkorrelation. Diese wird nach [Brandenbusch, 2018] verwendet, wenn für die Trainingsdaten bei n Klassen gilt, dass $y_i = 1$ wenn die Klasse korrekt und $y_i = 0$ wenn die Klasse nicht korrekt ist. Die Kreuzkorrelation berechnet sich dann wie folgt (vgl. [Jadon, 2020]):

$$L(\mathbf{y}, \hat{\mathbf{y}}) = -\left[\sum_{i=1}^{n} y_i \cdot log(\hat{y}_i) + (1 - y_i) \cdot log(1 - \hat{y}_i)\right] \tag{2.7}$$

Welche Loss-Function letztlich für die Berechnung des Fehlers genutzt wird, ist problemabhängig. Nach [Jadon, 2020] gibt es viele mögliche Loss-Functions, die beispielsweise für die semantische Segmentierung von Bildern genutzt werden können.

Optimizer

Für die Optimierung der Gewichte des neuronalen Netzes erfolgt ein Gradientenabstieg in Richtung des negativen Gradienten der genutzten Loss-Function (vgl. [Aggarwal, 2018]). Ein weiterer wichtiger Parameter an dieser Stelle ist die Learning-Rate α. Über diesen Parameter wird reguliert, wie stark die Gewichte des Netzes bei einem Trainingsdurchlauf beeinflusst werden. Die Gewichte $w_{ij}^{(l)}$ berechnen sich über die Optimierungsfunktion aus Gleichung 2.8.

$$w_{ij}^{(l)} \leftarrow w_{ij}^{(l)} - \alpha \frac{\partial \epsilon}{\partial w_{ij}^{(l)}} \tag{2.8}$$

Um den Rechen- und Speicheraufwand bei der Aktualisierung der Gewichte zu minimieren, kann der Stochastic Gradient Descent (SGD) verwendet werden. Bei diesem wird der Fehler für einen gewissen Teil der Trainingsdaten, welcher Mini-Batch genannt wird, berechnet und für die Aktualisierung der Gewichte genutzt (vgl. [Goodfellow u. a., 2016]). Größere Mini-Batches erlauben nach [Goodfellow u. a., 2016] eine genauere Schätzung der Gradienten, benötigen aber auch mehr Speicher während des Trainings, wohingegen eine kleine Batchsize die Generalisierungsleistung des Netzes erhöhen kann. Nach [Goodfellow u. a., 2016] muss beim Training mit kleinen Batchsizes gegebenenfalls die Learning-Rate verringert werden, da die Optimierung bei großer Varianz zwischen den Trainingsdaten und einer großen Learning-Rate instabil werden kann. Gleichung 2.9 zeigt die Berechnung der Aktualisierung der Gewichte mit $\mathcal{B}$ als Mini-Batch.

$$\mathbf{W}^{(l)} \leftarrow \mathbf{W}^{(l)} + \mathbf{V}^{(l)} \quad , mit \quad \mathbf{V}^{(l)} \leftarrow -\alpha \sum_{i \in \mathcal{B}} \frac{\partial \epsilon_i}{\partial \mathbf{W}^{(l)}} \tag{2.9}$$

Weiterhin kann das Momentum verwendet werden, um das Training zu beschleunigen. Dabei wird Gleichung 2.9 zur Anpassung der Gewichte um das Momentum $\beta \in (0, 1)$ erweitert (vgl. [Aggarwal, 2018]).

$$\mathbf{W}^{(l)} \leftarrow \mathbf{W}^{(l)} + \mathbf{V}^{(l)} \quad , mit \quad \mathbf{V}^{(l)} \leftarrow \beta \mathbf{V}^{(l)} - \alpha \sum_{i \in \mathcal{B}} \frac{\partial \epsilon_i}{\partial \mathbf{W}^{(l)}} \tag{2.10}$$

Auf diese Weise wird der Gradient der vorherigen Iteration bei der Korrektur der Gewichte in der aktuellen Iteration berücksichtigt. Somit bleiben Betrag und Richtung des

Gradienten über mehrere Wiederholungen ähnlich (vgl. [Goodfellow u. a., 2016]).
Abbildung 2.8 zeigt den Einfluss des Momentums während des Gradientenabstiegs zur Korrektur der Gewichte. Dort ist erkennbar, dass der relative Gradient so beeinflusst wird, dass das Training beschleunigt und das Optimum schneller erreicht werden. Außerdem kann einem Festhängen in lokalen Optima während des Trainings durch das Momentum vorgebeugt werden (vgl. [Aggarwal, 2018]).

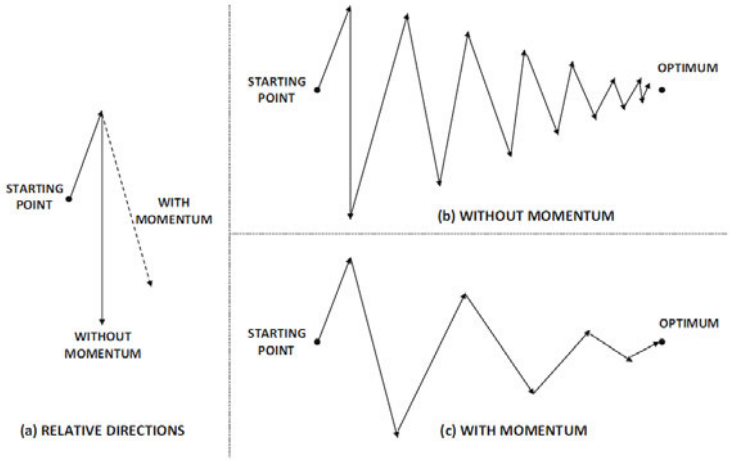


Abbildung 2.8: Gradientenabstieg mit und ohne Momentum (vgl. [Aggarwal, 2018])

Der Adam Optimizer, beschrieben durch [Kingma u. Ba, 2014], ist ein Beispiel für eine gradientenbasierte Optimierung für stochastische Zielfunktionen. Diese benötigt laut [Kingma u. Ba, 2014] lediglich Gradienten ersten Grades und hat daher einen geringen Speicherbedarf. Außerdem ist die Parameteraktualisierung invariant gegenüber der Skalierung des Gradienten. Nach [Kingma u. Ba, 2014] und [Aggarwal, 2018] wird die Parameteraktualisierung des Adam Optimizers wie nachfolgend dargestellt berechnet.
Die Parameter A_i sind die exponentiellen Mittelwerte der Gewichte w_i, welche mit der Decay-Rate $\rho \in (0, 1)$ aktualisiert werden.

$$A_i \leftarrow \rho A_i + (1 - \rho) \left(\frac{\partial L}{\partial w_i} \right)^2 \quad \forall i \tag{2.11}$$

Die Parameter F_i sind die exponentiell geglätteten Werte der Gradienten. Für die Glättung wird eine andere Decay-Rate $\rho_f \in (0, 1)$ genutzt.

$$F_i \leftarrow \rho_f F_i + (1 - \rho_f) \left(\frac{\partial L}{\partial w_i} \right) \quad \forall i \tag{2.12}$$

Nach [Aggarwal, 2018] stellt die Glättung des Gradienten mit ρ_f eine Variation der Momentum-Methode dar. Die Aktualisierung von w_i erfolgt mit der Learning-Rate α_t, welche wie in Gleichung 2.14 über die Decay-Rates ρ^t und ρ_f^t entlang der Iterationsschritte t angepasst wird:

$$w_i \leftarrow w_i - \alpha_t \frac{F_i}{\sqrt{A_i}} \quad \forall i \tag{2.13}$$

$$\alpha_t = \alpha \left(\frac{\sqrt{1-\rho^t}}{1-\rho_f^t} \right) \tag{2.14}$$

Die Decay-Rates ρ^t und ρ_f^t konvergieren zu 0 für große Werte von t, wodurch die Learning-Rate α_t zu α konvergiert. Eine Erweiterung des Adam Optimizers ist der Nadam Optimizer, bei dem nach [Dozat, 2016] der Algorithmus um das Nesterov-Momentum (vgl. [Aggarwal, 2018]) erweitert wird.

Backpropagation

Die Backpropagation wird zur Berechnung der Gradienten für den Gradientenabstieg verwendet (vgl. [Goodfellow u. a., 2016]). Über $\frac{\partial \varepsilon}{\partial w_{ij}^{(l)}}$ wird die Änderung des Fehlers bezogen auf die Änderung des jeweiligen Gewichts ermittelt. In einem neuronalen Netz werden die Gewichte entlang der Pfade angepasst. Anhand der Kettenregel können die Gradienten entlang des Pfades berechnet werden (vgl. [Aggarwal, 2018]). Gleichung 2.15 zeigt die allgemeine Berechnung von $\frac{\partial \varepsilon}{\partial w_{ij}^{(l)}}$ entlang eines Pfades.

$$\frac{\partial \epsilon}{\partial w_{ij}^{(l)}} = \frac{\partial \epsilon}{\partial y} \cdot \left[\frac{\partial y}{\partial h^{(k)}} \prod_{i=l}^{k-1} \frac{\partial h^{(i+1)}}{\partial h^{(i)}} \right] \cdot \frac{\partial h^{(l)}}{\partial w_{ij}^{(l)}} \quad \forall l \in \{1 \dots k\} \tag{2.15}$$

Da ein neuronales Netz aber wie in Abbildung 2.5 in der Regel aus mehreren Pfaden besteht, muss die Gleichung 2.15 um die Aufsummierung der Pfade $\mathcal{P}$ ergänzt werden (siehe Gl. 2.16).

$$\frac{\partial \epsilon}{\partial w_{ij}^{(l)}} = \frac{\partial \epsilon}{\partial y} \cdot \left[\sum_{[h^{(l)}, h^{(l+1)}, \dots, h^{(k)}, y] \in \mathcal{P}} \frac{\partial y}{\partial h^{(k)}} \prod_{i=l}^{k-1} \frac{\partial h^{(i+1)}}{\partial h^{(i)}} \right] \cdot \frac{\partial h^{(l)}}{\partial w_{ij}^{(l)}} \quad \forall l \in \{1 \dots k\} \tag{2.16}$$

Probleme beim Training von künstlichen neuronalen Netzen

Während des Trainings von künstlichen neuronalen Netzen kann eine Vielzahl von Problemen entstehen. In diesem Abschnitt werden drei wesentliche Problemfelder des Trainings betrachtet:

1) Overfitting / Underfitting
2) Vanishing / Exploding Gradient Problem
3) Lokale Optima und Scheinoptima

Overfitting / Underfitting sind häufige Probleme beim Training künstlicher neuronaler Netze, die verschiedene Auswirkungen und Ursachen haben. Overfitting führt dazu, dass das Netz Probleme bei der Generalisierung, also bei der Erkennung ähnlicher Muster anhand der trainierten Gewichte, hat. „The problem of overfitting refers to the fact that fitting a model to a particular training data set does not guarantee that it will provide good prediction performance on unseen test data, even if the model predicts the targets on the training data perfectly“ [Aggarwal, 2018]. Der Grund für ein solches Generalisierungsproblem ist in der Regel ein Mangel an Trainingsdaten bei einer zu großen Anzahl trainierbarer Parameter. Somit können hier bei der Prediction von bisher ungesehenen Trainingsdaten falsche Rückschlüsse bei der Erkennung von Mustern gezogen werden. Die Generalisierungsleistung nimmt also ab. Auf der anderen Seite gibt es ebenfalls ein Problem, wenn das künstliche neuronale Netz zwar viele Trainingsdaten erhält, jedoch zu wenig Parameter besitzt. In diesem Fall ist es nicht in der Lage, die Komplexität der Trainingsdaten zu erfassen und zu erlernen. Der Loss erreicht somit während des Trainings keinen ausreichend niedrigen Wert (vgl. [Goodfellow u. a., 2016]). Dieses Phänomen wird auch Underfitting genannt.
Um diese Probleme zu beseitigen, gibt es verschiedene Möglichkeiten. Um Underfitting zu vermeiden, kann die Komplexität des künstlichen neuronalen Netzes über die Anzahl der Parameter erhöht werden. Das Overfitting ist durch eine größere Menge Trainingsdaten oder unter Zuhilfenahme von Augmentations zu beseitigen. Dabei werden bestehende Trainingsdaten vervielfältigt und durch Operationen wie Rotation, Spiegelung oder anderer Manipulationen bearbeitet. Nach [Aggarwal, 2018] sollte die Anzahl der Trainingsdatenpunkte etwa zwei- bis dreimal größer sein, als die Anzahl der Parameter des künstlichen neuronalen Netzes. Es gibt jedoch auch andere Möglichkeiten, Overfitting zu verhindern:

Regularization ist ein Verfahren, welches durch eine Straffunktion $\lambda||\mathbf{W}||^p$ bei der Loss-Function die Anpassung der Gewichte $\mathbf{W}$ reguliert, wobei λ den Einfluss der Straffunktion reguliert. Aus mathematischer Sicht wird dadurch der Aktualisierung der Gewichte ein Wert proportional zu λw_i abgezogen. Der Gradientenabstieg zur Anpassung der Gewichte in einem Mini-Batch $\mathcal{B}$ kann in diesem Fall folgendermaßen aussehen (vgl. [Aggarwal, 2018]):

$$\mathbf{W} \leftarrow \mathbf{W}(1 - \alpha\lambda) + \alpha \sum_{\mathbf{X} \in \mathcal{B}} E(\mathbf{X}) \cdot \mathbf{X} \tag{2.17}$$

$E(\mathbf{X})$ stellt hier den Fehler und $\mathbf{X}$ die Eingangsdaten dar.
Nach [Goodfellow u. a., 2016] ist das Parameter Sharing ein weiteres Verfahren, welches der Regularization ähnelt. Dabei werden Parameter des neuronalen Netzes wie bei dem shared MLP geteilt. Bei CNN passiert das Parameter Sharing automatisch durch die Faltung, bei der ein Filter Kernel mit einer definierten Parameteranzahl auf den gesamten Eingangslayer angewendet wird (siehe Kap. 2.2.3).
Dropout ist ebenfalls ein Verfahren, welches Overfitting reduzieren kann. Dadurch erfolgt in jedem Trainingsschritt eine zufällige Abschaltung eines bestimmten Prozentsatzes von Neuronen in den, mit Dropout belegten, Layern. Das Ziel dieser Technik ist es, die Robustheit des Netzes zu erhöhen, indem das trainierte Wissen besser über das Netzwerk verteilt wird (vgl. [Srivastava u. a., 2014]).
Das Training der künstlichen neuronalen Netze erfolgt in der Regel in mehreren Epochen. Die Anzahl der Trainings-Epochen bestimmt, in wie vielen Iterationsschritten das Netz trainiert wird. Dabei wird in jeder Trainings-Epoche mit dem gesamten, in Mini-Batches aufgeteilten, Trainingsdatensatz trainiert. Eine zu große Anzahl an Trainings-Epochen bei zu wenig Trainingsdaten kann zum Overfitting des künstlichen neuronalen Netzes führen, wobei das Early Stopping genutzt werden kann, um diesem Effekt entgegenzuwirken (vgl. [Srivastava u. a., 2014]). Beim Early Stopping wird nach jeder Trainingsepoche anhand von Testdaten geprüft, ob der Fehler auf den Testdaten weiter abnimmt, wieder zunimmt oder stagniert. Das Early Stopping stoppt das Training des künstlichen neuronalen Netzes bei einer Stagnation oder Zunahme des Fehlers, um ein Overfitting zu verhindern (vgl. [Aggarwal, 2018]).

Vanishing / Exploding Gradient Problem sind Gradientenprobleme, die bei der Nutzung bestimmter Aktivierungsfunktionen auftreten. Das Vanishing Gradient Problem

tritt beispielsweise bei Nutzung der Sigmoidfunktion und Tangens Hyperbolicus Funktion für große positive bzw. negative Eingangswerte auf. Dieses Problem entsteht durch Gradienten < 1 oder die Sättigung des Gradienten und der daraus resultierenden verschwindenden Anpassung der Gewichte bei zunehmender Netztiefe. Zunehmende Netztiefe bedeutet eine Zunahme der Anzahl an Hidden-Layer im Netz. Auf der anderen Seite gibt es das Exploding Gradient Problem. Dieses tritt bei Gradienten > 1 auf. In diesem Fall erfolgt eine immer größer werdende Anpassung der Gewichte bei zunehmender Netztiefe (vgl. [Goodfellow u. a., 2016]). Je tiefer das Netz ist, desto größer wird – aufgrund der Multiplikation entlang der Gewichte bis zum Eingang des Netzes – der Einfluss des Gradienten.
Abhilfe schafft beispielsweise die ReLU-Aktivierungsfunktion. Diese Aktivierungsfunktion beugt den oben genannten Gradientenproblemen vor, da sich bei positiven Eingangswerten die Gradienten bei mehrfacher Multiplikation nicht verändern (vgl. [Aggarwal, 2018]). Wenn sichergestellt wird, dass die Eingangswerte dieser Aktivierungsfunktion stets > 0 sind, tritt hier nach Gleichung 2.2 keines der oben genannten Gradientenprobleme auf.
Eine weitere Möglichkeit, den Gradientenproblemen entgegenzuwirken, ist die Batch Normalization. Wie von [Aggarwal, 2018] beschrieben, können Normalization Layer eingesetzt werden, um die Ausgangssignale der Aktivierungsfunktionen zu normalisieren. Diese Layer enthalten trainierbare Parameter, welche den Mittelwert und die Standardabweichung der Ausgangssignale der Aktivierungsfunktionen beeinflussen. Auf diese Weise können sehr große beziehungsweise kleine Gradienten bei der Optimierung des neuronalen Netzes vermieden und so unausgeglichene Gewichtsverteilungen verhindert werden.

Lokale Optima und Scheinoptima können während der Optimierung des Netzes durch die Nichtlinearitäten in der Optimierungsfunktion zum Problem werden und das Training negativ beeinflussen (vgl. [Goodfellow u. a., 2016]). So kann es beispielsweise passieren, dass die Optimierungsfunktion ein Schein-Optimum findet, welches nur in den Trainingsdaten, jedoch nicht in den Testdaten existiert (vgl. [Aggarwal, 2018]). Eine Möglichkeit, diesem Problem vorzubeugen, ist ein Vortrainieren einzelner Layer des Netzes, um sinnvolle Initialwerte für das Training des gesamten Netzes zu erzeugen. Eine Konvergenz in einem lokalen Optimum ist darüber hinaus ebenfalls möglich. Abbildung 2.9 zeigt abstrahiert, wie die Optimierung des Netzes in einem lokalen Optimum konvergieren kann. Abhilfe schafft nach [Aggarwal, 2018] die Nutzung des Momentums.

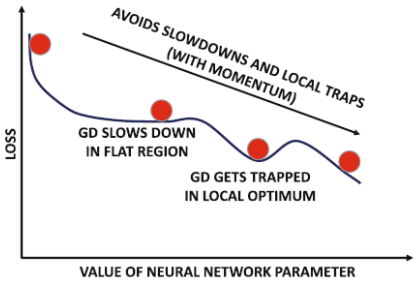


Abbildung 2.9: Probleme mit lokalen Optima während des Gradientenabstiegs (vgl. [Aggarwal, 2018])

2.2.3 Faltungsnetze (Convolutional Neural Network)

Das Convolutional Neural Network (CNN) stellt in Bereichen wie der Bild- oder Spracherkennung eine sehr wichtige Form künstlicher neuronaler Netze dar. Es erlaubt die Erkennung komplexer Strukturen in großen, strukturierten Datenmengen (vgl. [Arel u. a., 2010], [Fawaz u. a., 2018]). Im Beispiel der Bilderkennung werden den künstlichen neuronalen Netzen über Faltungsoperationen Merkmale (Features) der zu erkennenden Klassen antrainiert. Die dafür benötigten Trainingsdaten müssen je nach Lerntyp zuvor annotiert werden. Dies kann ein Problem bei der Nutzung von CNN darstellen, da es mitunter erhebliche Aufwände mit sich bringt (vgl. [Mullen u. a., 2019]). Bei komplexen Szenerien und hoher Klassenanzahl wird die Anzahl der benötigten Trainingsdaten zudem sehr groß (vgl. [Cho u. a., 2015]). Bei Verfahren wie der semantischen Segmentierung, welches optimalerweise eine pixelweise Klassifizierung der Trainingsdaten erfordert, steigt der Annotierungsaufwand zusätzlich gegenüber eines Klassifikators. Nachfolgend werden die wichtigsten Funktionen von CNN erläutert.

Faltung (Convolution)

Anders als beim MLP ist bei den CNN nicht jedes Neuron eines Layers mit jedem Neuron des vorherigen Layers verbunden. Die Verbindung erfolgt hier mittels einer Gewichtsmatrix (Filter Kernel), über die ein lokaler Teil des Gesamtkontextes repräsentiert wird (vgl. [Arel u. a., 2010]). Diese Filterung ermöglicht es dem CNN, auf Basis des Eingangsbildes mit den Dimensionen $A^{(l)} \times B^{(l)} \times D^{(l)}$ mehrdimensionale Merkmale zu extrahieren, um

eine geeignete Vorhersage am Netzausgang treffen zu können[3]. Die Filterung wird auf dem gesamten Eingangslayer durchgeführt, wobei die Höhe $A^{(l+1)}$ und die Breite $B^{(l+1)}$ des Ausgangslayers $\mathbf{H}^{(l+1)}$ der Breite und Höhe des Eingangsbildes $A^{(l)}$ und $B^{(l)}$ entsprechen, beziehungsweise je nach Parametrisierung der Filterung reduziert werden, was im weiteren Verlauf des Kapitels erläutert wird.
Mit der Anzahl der Kanäle (Tiefe) $D^{(l+1)}$ ergibt sich ein Ausgangslayer mit den Dimensionen $Dim\left(\mathbf{H}^{(l+1)}\right) = A^{(l+1)} \times B^{(l+1)} \times D^{(l+1)}$ durch die Filterung mit dem Filter Kernel $Dim\left(\mathbf{K}^{(l)}\right) = F_A^{(l)} \times F_B^{(l)} \times D^{(l)}$, welcher die Gewichte repräsentiert. Diese werden für die Erstellung eines vollständigen Kanals des darauffolgenden Layers verwendet. Pro Ausgangskanal wird ein Satz von Gewichten benötigt. $c = D^{(l+1)}$ spiegelt entsprechend die Anzahl der durchgeführten Filterungen wider (vgl. [Aggarwal, 2018]). Die Anzahl der Gewichte bei der Faltung, die für den Übergang von Schicht $l \rightarrow l+1$ benötigt werden, wird über $F_A^{(l)} \cdot F_B^{(l)} \cdot D^{(l)} \cdot D^{(l+1)}$ errechnet (siehe Abb. 2.10). Bei aktiviertem Bias kommt noch ein zusätzliches Gewicht pro Kanal hinzu.

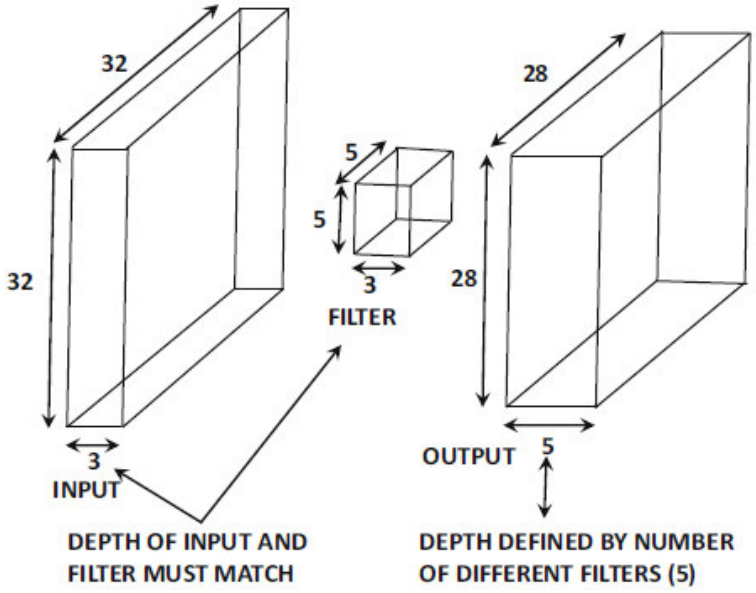


Abbildung 2.10: Faltung eines Eingangsbildes (vgl. [Aggarwal, 2018])

Diese Gewichte sind in einer Gewichtsmatrix $\mathbf{W}^{(c,l)} = \left[w_{rsk}^{(c,l)}\right]$ definiert, wobei r, s und k den Index des jeweiligen Gewichtes darstellen. Das aktuelle Objekt (oder auch

[3] $A^{(l)} \times B^{(l)}$ stellt die Pixelabmaße und $D^{(l)}$ die Anzahl der Farbkanäle des Eingangsbildes bei $l = 0$ dar.

Featuremap) wird durch $\mathbf{H}^{(l)} = \left[h_{ijc}^{(l)}\right]$ repräsentiert. Hierbei stellen i, j und c den jeweiligen Datenpunkt der Featuremap dar. Somit definiert sich der Übergang $l \to l+1$ wie folgt (vgl. [Aggarwal, 2018]):

$$h_{ijc}^{(l+1)} = \sum_{r=1}^{F_A^{(l)}} \sum_{s=1}^{F_B^{(l)}} \sum_{k=1}^{D^{(l)}} w_{rsk}^{(c,l)} h_{i+r-1,j+s-1,k}^{(l)} \quad \begin{aligned} &\forall i \in \left\{1, \ldots, A^{(l)} - F_A^{(l)} + 1\right\} \\ &\forall j \in \left\{1, \ldots, B^{(l)} - F_B^{(l)} + 1\right\} \\ &\forall c \in \left\{1, \ldots, D^{(l+1)}\right\} \end{aligned} \tag{2.18}$$

Der Filter Kernel $\mathbf{K}^{(l)}$ wird somit auf den gesamten Eingangslayer angewendet. Das rezeptive Feld eines jeden Datenpunktes, welches den lokalen Einfluss des Eingangs auf den Ausgang darstellt, hat die Größe $F_A^{(l)} \times F_B^{(l)} \times D^{(l)}$. Ein konkretes Beispiel ist in Abbildung 2.11 dargestellt.

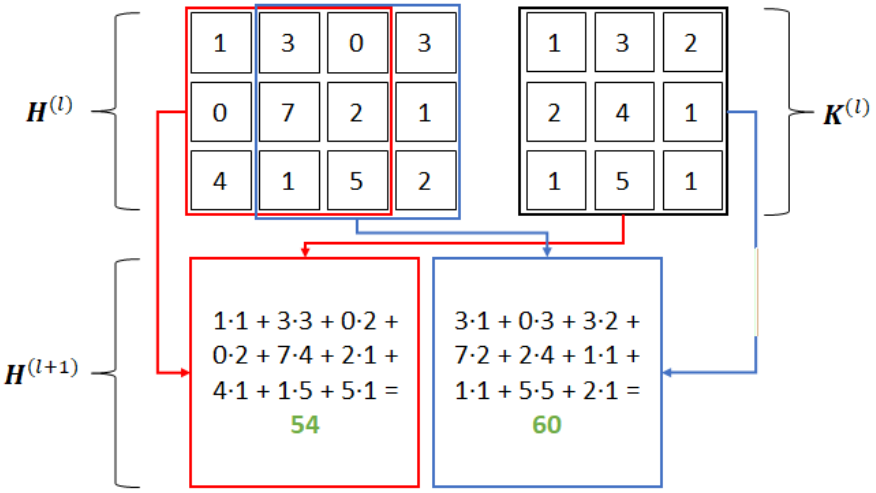


Abbildung 2.11: Übergang von $\mathbf{H}^{(l)} \to \mathbf{H}^{(l+1)}$ mittels Filter $\mathbf{K}^{(l)}$ mit $D^{(l)} = D^{(l+1)} = 1$

Wie Abbildung 2.11 zu entnehmen ist, ändert sich mit dem Übergang $\mathbf{H}^{(l)} \to \mathbf{H}^{(l+1)}$ die Dimension der Featuremap wie in Gleichung 2.19 allgemein beschrieben von 3×4 zu 1×2. Grund hierfür ist die Ausprägung des Filter Kernels $\mathbf{K}^{(l)}$, welcher lediglich in zwei verschiedenen Positionen der Featuremap anwendbar ist.

$$\begin{aligned} A^{(l+1)} &= A^{(l)} - F_A^{(l)} + 1 \\ B^{(l+1)} &= B^{(l)} - F_B^{(l)} + 1 \end{aligned} \tag{2.19}$$

Diese Dimensionsänderung muss berücksichtigt oder durch Padding unterdrückt werden (vgl. [Aggarwal, 2018]). Wie in Abbildung 2.12 verdeutlicht, wird bei Verwendung eines quadratischen Filter Kernels die Featuremap vor der Faltung um einen Rand mit der Breite $p = (F^{(l)} - 1) \cdot 1/2$ erweitert. Dies ermöglicht die Filterung im Randbereich ohne Beeinflussung der innenliegenden Werte (hier die Werte 54 und 60). Weiterhin wird die Dimension der Featuremap (hier: $3 \times 4 \times 1$) von $l \rightarrow l + 1$ nicht verändert.

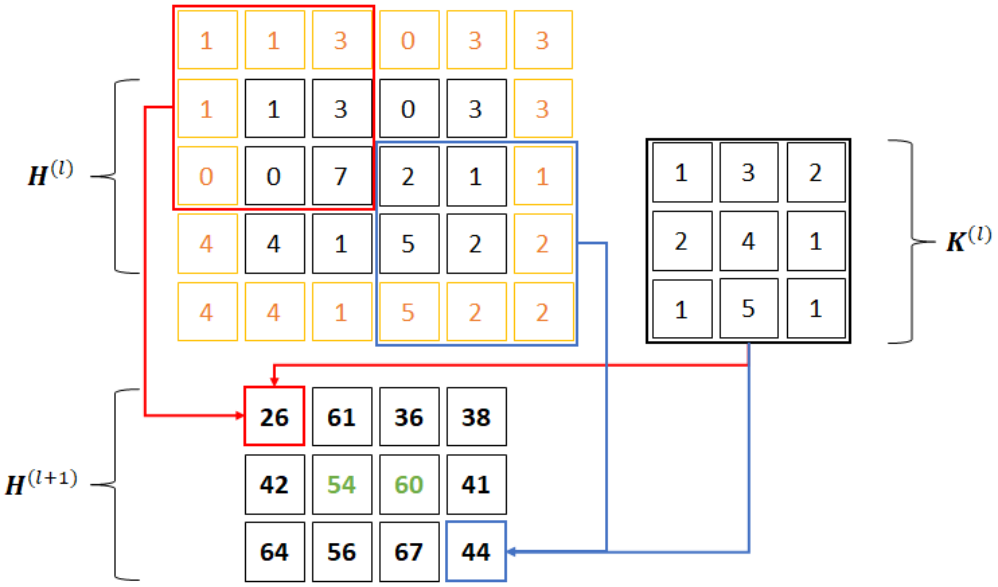


Abbildung 2.12: Übergang von $\mathbf{H}^{(l)} \rightarrow \mathbf{H}^{(l+1)}$ mittels Filter $\mathbf{K}^{(l)}$ mit $D^{(l)} = D^{(l+1)} = 1$ und Padding

Gleichung 2.19 kann entsprechend der Padding-Information erweitert werden, sodass gilt:

$$\begin{aligned} A^{(l+1)} &= A^{(l)} - F_A^{(l)} + 2 \cdot p + 1 \\ B^{(l+1)} &= B^{(l)} - F_B^{(l)} + 2 \cdot p + 1 \end{aligned} \tag{2.20}$$

Die in Abbildung 2.12 verwendete Strategie für das Padding wird als ‚same' bezeichnet und füllt den Rand mit den außenliegenden Werten auf.
Anders als das Padding bietet der Stride die Möglichkeit, beim Übergang $\mathbf{H}^{(l)} \rightarrow \mathbf{H}^{(l+1)}$ die Dimension des Ausgangs $A^{(l+1)}$ und $B^{(l+1)}$ aktiv zu verringern (vgl. [Aggarwal, 2018]). Dies geschieht, indem der Filter Kernel $\mathbf{K}^{(l)}$ um einen Wert $s = (s_A, s_B)$ mit $(s_A, s_B) > 1$ verschoben wird. Die Erweiterung um den Stride ist in Gleichung 2.21 definiert.

$$\begin{aligned} A^{(l+1)} &= \frac{A^{(l)} - F_A^{(l)} + 2 \cdot p}{s_A} + 1 \\ B^{(l+1)} &= \frac{B^{(l)} - F_B^{(l)} + 2 \cdot p}{s_B} + 1 \end{aligned} \tag{2.21}$$

Bei der Aktivierung des Strides ändert sich das rezeptive Feld aufgrund des identischen Filter Kernels nicht. Durch die Reduzierung der Anzahl an Faltungsoperationen wird jedoch Rechen- und Speicherleistung eingespart.
Eine Möglichkeit, das rezeptive Feld ohne Vergrößerung des Filter Kernels zu erhöhen, bietet die Dilated Convolution (vgl. [Yu u. Koltun, 2016]). Hier wird die Dilation-Rate $d \geq 1$ eingeführt, welche den Filter Kernel spreizt. Das rezeptive Feld erhöht sich, wie in Abbildung 2.13 erkennbar, durch die Spreizung des Filter Kernels ($1 \mathrel{\hat{=}}$ keine Spreizung des Filter Kernels). Die gespreizten Dimensionen des Filter Kernels mit der selben Anzahl von Gewichten ergeben sich somit zu $F^{(l)} \leftarrow F^{(l)} + \left(F^{(l)} + 1\right) \cdot (d - 1)$.

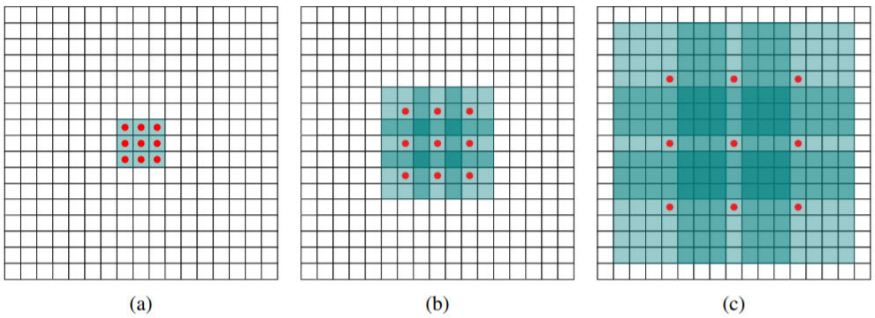


Abbildung 2.13: Rezeptives Feld der Dilated Convolution (a: d = 1, b: d = 2, c: d = 4) (vgl. [Yu u. Koltun, 2016])

Eine Erweiterung der Gleichung 2.21 ist wie folgt möglich:

$$
\begin{aligned}
A^{(l+1)} &= \frac{A^{(l)} - \left(F_A^{(l)} + \left(F_A^{(l)} + 1\right) \cdot (d-1)\right) + 2 \cdot p}{s_A} + 1 \\
B^{(l+1)} &= \frac{B^{(l)} - \left(F_B^{(l)} + \left(F_B^{(l)} + 1\right) \cdot (d-1)\right) + 2 \cdot p}{s_B} + 1
\end{aligned}
\tag{2.22}
$$

Abschließend ist zu erwähnen, dass durch die Verwendung eines Filter Kernels pro Kanal des Ausgangslayers die Anzahl der Parameter gegenüber vollständig verbundenen Layern verhältnismäßig gering ist. Dies ist durch die geringere Ausdehnung vom Filter Kernel gegenüber der Featuremap zu begründen. Des Weiteren ermöglicht dies die Repräsentation eines Merkmals pro Kanal, welches aus dem jeweils vorhergehenden Layer extrahiert wird. Die vorab beschriebenen Variationen der Faltung bieten die Möglichkeit, bei der Filterung die Bildfläche konstant zu halten (Padding), Rechen- und Speicherleistung zu reduzieren (Stride) und das rezeptive Feld ohne Erhöhung der Parameteranzahl zu vergrößern (Dilated Convolution).

Transposed Convolution

Eine weitere Form der Filterung ist die Transposed Convolution. Hier wird die Faltung aus Gleichung 2.18 in eine Matrixmultiplikation überführt. Zunächst wird der Eingangslayer $\mathbf{H}^{(l)}$ vektorisiert (auch Flatten genannt):

$$
vec\,(\mathbf{H}) = [h_{1,1},\ \ldots,\ h_{1,n},\ \ldots, h_{1,2},\ \ldots,\ h_{2,n},\ \ldots,\ h_{m,1},\ \ldots,\ h_{m,n}]^T
$$

Die Gewichte werden in einer Matrix $\mathbf{C}^{(l)}$ dargestellt (erläutert am Beispiel aus Abbildung 2.11):

$$
\mathbf{C}^{(l)} = \begin{pmatrix}
w_{0,0}^{(l)} & w_{0,1}^{(l)} & w_{0,2}^{(l)} & 0 & w_{1,0}^{(l)} & w_{1,1}^{(l)} & w_{1,2}^{(l)} & 0 & w_{2,0}^{(l)} & w_{2,1}^{(l)} & w_{2,2}^{(l)} & 0 \\
0 & w_{0,0}^{(l)} & w_{0,1}^{(l)} & w_{0,2}^{(l)} & 0 & w_{1,0}^{(l)} & w_{1,1}^{(l)} & w_{1,2}^{(l)} & 0 & w_{2,0}^{(l)} & w_{2,1}^{(l)} & w_{2,2}^{(l)}
\end{pmatrix}
$$

$$
\mathbf{C}^{(l)} = \begin{pmatrix}
1 & 3 & 2 & 0 & 2 & 4 & 1 & 0 & 1 & 5 & 1 & 0 \\
0 & 1 & 3 & 2 & 0 & 2 & 4 & 1 & 0 & 1 & 5 & 1
\end{pmatrix}
$$

Unter Berücksichtigung der Gewichte $w_{i,j}^{(l)}$ des Filter Kernels $\mathbf{K}^{(l)}$ bezugnehmend auf Abbildung 2.11 mit der Matrix $\mathbf{C}^{(l)}$ gilt somit nach [Dumoulin u. Visin, 2018]:

$$vec\left(\mathbf{H}^{(l+1)}\right) = \mathbf{C}^{(l)} \times vec\left(\mathbf{H}^{(l)}\right) \tag{2.23}$$

Die Zeilen der Matrix $\mathbf{C}$ geben die Anzahl der Ausgangsdatenpunkte vor. Operationen wie Stride, Padding und Dilation können durch hinzugefügte Zeilen und Spalten in $\mathbf{H}^{(l)}$ und $\mathbf{C}^{(l)}$, sowie deren Besetzung abgebildet werden.
Für eine Umkehrung eben dieser Operationen wird die Matrix $\mathbf{C}^{(l)T}$ verwendet (vgl. [Shi u. a., 2016]). Bei der direkten Anwendung der Matrix $\mathbf{C}^{(l)T}$ auf den Ausgang der Featuremap $\mathbf{H}^{(l)}$ ergibt sich somit der Folgelayer $\mathbf{H}^{(l+1)}$ wie in Gleichung 2.24 definiert.

$$\mathbf{H}^{(l+1)} = Matrix\left(\mathbf{C}^{(l)T} \times vec\left(\mathbf{H}^{(l)}\right)\right) \tag{2.24}$$

Da es sich bei der Transposed Convolution nicht um die Umkehrfunktion der Faltung handelt, wird diese somit auch als backward strided convolution (vgl. [Long u. a., 2015]) oder fractional strided convolution (vgl. [Shi u. a., 2016]) bezeichnet.

Down- / Up-Sampling

Operationen, die häufig im Zusammenhang mit CNN verwendet werden, sind Down- und Up-Sampling. Ziel ist es, die Dimensionen der Featuremap $A^{(l)} \times B^{(l)}$ zu verringern (Down-Sampling) bzw. zu erhöhen (Up-Sampling).

Down-Sampling ist eine häufig genutzte Operation, um über die Verringerung der Dimensionen der Featuremaps den Rechenaufwand bei der Optimierung des Netzes zu verringern und das rezeptive Feld der darauffolgenden Filterung zu erhöhen (vgl. [Goodfellow u. a., 2016]). Neben der Faltung ist das Pooling eine geeignete Methode zur Verkleinerung der Dimensionen der Featuremaps.
Pooling-Operationen besitzen einen sogenannten Pooling Kernel $\left(K_P^{(l)}\right)$. Dieser wird mit einem Stride $s \geq (s_A, s_B)$ auf jeden Kanal der Featuremap angewendet (siehe Abb. 2.14). Mögliche Pooling-Operationen sind Max-Pool, bei dem der Maximalwert des Pooling Kernels übernommen wird, sowie Average-Pool, bei dem der Mittelwert innerhalb des Pooling Kernels ermittelt und übernommen wird (vlg. [Aggarwal, 2018]).

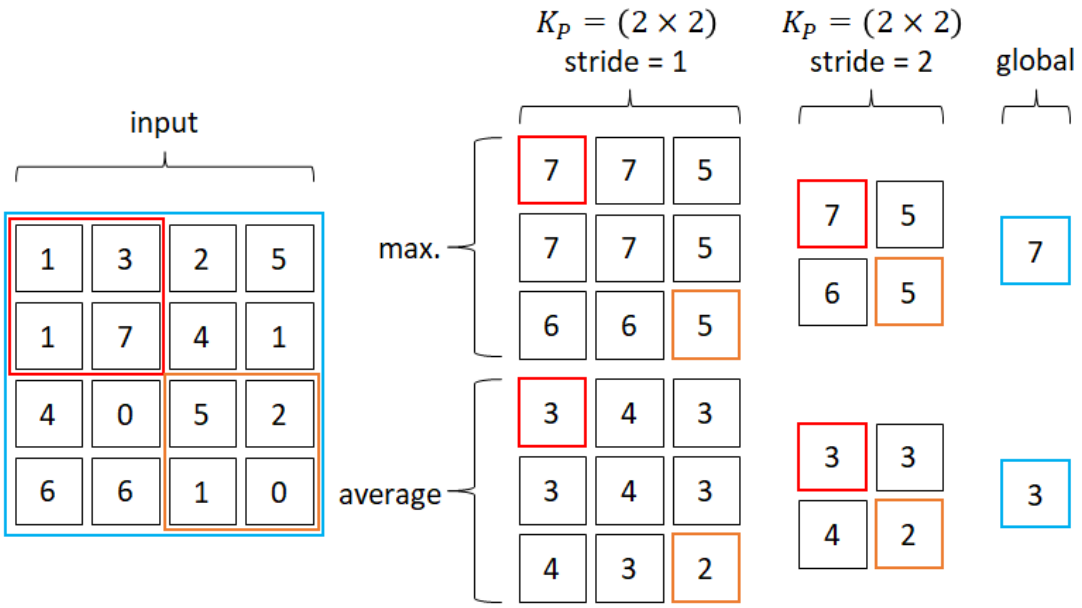


Abbildung 2.14: Beispiel für Max-, Average- und Global-Pooling

Weitere Variationen des Poolings sind Global-Pooling und Channel-Pooling. Das Global-Pooling ist ebenfalls in Abbildung 2.14 dargestellt. Hier wird der Pooling Kernel auf die gesamte Featuremap ausgedehnt $\left(K_P^{(l)} = A^{(l)} \times B^{(l)}\right)$ und es entsteht eine ‚Feature-Säule' mit der Dimension $1 \times 1 \times D^{(l)}$. Das Channel-Pooling hingegen ermöglicht das Pooling durch die Kanäle der Featuremap. Hier wird mit einem Pooling Kernel von $K_P^{(l)} = 1 \times 1 \times D^{(l)}$ gearbeitet und eine flache Featuremap erstellt, welche die Dimensionen $A^{(l)} \times B^{(l)} \times 1$ aufweist.

Up-Sampling wird im Gegensatz zu dem Down-Sampling verwendet, um die Dimensionen der Featuremaps zu vergrößern. Genau wie bei dem Down-Sampling existieren verschiedene Ansätze, die hier in zwei Kategorien aufgeteilt werden: Up-Sampling und learned Up-Sampling.
Beim Up-Sampling wird analog zum Down-Sampling ein sogenannter Pooling Kernel verwendet. In Abbildung 2.15 sind Beispiele für eine Vergrößerung der Featuremap dargestellt.

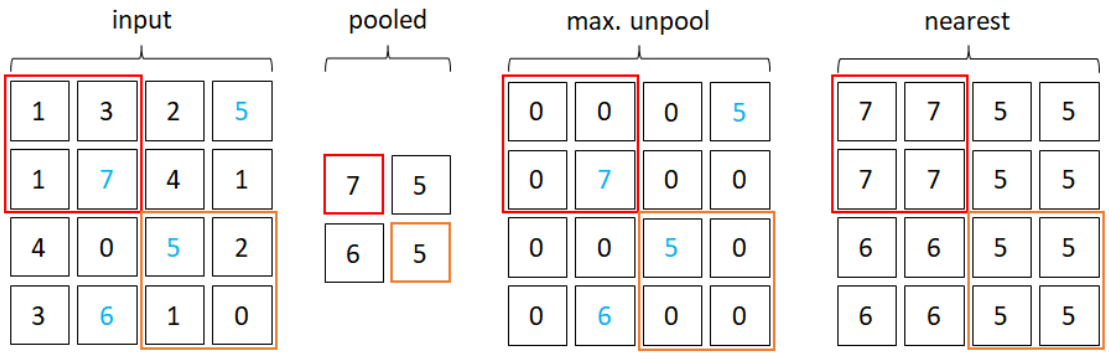


Abbildung 2.15: Up-Sampling mittels Max-Unpool und Nearest mit (2×2) Pooling Kernel und $(2, 2)$ Stride

Die hier gezeigten Up-Sampling Stategien sind Max-Unpool und Nearest. Das Max-Unpool ist nur in Architekturen möglich, in denen eine korrespondierende Max-Pool- zur Max-Unpool-Operation vorhanden ist. Hier wird die Position der für das Max-Pool verwendeten Datenpunkte für das Up-Sampling genutzt. Die restlichen Stellen des Pooling Kernels werden mit Nullen aufgefüllt. Die Nearest-Operation füllt den gesamten Pooling-Kernel mit dem jeweiligen Wert auf, was bei einer anschließenden Average-Pool-Operation wieder zum Ursprungswert führt.
Das learned Up-Sampling hingegen wird durch die Transposed Convolution realisiert (vgl. [Long u. a., 2015]) und besitzt somit einen Filter Kernel mit Gewichten für die Berechnung des Folgelayers $\mathbf{H}^{(l+1)}$. Auf diese Weise können die Dimensionen von Featuremaps über trainierbare Gewichte im Filter Kernel der Transposed Convolution durch Vormultiplikation der Matrix $\mathbf{C}^T$ erhöht werden. Somit ergibt sich für das learned Up-Sampling die bereits thematisierte Gleichung 2.24 (vgl. [Dumoulin u. Visin, 2018]).

Fully Connected Layer und Fully Convolutional Networks

Dieses Kapitel behandelt den Unterschied zwischen einem Fully Convolutional Network (FCN) und einem CNN mit Fully Connected Layer (FCL) sowie mögliche Anwendungsfälle, die im Folgenden genauer erläutert werden.

Convolutional Neural Networks mit Fully Connected Layer sind CNN mit einem Output hinter einem FCL beziehungsweise Dense Layer und werden zur Klassifizierung von Objekten verwendet (vgl. [Aggarwal, 2018]). Das Hauptmerkmal ist, dass jeder

Datenpunkt der $A^{(l)} \times B^{(l)} \times D^{(l)}$ großen Featuremap $\mathbf{H}^{(l)}$ mit einem Neuron des Dense Layers verbunden ist. Dies wird ermöglicht, indem die Featuremap vektorisiert und dem FCL als Eingang zur Verfügung gestellt wird (siehe Abb. 2.16).

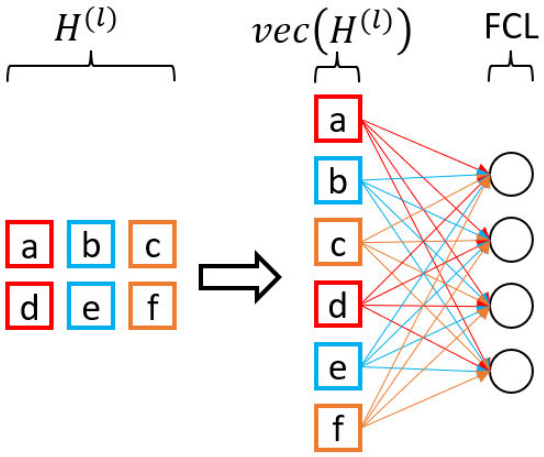


Abbildung 2.16: Convolutional Neural Network mit Fully Connected Layer (FCL) bei einem Kanal

Durch die Verbindung jedes Datenpunktes mit jedem Neuron des FCL wird hier eine Vielzahl von Gewichten benötigt. Je nach Eingangsgröße des Objektes und vorhandener Rechenleistung für das Training ist gegebenenfalls ein Down-Sampling erforderlich. Weiterhin wird das Netz durch die feste Anzahl an Gewichten am Eingang dahingehend beschränkt, dass alle Eingangsobjekte dieselbe Größe aufweisen müssen. Durch die Vektorisierung der Featuremap werden zudem Kontext- und somit Positionsinformationen eliminiert, sodass diese Methode ungeeignet für Lokalisierungsaufgaben ist. Reine Klassifikationsaufgaben können mit dieser Architektur bewältigt werden.

Fully Convolutional Networks besitzen den oben beschriebenen FCL nicht. Es handelt sich um eine Architektur, in der die Gewichte ausschließlich in Filter Kerneln enthalten sind. Durch die feste Anzahl an Gewichten pro Layer mit $F_A^{(l)} \times F_B^{(l)} \times D^{(l)}$ ist diese Archiektur gegebüber veränderlicher Eingangsgrößen $A^{(0)} \times B^{(0)}$ robust, nicht aber gegenüber veränderlicher Tiefe $D^{(0)}$ (vlg. [Long u. a., 2015]). Um bei Eingangsbildern variabler Größe eine minimale Kompression und dadurch einen minimalen Informationsverlust sicherzustellen, müssen die Dimensionen $A^{(0)} \times B^{(0)}$ des Eingangsbildes durch den ‚Shape-Factor' $sf = (sf_A, sf_B)$ aus Gleichung 2.25 ohne Rest teilbar sein.

$$
\begin{aligned}
sf_A &= \prod_{i=0}^{l-1} \frac{A^{(i)}}{A^{(i+1)}} \quad \forall \left\{ A^{(i)} > A^{(i+1)} \right\} \\
sf_B &= \prod_{i=0}^{l-1} \frac{B^{(i)}}{B^{(i+1)}} \quad \forall \left\{ B^{(i)} > B^{(i+1)} \right\}
\end{aligned}
\tag{2.25}
$$

Andernfalls kann es beim Down- und Up-Sampling zu Fehlern bei den Dimensionen der Featuremap kommen. Daher muss für das Netz der Shape-Factor vorab ermittelt werden, um Fehler wie in Abbildung 2.17 zu vermeiden.

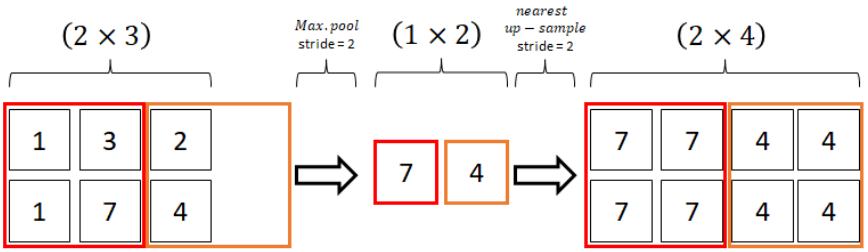


Abbildung 2.17: Fehlerhafte Dimensionen durch Down- und Up-Sampling bei einem Fully Convolutional Network

Durch ein Ausbleiben der FCL bietet ein FCN die Möglichkeit, Objekte nicht nur zu erkennen, sondern diese auch über die Positionsinformationen im Bild zu lokalisieren (vgl. [Long u. a., 2015]).

2.3 Semantische Segmentierung

Die semantische Segmentierung ist ein Verfahren zur Klassifizierung und Lokalisierung von Objekten in Bildern. Hierbei werden die Pixel eines Bildes einer definierten Anzahl von Klassen zugeordnet. In Abbildung 2.18 ist ein Beispiel einer solchen Segmentierung mit drei Klassen dargestellt.

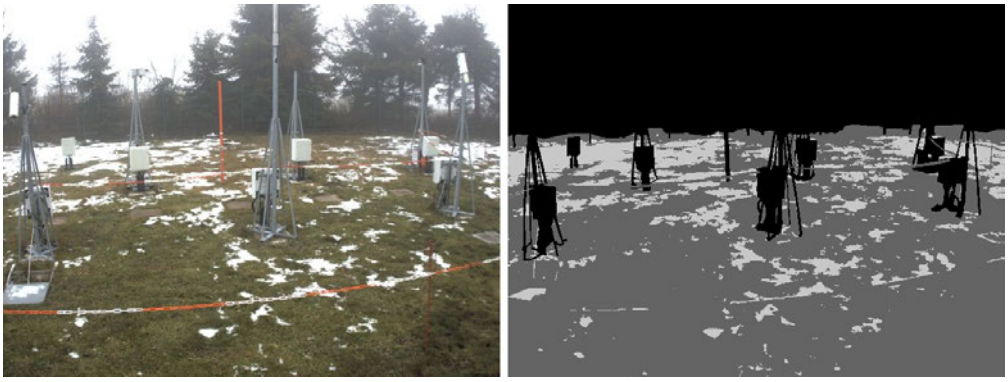


Abbildung 2.18: Beispiel einer semantischen Segmentierung mit Eingangsbild (links) und Ausgangsbild (rechts)

Die Klassen in diesem Beispiel sind: schneebedeckter Boden (weiß), unbedeckter Boden (grau) und Hintergrund (schwarz). Durch die Segmentierung können Aussagen über Häufigkeit und Position von Schneevorkommen getroffen werden.
Für die Lösung eines solchen Klassifizierungsproblems können, wie in den folgenden Abschnitten beschrieben, sowohl die merkmalsbasierte Klassifizierung mit und ohne maschinellem Lernen als auch Methoden des Deep Learnings eingesetzt werden. Weiterhin geht dieses Kapitel auf Parameter zur Auswertung der Qualität der semantischen Segmentierung ein.

2.3.1 Merkmalsbasierte semantische Segmentierung

In diesem Abschnitt wird die Methode der merkmalsbasierten semantische Segmentierung dargestellt. Ziel hierbei ist, jedem Pixel des Eingangsbildes eine Klasse wie beispielsweise ‚schneebedeckter Boden' oder ‚spezifizierte Objekte' zuzuordnen. Für diese Zuordnung kann jedes Pixel isoliert oder als Gruppe von Pixeln, die ein Merkmal darstellen, betrachtet werden. Bei dieser Segmentierungsstrategie ist es möglich, einen oder mehrere logische Zusammenhänge pro Klasse zu formulieren. Eine anschließende Anwendung dieser Regeln auf jeden Pixel beziehungsweise jedes Merkmal ordnet diese der entsprechenden Klasse zu, um das semantisch segmentierte Bild zu erstellen. Weiterhin kann die Interpretation der Farbkanäle an sich oder die Kombination der Grauwerte mit in die Regeln aufgenommen werden.

2.3.2 Merkmalsbasierte semantische Segmentierung mittels maschinellen Lernens

Ein weiterer Ansatz zur semantischen Segmentierung ist die Verwendug von Algorithmen des maschinellen Lernens. Der Ansatz des merkmalsbasierten maschinellen Lernens gibt die zu verwendenden Merkmale fest vor. Im Folgenden werden zwei Verfahren des maschinellen Lernens zur merkmalsbasierten semantischen Segmentierung beschrieben.

k-Nearest Neighbour-Klassifikator

Eine Möglichkeit der semantischen Segmentierung im Bereich des maschniellen Lernens bietet der k-Nearest Neighbour (kNN) - Klassifikator (vgl. [Cover u. Hart, 1967]). Hierbei ist ein Trainigsdatensatz notwendig. Dieser enthält bereits klassifizierte Objekte bzw. Regionen, aus denen anwendungsspezifische Merkmale extrahiert werden. Eine Standardisierung verhindert die Dominanz von Merkmalen mit großer Varianz (vgl. [von der Hude, 2020]). Bei anschließendem Training des Klassifikators wird ein n-dimensionaler Merkmalsraum aufgespannt, in dem jedes Objekt aus dem Trainigsdatensatz positioniert wird, wobei n die Anzahl der für die Klassifizierung genutzen Merkmale darstellt.
Für die semantische Segmentierung eines Bildes muss dieses zunächst geeignet vorverarbeitet werden, um die zu klassifizierenden Regionen zu extrahieren. Eine anschließende Berechnung der für den Anwendungsfall definierten Merkmale bildet die Grundlage der Klassifizierung. Diese Merkmale werden ebenso wie die der Trainingsdaten im Merkmalsraum positioniert. Durch eine Abstandsberechnung des zu klassifizierenden Objektes zu den Trainigsobjekten wird die Klassenzugehörigkeit ermittelt. Der Abstand d des zu klassifizierenden Objektes Q zu den Trainingobjekten T wird über n Merkmale berechnet. Eine verbreitete Methode der Abstandsberechnung ist die normalisierte euklidische Distanz aus Gleichung 2.26 (vgl. [Hu u. a., 2016]).

$$d(T,Q) = \sqrt{\frac{\sum_{i=1}^{n}(t_i - q_i)^2}{n}} \tag{2.26}$$

Die Zuweisung der Klasse erfolgt über die k geringsten Abstände (siehe Abb. 2.19).

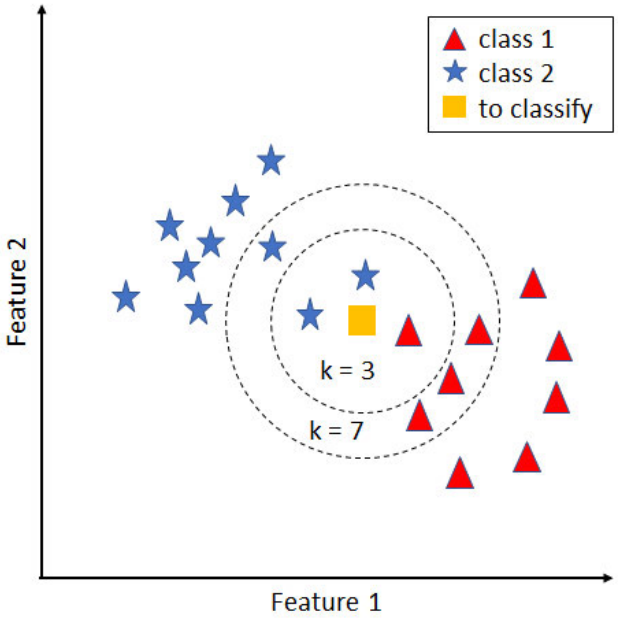


Abbildung 2.19: Beispiel der k-Nearest Neighbour Klassifizierung

Für dieses konkrete Beispiel wird das zu klassifizierende Objekt bei der Wahl von $k = 3$ der Klasse 2 und bei $k = 7$ der Klasse 1 zugeordnet. k ist geeignet zu wählen. Ein abschließendes Eintragen der Klasse in die Featuremap vervollständigt die Segmentierung für diese Region.

Support Vector Machine-Klassifikator

Ein weiterer Algorithmus des maschniellen Lernens, der bei der merkmalsbasierten semantischen Segmentierung verwendet wird, ist die Support Vector Machine (SVM)(vgl. [Müller u. Behnke, 2014]). Die SVM benötigt ebenso wie der kNN-Klassifikator aus Regionen extrahierte Merkmale als Eingangsparameter für die Klassifizierung. Anders als bei den kNN werden während des Trainings die Klassen nicht nur im Merkmalsraum positioniert, sondern auch Stützvektoren berechnet, die eine Hyperebene aufspannen. Die Abgrenzungen der Hyperebene verlaufen parallel durch die beiden Klassenpunkte mit der geringsten Distanz, sodass diese die Klassen voneinander trennen. Für eine optimale Trennung wird der Verlauf der Hyperebene mittels einer Kostenfunktion ermittelt. Bei entsprechender Parametrisierung der SVM sind bei der Berechung der Hyperebene Grenzverletzungen möglich.

Die Klassifizierung mittels SVM wird über die Position des zu klassifizierenden Objektes im Merkmalsraum durchgeführt. Wie in Abbildung 2.20 dargestellt, können die Objekte A (Klasse 1) und B (Klasse 2) eindeutig klassifiziert werden. Objekt C hingegen wird je nach Parametrierung keiner beziehungsweise einer ‚default' Klasse oder Klasse 1 durch Abstandsermittlung zugeordnet.

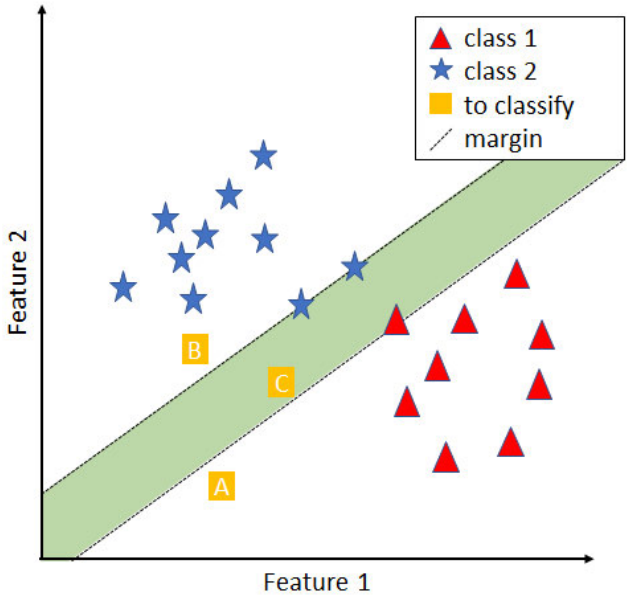


Abbildung 2.20: Beispiel der Klassifizierung mittels Support Vector Machine

2.3.3 Semantische Segmentierung mittels Deep Learning

Anders als bei den merkmalsbasierten Ansätzen bietet Deep Learning die Möglichkeit, eine semantische Segmentierung ohne vorgelagerte Merkmalsextraktion durchzuführen. Als Basis hierfür werden häufig CNN verwendet (vgl. [Farabet u. a., 2013]). Eine Vorverarbeitung des Bildes ist bei diesem Verfahren nicht ausgeschlossen und kann bei Bedarf ebenfalls erfolgen. Der Ausgang des neuronalen Netzes ist ein vollständig segmentiertes Bild in der Größe des Eingangsbildes, welches auch als ‚Labelmap'[4] bezeichnet wird (siehe Abb. 2.18).
Es muss entsprechend für jedes Pixel eine Aussage bezüglich der Klassenzugehörigkeit getroffen werden. Um dies zu ermöglichen und jeden Pixel im Kontext zu betrachten,

[4] Auch die Ground Truth beziehungsweise die annotierten Trainingsdaten werden bei der semantischen Segmentierung als Labelmap bezeichnet.

werden für die semantische Segmentierung sogenannte Encoder/Decoder Netzwerke eingesetzt (vgl. [Farabet u. a., 2013]). Ein solches Netzwerk zeichnet sich durch ein Down-Sampling des Eingangsbildes, gefolgt von einem Up-Sampling auf die Ursprungsgröße aus. Die dafür verwendeten Filteroperationen extrahieren hierbei die Featuremaps. Durch das mehrfache Down-Sampling und anschließende Up-Sampling wird das rezeptive Feld der Filterung in Bezug auf das Eingangsbild erhöht. Folglich repräsentieren bei zunehmender Netztiefe immer weniger Datenpunkte dieselbe Anzahl an Eingangsdatenpunkten. Eine Berücksichtigung von Informationen in einem immer größer werdenden Kontext wird hierdurch ermöglicht.

2.3.4 Auswertung der semantischen Segmentierung

Nachfolgend werden vier Parameter zur Auswertung und Optimierung der semantischen Segmentierung erläutert. Dazu zählt der Micro F1-Score, welcher einen Genauigkeitswert für eine bestimmte Klasse abbildet, der Macro F1-Score, der alle Klassen in der Vorhersage berücksichtigt, der Jaccard-Score, der die Überlagerung der Prediction gegenüber der Ground Truth bestimmt und die Pixelwise Accuracy (Acc), welche allgemein die Genauigkeit über alle Pixel des Bildes widerspiegelt.
Der F1-Score oder auch Micro F1-Score (bei neuronalen Netzen mit mehreren Klassen) wird von [Grandini u. a., 2020] beschrieben. Demnach berechnet sich der F1-Score nach Gleichung 2.27 mit ‚TP' als True Positive, ‚FP' als False Positive und ‚FN' als False Negative.

$$\begin{aligned} F1 &= 2 \cdot \left(\frac{Precision \cdot Recall}{Precision + Recall} \right) \quad \text{, mit} \\ Recall &= \frac{TP}{TP + FN} \\ Precision &= \frac{TP}{TP + FP} \end{aligned} \tag{2.27}$$

Durch Einsetzen von *Recall* und *Precision* ergibt sich Gleichung 2.28.

$$F1 = \frac{2 \cdot TP}{2 \cdot TP + FP + FN} \tag{2.28}$$

Der Micro F1-Score kann nach [Opitz u. Burst, 2021] auf alle Klassen des neuronalen Netzes erweitert werden. Damit ergibt sich der Macro F1-Score, welcher die Class Average Accuracy darstellt. Es wird zunächst der Micro F1-Score für alle n-Klassen separat berechnet und anschließend der Mittelwert gebildet (siehe Gl. 2.29).

$$F1_{macro} = \frac{1}{n} \sum_{x=1}^{n} F1_x \tag{2.29}$$

Der Jaccard-Score gibt nach [Jaccard, 1912] die Überlagerung von zwei Mengen an und wird auch als Intersection over Union (IoU) bezeichnet. Bei der Analyse der semantischen Segmentierung wird dieser benutzt, um die Überlagerung der prädizierten Klasse mit der Ground Truth darzustellen. Die Berechnung erfolgt nach Gleichung 2.30.

$$IoU = \frac{TP}{TP + FP + FN} \tag{2.30}$$

Die Erweiterung auf n-Klassen ist beim Jaccard-Score ebenso möglich. Dieser Parameter wird als mean Intersection over Union (mIoU) bezeichnet und nach Gleichung 2.31 berechnet.

$$mIoU = \frac{1}{n} \sum_{x=1}^{n} IoU_x \tag{2.31}$$

Die Pixelwise Accuracy stellt einen weiteren Parameter für die Auswertung der Qualität von neuronalen Netzen dar. Dabei wird nicht die Genauigkeit bezogen auf die einzelnen Klassen, sondern lediglich die pixelweise Übereinstimmung von Ground Truth und Prediction betrachtet. Somit berechnet sich die Accuracy folgendermaßen:

$$Acc = \frac{Anzahl\ übereinstimmender\ Datenpunkte}{Gesamtanzahl\ der\ Datenpunkte}$$

3 Stand der Technik

Im Folgenden wird der für diese Arbeit relevante Stand der Technik analysiert. Dabei geht es zum einen um Deep Learning-Architekturen, die generell für die semantische Segmentierung genutzt werden und zum anderen um Architekturen, die speziell für die Erkennung von Wetterereignissen wie Schnee oder Wolken verwendet werden. Auch etwaige Überschneidungen bei den Architekturen werden hier dargestellt. Abschließend wird ein Ansatz für Domain Adversarial Transfer Learning (DATL) sowie das dieser Arbeit übergeordnete Projekt ‚Snowcam' analysiert.

3.1 Semantische Segmentierung

Der erste Abschnitt dieses Kapitels beschreibt mit dem U-Net und dem DeepLab v3+ Architekturen, die für die semantische Segmentierung von Bildern genutzt werden. Im Zuge der Beschreibung des U-Nets wird auch die dazugehörige Erweiterung zum UX-Net nach [Komiyama u. a., 2018] analysiert.

3.1.1 U-Net Architektur

Eine weit verbreitete Methode zur semantischen Segmentierung von Bildern ist das U-Net, welches den FCN zuzuordnen ist. Das U-Net hat eine U-förmige Struktur aus einem Encoder und einem Decoder, welche über die sogenannte Bridge verbunden sind (siehe Abb. 3.1). Außerdem erfolgt eine einseitige Übertragung von Informationen vom Encoder zum Decoder, welche in Abbildung 3.1 anhand von grauen Pfeilen angedeutet wird.

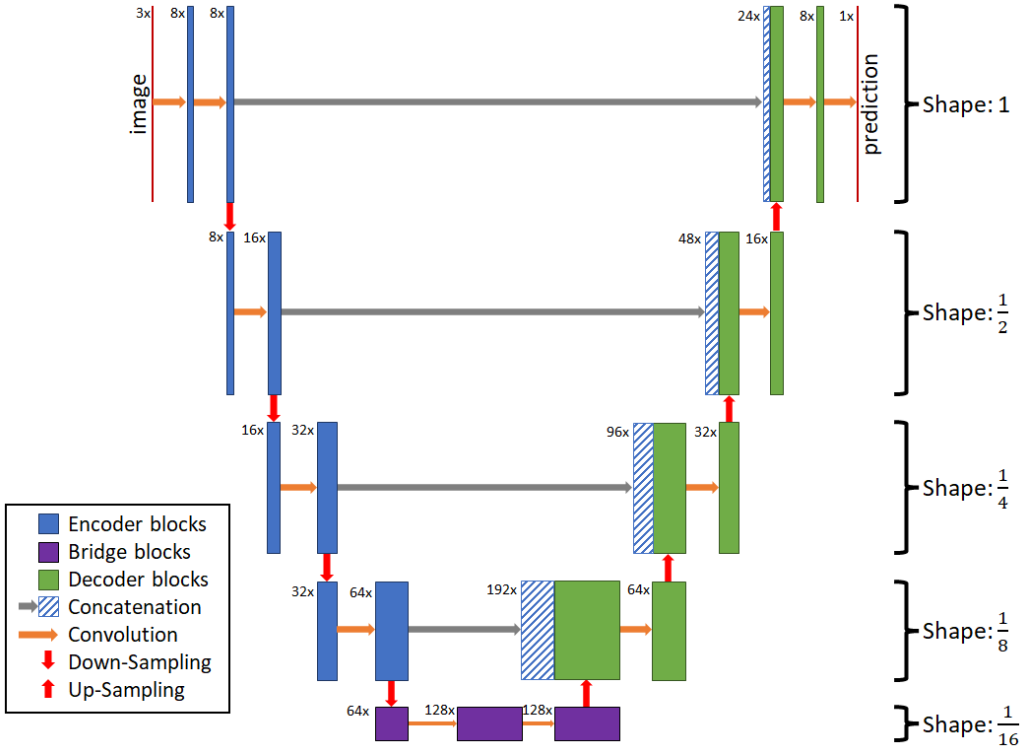


Abbildung 3.1: Beispielhafter Aufbau eines U-Nets

Im Encoder wird zunächst schrittweise die Anzahl der Featuremaps über Faltungsoperationen erhöht. Dadurch wird der Merkmals-Informationsgehalt der Eingangsbilder erhöht. Über Down-Sampling-Operationen wird die Auflösung der Bilder im Encoder immer weiter verringert und das rezeptive Feld der nachfolgenden Faltungsoperationen damit vergrößert. Dies ist notwendig, um Rechenkapazitäten zu sparen und Merkmale mit höherem Kontext zu extrahieren (vgl. [Kim u. a., 2017]). Bei der Bridge angelangt, besitzen die Bilder durch die geringe Auflösung nahezu keine räumlichen Informationen mehr. Dafür besitzen sie durch die hohe Anzahl der Featuremaps einen großen Informationsgehalt bezüglich verschiedener Merkmale. Im Decoder werden die räumlichen Informationen schrittweise wiederhergestellt. Nach [Ronneberger u. a., 2015] werden durch die hohe Anzahl an Featuremaps im Decoder Kontextinformationen in die höher aufgelösten Layer getragen. Für die Wiederherstellung der räumlichen Informationen im Decoder wird der Ausgang der zugehörigen Layer des Encoders verwendet. Diese werden, wie in Abbildung 3.1 erkennbar, vor der Faltung an den Eingang der jeweiligen Decoder-Layer gehängt. Durch die nachfolgende Faltung wird die Anzahl der Featuremaps im Decoder wieder reduziert und so die daraus gewonnenen Merkmals-Informationen zusammengefasst. Durch

Up-Sampling-Operationen wird nun symmetrisch zum Encoder die Auflösung wieder bis zur Ursprungsgröße der Bilder erhöht. Das Ausgangsbild des U-Nets hat schließlich das selbe Format wie das dazugehörige Eingangsbild. Die Anzahl der Faltungen pro Layer und welche Dimensionen die Sampling und Filter Kernel besitzen, ist auf den jeweiligen Anwendungsfall zu optimieren. Auch die Anzahl der Layer des U-Nets ist nicht fest vorgegeben.

3.1.2 UX-Net Architektur

Eine Erweiterung des U-Nets wird von [Komiyama u. a., 2018] beschrieben. Dabei handelt es sich um das UX-Net, welches zwei zusätzliche, sich kreuzende Verbindungen zwischen Encoder und Decoder enthält. Wie in Abbildung 3.2 ersichtlich ist, wird dabei dem ersten Decoder-Layer das Ausgangsbild des ersten Encoder-Layer zur Verfügung gestellt. Darüber hinaus wird dem Output-Layer das Ausgangsbild des letzten Encoder-Layer bereitgestellt.

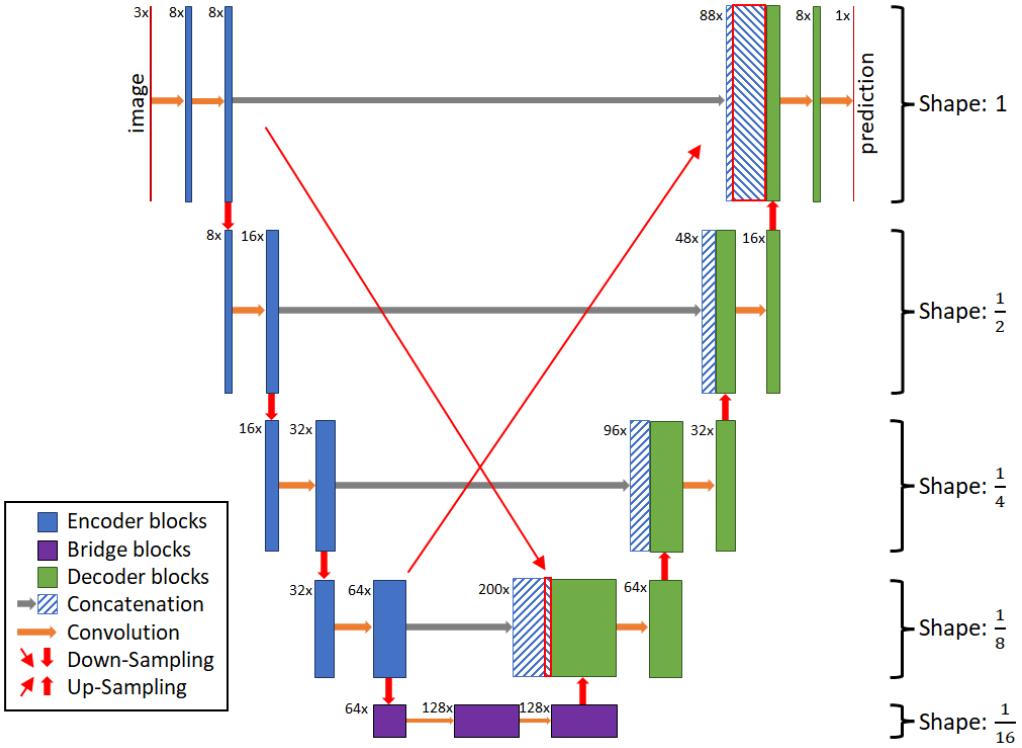


Abbildung 3.2: Beispielhafter Aufbau eines UX-Nets

Nach [Komiyama u. a., 2018] kann die Prediction mittels U-Net durch die zusätzlichen Verbindungen des UX-Nets verbessert werden. Im Detail wird der Input-Layer des Decoders durch Informationen zu Positionierungen, Kanten und kleinen Objekten des ersten Encoder-Layer ünterstützt. Darüber hinaus unterstützen die Klassifizierungs-Informationen der Featuremaps des letzten Encoder-Layer den Output-Layer des Decoders. Dadurch wird die Class Average Accuracy, in Bezug auf ein analoges U-Net, von $81,49$ % auf $90,76$ % erhöht. Dieser Wert ist vor allem dann ausschlaggebend, wenn viele kleinere Objekte oder Flächen erkannt werden müssen, die einen eher geringen Anteil der Pixelanzahl des Bildes ausmachen. Zusätzlich sinkt die Pixelwise Accuracy von $98,08$ % auf $97,70$ %. Wie in Abbildung 3.2 zu erkennen ist, müssen die Bilder an dieser Stelle in ihrer Auflösung mit Hilfe von Down- beziehungsweise Up-Sampling dem jeweiligen Ziel-Layer angepasst werden.

3.1.3 DeepLab v3+ Architektur

Das DeepLab v3+ von [Chen u. a., 2018] ist eine weitere Architektur zur semantischen Segmentierung von Bildern. Das DeepLab v3+ gehört ebenso wie das U-Net zu den FCN und ist als Encoder-Decoder-Netzwerk aufgebaut (siehe Abb. 3.3). Als Encoder wird das DeepLab v3 (vgl. [Chen u. a., 2017]) verwendet. Dieses stellt zusätzlich zu der High-Level Featuremap eine Low-Level Featuremap zur Verfügung. Die Kombination der semantischen High-Level Featuremap und der Positions-Informationen der Low-Level Featuremap generieren die Prediction dieser Architektur.

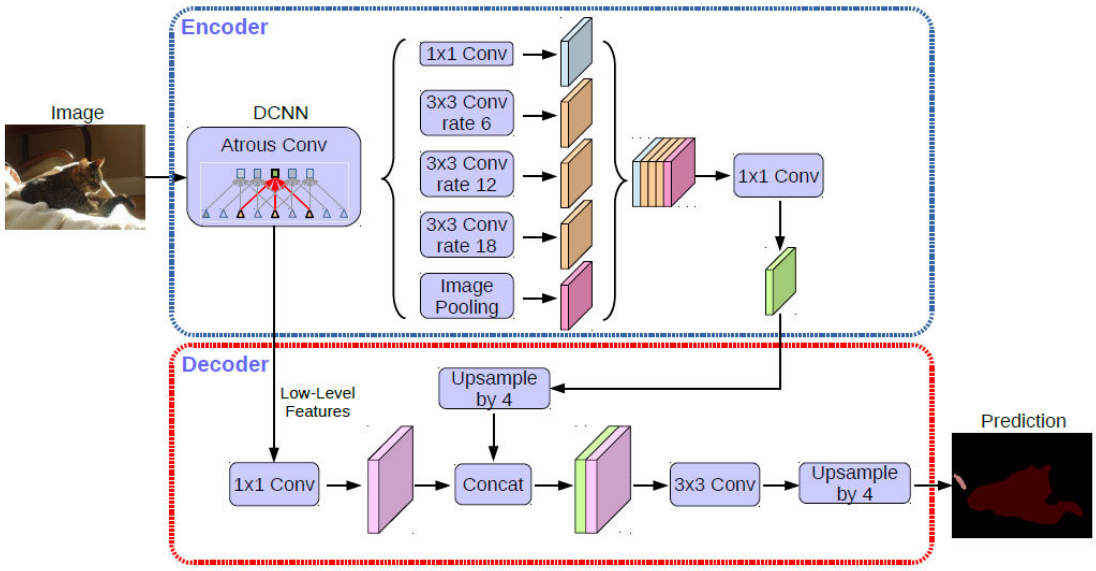


Abbildung 3.3: Encoder und Decoder Architektur des DeepLab v3+ (vgl. [Chen u. a., 2018])

Der Encoder lässt sich in zwei Hauptbestandteile zerlegen: die Atrous Depthwise Separable Convolution (ADSC) und das Atrous Spatial Pyramid Pooling (ASPP). Die ADSC, welche in Abbildung 3.3 als Deep Convolutional Neural Network (DCNN) eingeführt wird, komprimiert die Featuremaps auf ein Sechzehntel der ursprünglichen Größe und bietet jedem Kanal ein hohes rezeptives Feld durch die Dilated Convolutions. Die Low-Level Featuremap wird dem Decoder mit einem Viertel der ursprünglichen Größe zur Verfügung gestellt, um Positionsinformationen nutzen zu können. Der Ausgang der ADSC wird im ASPP über fünf verschiedenen Operationen verarbeitet, welche in ihrem rezeptiven Feld einer Pyramide ähneln. Über Faltungsoperationen mit wechselndem Filter Kernel, verschiedenen Dilation Rates und Image Pooling werden die Featuremaps extrahiert. Diese Kombination der unterschiedlichen rezeptiven Felder ermöglicht es, Informationen sowohl im gesamten Bildkontext als auch in einem Bildausschnitt zu erfassen. Eine abschließende Faltung der konkatenierten Featuremaps bildet die Schnittstelle zum Decoder.
Der Encoder-Ausgang wird zunächst um ein Vierfaches vergrößert. Ein anschließendes Verketten mit der Low-Level Featuremap, welche eine weitere Faltung erfährt, verbindet die tiefen Klasseninformationen mit Positionsinformationen der Low-Level Featuremap. Die Prediction wird durch eine abschließende Faltung und ein weiteres Up-Sampling auf das Format des Eingangsbildes komplettiert.

Die Encoder Struktur des DeepLab v3+ ermöglicht nach [Chen u. a., 2018] sowohl die Verwendung der Kontext- als auch der Positionsinformationen des Eingangsbildes und ist gut geeignet, um Klassen mit prägnanten Merkmalen zu prädizieren. Weiterhin bietet das DCNN durch definierte Ausgangsgrößen des Encoders die Möglichkeit, das ADSC durch verschiedene validierte Strukturen zu ersetzen.
Eine Analyse der Performance erfolgt in [Khan u. a., 2020]. Hier wird das DeepLab v3+ mit einem Xception Netz (vgl. [Chollet, 2017]) als DCNN verschiedenen anderen Netz-Architekturen gegenübergestellt. Die Auswertung des F1-Scores ergibt bei einem Trainingsdatensatz mit zwei Klassen einen Wert von $91,9\ \% \pm 2,0\ \%$, wobei das verglichene U-Net einen F1-Score von $88,4\ \% \pm 3,7\ \%$ aufweist.

3.2 Wetterbeobachtung

Dieses Kapitel behandelt die Verwendung von neuronalen Netzen bei der Erkennung von Wetterereignissen wie Schnee oder Wolken. Dabei werden beispielhaft zwei Verfahren näher betrachtet, welche für diese Aufgabe genutzt werden. Nachfolgend wird zunächst eine Architektur beschrieben, welche Wolkenerkennung auf Basis einer modifizierten U-Net Architektur realisiert. Außerdem wird ein Verfahren zur Schnee- und Wolkenerkennung basierend auf einer modifizierten DeepLab v3+ Architektur näher betrachtet.

3.2.1 Wolkenerkennung auf Basis einer U-Net Architektur

[Li u. a., 2018] nutzen für die Erkennung von Wolken und die von Wolken verursachten Schatten eine Multi-Scale Convolutional Feature Fusion Architektur, die als Encoder und Decoder die Struktur eines symmetrischen U-Nets aufweist. Eine Besonderheit in der U-Net Architektur von [Li u. a., 2018] ist die Verwendung von Dilated Convolutions in den letzten zwei Faltungsblöcken des Encoders und den ersten zwei Faltungsblöcken des Decoders. Dadurch werden Merkmale aus einem größeren Bereich des Bildes extrahiert. Darüber hinaus gibt es eine weitere Besonderheit gegenüber der klassischen U-Net Architektur. [Li u. a., 2018] übertragen die Featuremaps, anstatt nur am Ende jedes Encoder-Layers, nach jedem CBRR-Block an die dazugehörige Position im Decoder (siehe Abb. 3.4). Parallel zum Decoder nutzt diese Architektur ein Multi-level/scale Feature Fusion Module (MFFM), welches, wie in Abbildung 3.4 zu sehen ist, die Featuremaps

nach den einzelnen Faltungs-Blöcken des Decoders abgreift und diese Multi-scale Features fusioniert. Daraus wird schließlich im letzten Faltungs-Layer der Ausgang gebildet.

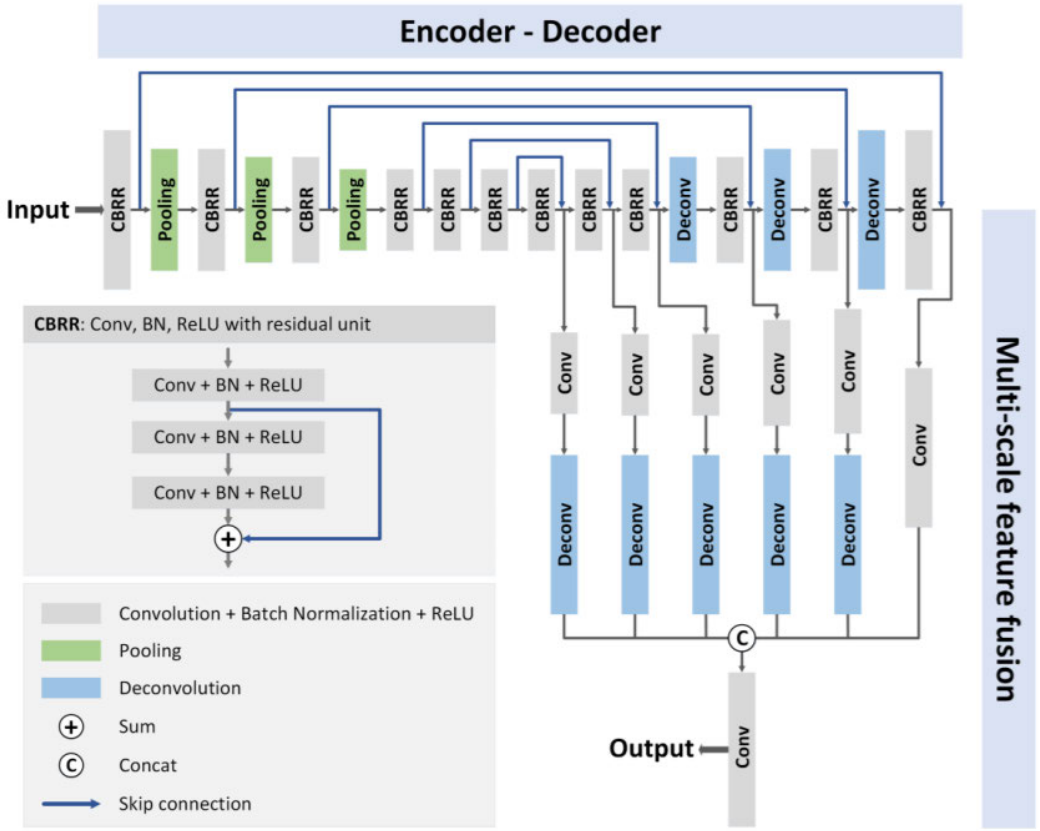


Abbildung 3.4: U-Net Architektur inkl. Multi-level/scale Feature Fusion Module (vgl. [Li u. a., 2018])

Abbildung 3.5 zeigt das Flussmodell der Architektur mit allen internen Verbindungen. Dort ist erkennbar, wie das MFFM die Featuremaps verschiedener Größe erhält, diese an die Größe des Eingangsbildes angleicht und schließlich fusioniert.

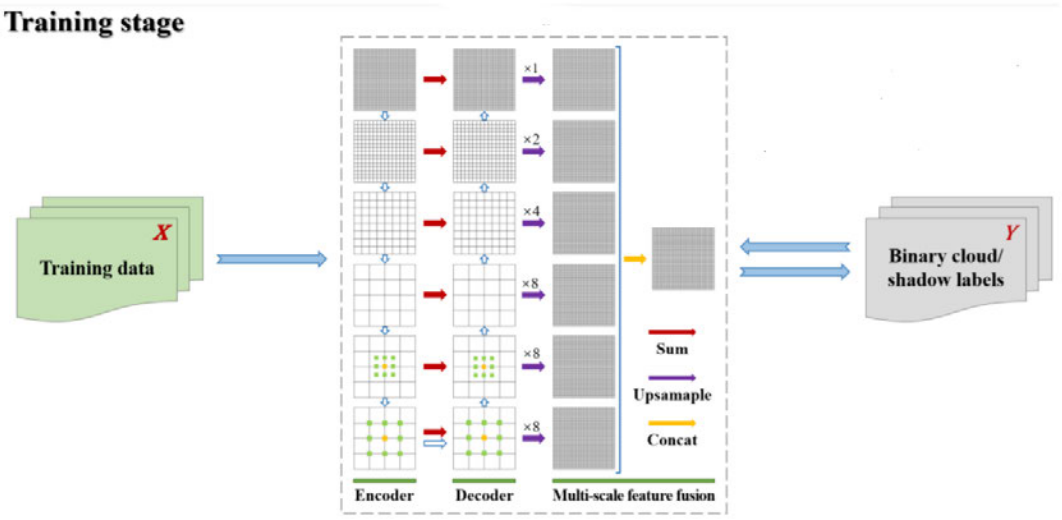


Abbildung 3.5: Flussmodell der U-Net Architektur inkl. Multi-level/scale Feature Fusion Module (vgl. [Li u. a., 2018])

Abbildung 3.5 zeigt, dass in den letzten zwei CBRR-Blöcken des Encoders und in den ersten zwei CBRR-Blöcken des Decoders Dilated Convolutions genutzt werden, um das rezeptive Feld dort zu erhöhen und Merkmale in einem größeren Bereich des Bildes zu betrachten.
Mit dieser Erweiterung des U-Nets um die Feature Fusion des MFFM werden abhängig vom Datensatz nach [Li u. a., 2018] F1-Scores von 93, 1 % bis 94, 9 % erreicht. Die Performance dieses Netzes liegt dabei klar vor der eines von [Li u. a., 2018] gegenübergestellten DeepLab Netzes, welches auf den selben Datensätzen F1-Scores von 83, 0 % bis 92, 2 % erreicht.

3.2.2 Schnee- und Wolkenerkennung auf Basis einer DeepLab Architektur

Eine weitere Herausforderung bei der Auswertung von Wetterdaten ist die Selektierung von Schnee und Wolken auf Satellitenaufnahmen. Die Arbeit von [Hongcai u. a., 2019] stellt einen möglichen Ansatz für diese Problemstellung vor. Die hier durchzuführende semantische Segmentierung wird anhand eines FCN durchgeführt, dessen Struktur der des DeepLab v3+ ähnelt (siehe Abb. 3.6).

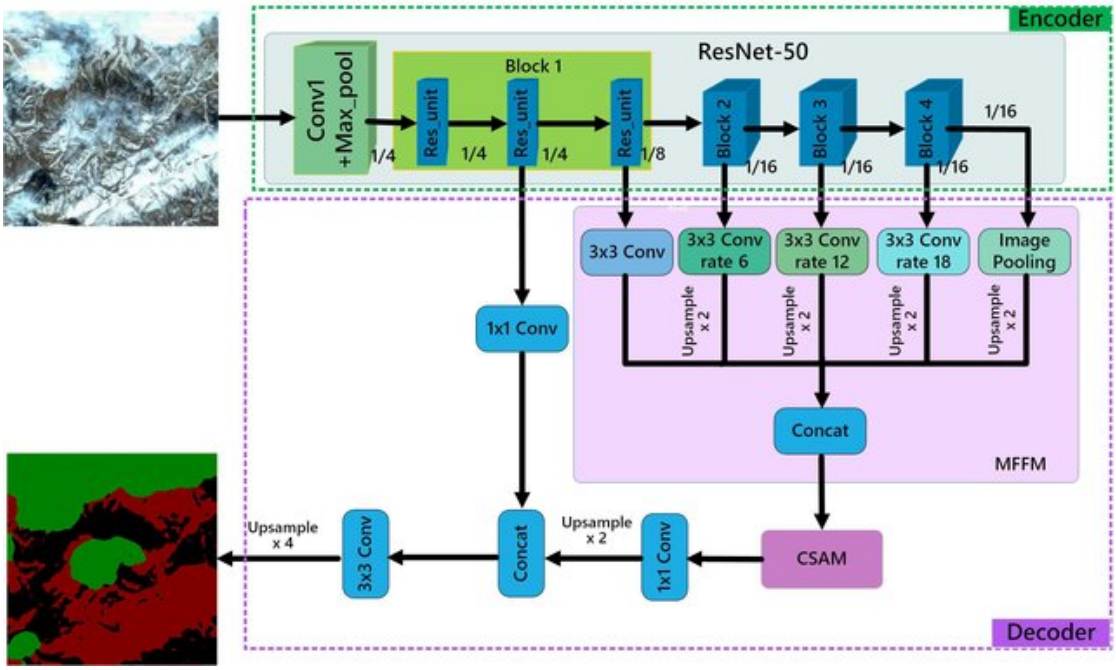


Abbildung 3.6: Fully Convolutional Network zur semantischen Segmentierung von Wolken und Schnee (vgl. [Hongcai u. a., 2019])

Als Encoder-DCNN wird hier das ResNet-50 (vgl. [He u. a., 2016]) anstatt des ADSC verwendet. Die Low-Level Featuremap wird dem Encoder, wie beim DeepLab v3+, bei einem Viertel der ursprünglichen Bildgröße entnommen. Anschließend werden Featuremaps bis hin zu einem Sechzehntel der Eingangsgröße erstellt. Anders als bei dem DeepLab v3+ wird das ASPP durch ein MFFM ersetzt, welches dem Decoder zuzuordnen ist. Das MFFM erhält, anders als das ASPP im DeepLab v3+, nicht nur Featuremaps mit einem Sechzehntel der Eingangsgröße an einer Position des Encoders, sondern darüber hinaus auf einem Pfad auch Featuremaps mit einem Achtel der ursprünglichen Bildgröße. Dies ermöglicht, in Kombination mit der Erhöhung der rezeptiven Felder in Pyramidenform, eine noch breitere Anzahl an Merkmalen zu extrahieren. Ein anschließendes Up-Sampling der Featuremaps bildet den Übergang zum Channel and Spatial Attention Module (CSAM).
Das CSAM ist ebenso eine Erweiterung zum DeepLab v3+. Diese besteht aus zwei Bausteinen: dem Channel Attention Module (CAM) und dem Spatial Attention Module (SAM) (siehe Abb. 3.7).

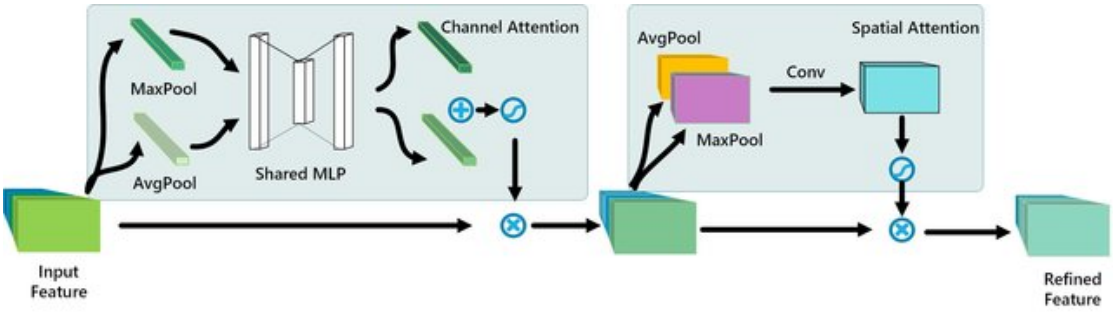


Abbildung 3.7: Channel and Spatial Attention Module (vgl. [Hongcai u. a., 2019])

Das CAM hat die Aufgabe, die einzelnen Featuremaps zu gewichten. Dies wird durchgeführt, indem auf jeden Kanal der Featuremap jeweils ein Global-Max- und Global-Average-Pooling ausgeführt wird. Die Ergebnisse dieser Operationen werden separat voneinander durch ein Shared MLP verarbeitet. Eine Addition der Ergebnisse mit anschließender Aktivierung durch eine Sigmoidfunktion bildet den Gewichtungsvektor der Featuremaps. Ziel hierbei ist es, die für die Selektion von Schnee und Wolken aussagekräftigsten Featuremaps in ihrer Gewichtung zu stärken und weniger aussagekräftige Featuremaps zu schwächen.
Der zweite Teil des CSAM ist das SAM. Dieses gewichtet, anders als das CAM, die räumlichen Informationen. Es wird je ein Channel-Max- und Channel-Average-Pooling durchgeführt. Mit einer Konkatenation mittels Faltung und abschließender Aktivierung durch eine Sigmoidfunktion wird eine einzelne Featuremap erstellt, welche als Gewichtungskarte dient. Diese Gewichtungskarte verstärkt die für die Erkennung von Schnee und Wolken aussagekräftigen geometrischen Informationen und schwächt jene, die weniger prägnant sind.
Der Ausgang des CSAM mit dem darauffolgenden Up-Sampling und Zusammenführung mit den Low-level Featuremaps bildet den Übergang zur ursprünglichen DeepLab v3+ Architektur aus Kapitel 3.1.3.
Durch die Erweiterung des DeepLab v3+ um ein MFFM und die Verwendung des ResNet-50 Encoders kann die mIoU nach [Hongcai u. a., 2019] von 88,87 % auf 89,12 % erhöht werden. Durch die Einbindung des CSAM wird der mIoU zusätzlich auf 89,25 % gesteigert.

3.3 Domain Adversarial Transfer Learning

Das Erstellen von Daten für das Training künstlicher neuronaler Netzte ist aufwendig und oft stehen zu wenig geeignete Trainingsdaten zur Verfügung. Um dieses Problem zu schmälern, sollen Ansätze wie das Domain Adversarial Transfer Learning (DATL) Abhilfe schaffen. Ziel des DATL ist es, ein künstliches neuronales Netz zu schaffen, welches in der Lage ist, Bilddaten einer bisher unbekannten Domain durch Training mit Trainingsdaten einer ähnlichen, bekannten Domain zu klassifizieren, beziehungsweise dafür Labelmaps zu erstellen. Außerdem sollen die Klassifizierungsergebnisse für die verwendeten Domains verbessert werden. Nach [Brion u. a., 2021] kann die Domain-Übertragung auf Merkmalsebene, Bildebene und Labelebene aufgeteilt werden.
Die Ansätze zur Übertragung auf Bild- und Label-Ebene nutzen Generative Adversarial Networks (GANs) welche zum einen Quell-Trainingsbilder so adaptieren, dass diese Ähnlichkeiten zu den Ziel-Bildern aufweisen (Image-Level) und zum anderen Pseudo-Labelmaps für Ziel-Daten prädizieren (Label-Level).
Bei der Domain-Übertragung auf Merkmalsebene wird einem Segmentierungsnetzwerk, im Beispiel aus Abbildung 3.8 einem U-Net, ein Domain-Klassifikator angehängt (vgl. [Brion u. a., 2021]). Dieser Domain-Klassifikator wird sowohl mit Bildern der Source-Domain als auch mit Bildern der Target-Domain trainiert und lernt, zwischen diesen zu unterscheiden.

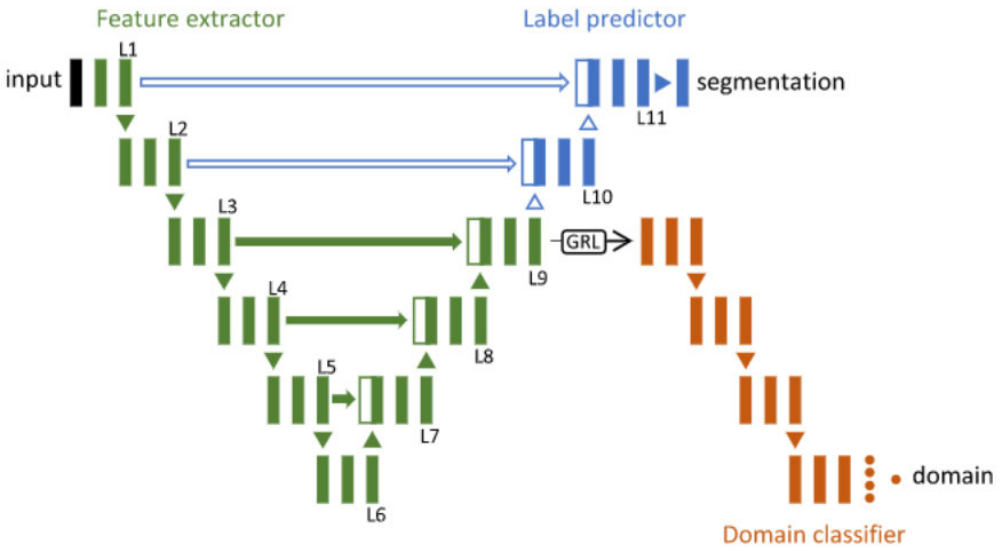


Abbildung 3.8: Domain Adversarial Transfer Learning auf Merkmalsebene (vgl. [Brion u. a., 2021])

Der Domain-Klassifikator ist über einen Gradient Reversal Layer (GRL) mit dem Segmentierungsnetz verbunden, wodurch der Gradient seiner Loss-Function negativ in die Optimierung eingeht. Nach [Scannell u. a., 2020] wird der Loss des Domain-Klassifikators dadurch im Segmentierungsnetzwerk maximiert und bewahrt dieses so vor der Wiederherstellung von Domain-Informationen. Auf diese Weise werden im Segmentierungsnetzwerk, in Verbindung mit der Nutzung verschiedener Augmentations, möglichst domaininvariante Features extrahiert und so die Generalisierungsleistung des neuronalen Netzes erhöht. Eine domainunabhängige Nutzung des neuronalen Netzes soll so ermöglicht werden.
[Scannell u. a., 2020] vergleichen ein um einen Domain-Klassifikator erweitertes U-Net mit der U-Net Architektur von [Ronneberger u. a., 2015]. Hierbei ist über drei Segmentierungsaufgaben unbekannter Domains eine Steigerung des F1-Scores von 7 % bis 10 % zu verzeichnen. Außerdem verbessert das DATL die Segmentierungergebnisse für jene Domains, die für das Training verwendet werden. Somit hat die Extraktion domaininvarianter Features nicht nur einen positiven Effekt auf die Segmentierung unbekannter Domains.

3.4 Projekt Snowcam

Bei dem Projekt ‚Snowcam' handelt es sich um ein Kooperationsprojekt der Hochschule für Angewandte Wissenschaften Hamburg (HAW)[5] und des Deutschen Wetterdienstes (DWD)[6]. Ziel dieses Projektes ist es, die Ermittlung des Schneebedeckungsgrades mittels Bodenkameraaufnahmen durch verschiedene Methoden zu automatisieren. Im Folgenden werden die zur Verfügung gestellten Datensätze sowie bereits durchgeführte Arbeiten zur Ermittlung des Schneebedeckungsgrades erläutert.

3.4.1 Verfügbare Datensätze

Um eine kamerabasierte Schneeerkennung umsetzen und validieren zu können, sind Aufnahmen mit realistischen Szenerien erforderlich. Dafür stehen drei Datensätze zur Verfügung. Bei den Datensätzen handelt es sich um Aufnahmen der Wasserkuppe[7] bei Tag und Nacht, eines Gartens und eines Flughafenvorfeldes. Die Tagaufnahmen des Messfeldes auf

[5] https://www.haw-hamburg.de .
[6] https://www.dwd.de .
[7] Messfeld des DWDs (https://www.dwd.de/DE/wetter/wetterundklima_vorort/hessen/wasserkuppe/_node.html (zul. aufgerufen am 24.03.2022)).

der Wasserkuppe sind Farbaufnahmen, wohingegen die Nachtaufnahmen einkanälige Infrarot (IR)-Aufnahmen sind (siehe Abb. 3.9). Die Bildgröße dieses Datensatzes beträgt 640×480 Pixel.

Abbildung 3.9: Farbaufnahme der Wasserkuppe bei Tag (links) und IR-Aufnahme der Wasserkuppe bei Nacht (rechts)

Die Gartenbilder enthalten keine Aufnahmen bei Dunkelheit. Die Bildgröße beläuft sich auf 220×240 Pixel. Der Datensatz des Flughafens hingegen stellt Aufnahmen bei Tag und Nacht zur Verfügung, wobei die genutzte Kamera Nachtaufnahmen als Farbbilder darstellt. Die Originalgröße der Aufnahmen beträgt 6000×4000 Pixel, wird jedoch durch entfernen des DWD-Logos am Bildrand und Reduzierung der Auflösung auf 600×360 Pixel herabgesetzt.
Die Datensätze der Wasserkuppe werden folgend als *waterday* sowie *waternight* bezeichnet und sind in fünf Referenzklassen nach [DWD, 2021] vorsortiert: unbedeckt (0 %), Schneereste ($>$ 0 % bis $<$ 10 %), Schneeflecken (10 % bis $<$ 50 %), durchbrochene Schneedecke (50 % bis $<$ 100 %) und geschlossene Schneedecke (100 %). Die Datensätze des Flughafens und des Gartens, die nachfolgend als *muccam01* und *garden* bezeichnet werden, sind in elf Referenzklassen in 10 % Schritten von 0 % bis 100 % vorsortiert.

3.4.2 Klassifizierung des Schneebedeckungsgrades

In diesem Abschnitt werden bereits evaluierte Ergebnisse aus vorangegangenen Arbeiten dieses Projektes zur Ermittelung des Schneebedeckungsgrades herausgearbeitet. Sowohl ein Ansatz der klassischen Bildverarbeitung, als auch die Klassifizierung unter Zuhilfenahme künstlicher neuronaler Netze werden analysiert.

Klassifizierung mittels klassischer Bildverarbeitung

Die klassische Bildverarbetung stützt sich auf eine geeignete Vorverarbeitung der Aufnahmen sowie eine Beschränkung auf die Auswertung des Bildbereichs, der keinen Himmel zeigt, um Komplikationen bei der Klassifizierung des Himmelsraums zu umgehen. Durch diesen Ansatz wird für den Datensatz *garden* mit der Einordnung in die fünf Referenzklassen aus Kapitel 3.4.1 eine Accuracy von $73,2$ % erreicht. Bei einer Einordnung in elf Referenzklassen sinkt die Accuracy auf $62,8$ %. Der Mean Absolute Error (MAE) beträgt in beiden Fällen $6,8$ % (vgl. [Padberg, 2021]).

Klassifizierung mittels künstlicher neuronaler Netze

Die Arbeit nach [Perkovic, 2022] verwendet CNN zur Extration von Features mit FCL zur Klassifizierung. Als CNN werden die vorhandenen Architekturen VGG16, Xception und DenseNet201 verwendet (vgl. [Perkovic, 2022]). Bei Verwendung des Datensatzes *muccam01* und einer Klassifizierung in elf Referenzklassen kann eine Accuracy von $93,98$ % bis $94,63$ % erreicht werden. Die Accuracy bei Verwendung von fünf Referenzklassen erreicht $95,52$ % bis $96,55$ %. Ein Training mit dem, aus IR-Aufnahmen bestehenden, Datensatz *waternight* erreicht eine Accuracy von 100 % mit der VGG16- und $95,71$ % mit der Xception-Architektur.
Weiterhin wird die Auswirkung des Domain Transfers analysiert. Ein Weitertrainieren eines mit *muccam01* trainierten künstlichen neuronalen Netzes mit dem Datensatz *waterday* lässt die Accuracy in fünf Referenzklassen auf $76,90$ % sinken. Abhilfe schafft hier das Training mit einem gemischten Datensatz, wodurch bei fünf Referenzklassen wieder eine Accuracy von $95,86$ % erreicht wird. Die Einbindung aller in [Perkovic, 2022] verwendeten Datensätze lässt die Accuracy auf $78,76$ % sinken. Die Laufzeitanalyse ergibt eine Verarbeitungszeit der Prediction von $0,47$ bis zu $3,68$ Sekunden abhänging von Architektur und Bildgröße.

4 Anforderungsanalyse

Das folgende Kapitel thematisiert die an die Arbeit gestellten Anforderungen. Dabei wird eine Unterscheidung der Priorität (Prio.) zwischen erforderlichen (erf.)- und optionalen (opt.)-Anforderungen getroffen. Die Anforderungsanalyse ist aufgeteilt in die Hauptanforderungen des Projektes, Qualitätsanforderungen, Anforderungen an die Software-Implementierung und spezielle Anforderungen an die Anwendung für Endnutzer*innen.

4.1 Allgemeine Anforderungen

Das übergeordnete Thema dieser Arbeit ist die semantische Segmentierung von Kameraaufnahmen, welche zur Schnee- und Wolkenerkennung eingesetzt wird. Die daraus abgeleiteten Hauptanforderungen A1 und A2 (siehe Tab. 4.1) stützen sich auf die aufbereiteten Daten aus Anforderungen A8 sowie auf die Software für die Netzimplementierung aus A7.
Ein weiteres Hauptanforderungsgebiet ist die Berechnung des Schnee- beziehungsweise Wolkenbedeckungsgrades und die Einordnung in die Referenzklassen aus den Anforderungen A3 bis A6. Die Wolkenbedeckung wird dabei in die Referenzklassen ‚wolkenlos' mit einem Gesamtbedeckungsgrad 0/8, ‚leicht bewölkt' mit einem Gesamtbedeckungsgrad von 1/8 bis 3/8, ‚wolkig' mit einem Gesamtbedeckungsgrad 4/8 bis 6/8, ‚stark bewölkt' mit einem Gesamtbedeckungsgrad 7/8 und ‚bedeckt' mit einem Gesamtbedeckungsgrad 8/8 eingeordnet. Bei der Schneeerkennung werden die Referenzklassen aus Kapitel 3.4.1 verwendet.
Darüber hinaus ist mit Anforderung A9 ein Programm zur Auswertung der Bilddaten beispielsweise auf Wetterstationen des DWD zu entwickeln.

Tabelle 4.1: Hauptanforderungen des Projektes

Nr.	Prio.	Anforderung	Beschreibung
A1	erf.	Schneeerkennung	semantische Segmentierung von Bodenkameraaufnahmen in drei Klassen (schneebedeckter Boden, unbedeckter Boden und Hintergrund)
A2	erf.	Wolkenerkennung	semantische Segmentierung von Bodenkameraaufnahmen des oberen Halbraumes in drei Klassen (wolkenbedeckter Himmel, unbedeckter Himmel und Hintergrund)
A3	erf.	Bedeckungsgrad	Bestimmung des Bedeckungsgrades für A1 und A2 in Prozent
A4	erf.	Klassifizierung (Schnee, 5 Klassen)	Zuordnung der Aufnahmen in die fünf Referenzklassen des Schneebedeckungsgrades
A5	erf.	Klassifizierung (Schnee, 11 Klassen)	Zuordnung der Aufnahmen in die elf Referenzklassen des Schneebedeckungsgrades
A6	erf.	Klassifizierung (Wolken, 5 Klassen)	Zuordnung der Aufnahmen in die fünf Referenzklassen des Wolkenbedeckungsgrades
A7	erf.	Netzimplementierung	Softwarelösung zum Trainieren und Testen künstlicher neuronaler Netze zur Erfüllung von A1 und A2
A8	erf.	Datenaufbereitung	effiziente Erstellung geeigneter Trainings-, Test- und Validierungsdaten
A9	erf.	Anwendung für Endnutzer*innen	Softwarelösung zur Anwendung der trainierten neuronalen Netze aus A7 durch die Endnutzer*innen

Nachfolgend werden die Qualitätsanforderungen an die Schnee- und Wolkenerkennung in Abschnitt 4.2 sowie die Anforderungen an die Softwarelösungen in Abschnitt 4.3 genauer aufgeschlüsselt.

4.2 Qualitätsanforderungen

In diesem Abschnitt werden die zu erreichenden Qualitätsansprüche bei der Erfüllung der Anforderungen A1 bis A6 aufgezeigt. Diese werden zusammen mit der Definition der Eingangsbilder in Tabelle 4.2 dargestellt. Auf Basis dieser Eingangsbilder soll die zu entwickelnde Software aus A9 eine Ausgabe generieren.

Tabelle 4.2: Qualitätsanforderungen

Nr.	Prio.	Anforderung	Beschreibung
A10	erf.	variable Bildgröße	Verarbeitung von Bilddaten variabler Größe für A1 und A2 (Referenzgröße: 320×240 bis 640×480 Pixel)
A11	opt.	Bildverzerrung	geeignete Berücksichtigung von Verzerrungs- und Perspektivfehlern entsprechend der Art und Position der jeweiligen Kamera
A12	opt.	Genauigkeit	Klassifizierung des Schnee- bzw. Wolkenbedeckungsgrades mit MAE ≤ 10 % und Acc ≥ 90 %
A13	erf.	Geschwindigkeit	Berechnungsdauer[8] des Schnee- bzw. Wolkenbedeckungsgrades für Eingangsbilder aus A10 $\leq 200\ ms$
A14	erf.	Tag/Nacht	Erfüllung der Qualitätsanforderungen A12 und A13 bei Tag und Nacht

4.3 Softwareanforderungen

Ein weiteres Anforderungsfeld sind technische Anforderungen an die zu entwickelden Softwarelösungen (siehe Tab. 4.3). Hierbei handelt es sich um allgemein gültige Anforderungen an jegliche Software, die für die Erfüllung der Anforderungen aus Tabelle 4.1 entwickelt wird.

[8] Ermittelt mit einem PC der Leistungsklasse Intel i7 oder vergleichbar.

Tabelle 4.3: Anforderungen an die Software-Implementierungen

Nr.	Prio.	Anforderung	Beschreibung
A15	erf.	Kompatibilität	Entwicklung der Softwarelösungen aus 4.1 in der Programmiersprache Python mit Kompatibilität zum Betriebssystem Linux
A16	erf.	Bibliotheken	Nutzung lizenzfreier Software u.a. OpenCV[9], Keras[10] und Tensorflow[11]
A17	opt.	Parametrierung	Parametrierbarkeit von Netz-Architekturen, Hyperparametern, sowie user- und projektspezifischer Daten

Abschließend wird in Tabelle 4.4 die Anwendung für Endnutzer*innen aus Anforderung A9 genauer betrachtet.

Tabelle 4.4: Anforderungen an die Anwendung für Endnutzer*innen aus A9

Nr.	Prio.	Anforderung	Beschreibung
A18	erf.	Ausgabeformat	Ausgabe des Dateinamen und der Ergebnisse von A3 bis A6 in csv-Datei
A19	erf.	Portierung	kamera- und szenerieunabhängige Implementierung
A20	opt.	Datenspeicherung	optionales Speichern der prädizierten semantisch segmentierten Bilder
A21	opt.	Datenverarbeitung	Automatisches Einlesen der zu verarbeitenden Aufnahmen mittels Watchdog oder Einlesen aller Aufnahmen eines Verzeichnisses
A22	opt.	Maskierung	optionales Maskieren des für die Berechnung irrelevanten Bildbereiches

[9] https://opencv.org .
[10] https://keras.io .
[11] https://www.tensorflow.org .

5 Konzeptionierung

In diesem Kapitel werden auf Grundlage der Anforderungen aus Kapitel 4 die am besten geeigneten Lösungsansätze hergeleitet, indem alternative Lösungswege verglichen und bewertet werden. Als Bewertungsmetrik wird die Notation ‚+' für positiv, ‚-' für negativ und ‚o' für neutral herangezogen. In der Konzeptionierung enthalten sind Themen zur Entwicklungsumgebung und genutzten Dateiformaten, die Datenaufbereitung zur Erstellung der Trainings- Validierungs- und Testdaten, Konzepte zur Optimierung und Validierung der Ergebnisse, Architekturvergleiche, Methoden für den Domain Transfer sowie die Konzepte speziell für die Schnee- und Wolkenerkennung. Abschließend wird die Klassifizierung nicht nutzbarer Bilddaten behandelt.

5.1 Entwicklungsumgebung

Bezogen auf die Entwicklungsumgebung gibt es mit den Anforderungen A15 und A16 feste Vorgaben an die Programmiersprache, die Betriebssystemkompatibilität und die zu nutzenden Bibliotheken. Für die Parametrierung der Netz-Architekturen, Hyperparameter, sowie Daten zu Projekten und Nutzer*innen aus A17 gibt es jedoch mehrere mögliche Ansätze. In dieser Arbeit werden drei verschiedene Möglichkeiten der Parametrierung gegenübergestellt: Parametrierung direkt im Code, Parametrierung in einer csv-Datei und Parametrierung in einer json-Datei (siehe Tab. 5.1).

Tabelle 5.1: Bewertung der Parametrierungsoptionen

Bewertungskriterium	**Code**	**csv-Datei**	**json-Datei**
Aufwand für Datenzugriff	+	-	+
Änderungsaufwand durch Endnutzer*innen	-	o	+
Dokumentationsaufwand	o	+	+

Der Aufwand für den Zugriff auf die Parameter durch den Programmcode ist bei der Nutzung der json-Datei im Gegensatz zur csv-Datei vergleichsweise einfach, da nach dem Einlesen der json-Datei direkt über die Parameternamen auf die Daten zugegriffen werden kann. Der Zugriff auf die Daten in einer csv-Datei wird bei einer größeren Anzahl von Parametern und hierarchisch aufgebauten Parametersätzen kompliziert.
Der Änderungsaufwand ist im Code ähnlich aufwendig wie bei der Nutzung der json-Datei, setzt jedoch bei der Änderung der Parameter Grundkenntnisse in der Programmiersprache Python voraus. Eine csv-Datei ist in dieser Hinsicht einfacher anzupassen, wird jedoch bei großen, hierarchisch aufgebauten Parametersätzen sehr unübersichtlich, was den Änderungsaufwand für die Endanwender*innen erhöht. Json-Dateien sind dagegen in Quellcode-Editoren einfach anzupassen und werden auch bei hierarchisch aufgebauten Parametersätzen übersichtlich dargestellt.
Der Dokumentationsaufwand ist bei der Nutzung von csv- und json-Dateien geringer als bei Parametrierungen direkt im Code, da diese Dateien direkt mit den Modellen archiviert werden können. Auf Basis dieser Bewertungskriterien wird für Parametrierungsaufgaben in dieser Arbeit das json-Dateiformat gewählt.

5.2 Datenaufbereitung

Nachfolgend wird eine geeignete Vorgehensweise bei der Erstellung von Trainings-, Validierungs- und Testdaten aus Anforderung A8 mit variabler Bildgröße nach Anforderung A10 erarbeitet. Dabei werden zwei Lösungsansätze für dieses Problem gegenübergestellt. Zunächst wird die Nutzbarkeit eines frei verfügbaren ‚Region-Growing' Algorithmus[12], sowie eines frei verfügbaren Programms zur manuellen Erstellung von Labelmaps über Bereichsmarkierung namens ‚Labelme'[13] gegenübergestellt (siehe Tab. 5.2). Abschließend wird die Entwicklung eines eigenen Programms diskutiert.

[12] https://github.com/Panchamy/RegionGrowing/blob/master/RegionGrowing.py (zul. aufgerufen am 14.03.2022).
[13] https://pypi.org/project/labelme/ .

Tabelle 5.2: Bewertung der Programmoptionen

Bewertungskriterium	**Region-Growing**	**Labelme**
Datenmanagement	-	-
Anpassbarkeit	+	-
Verwendbarkeit von Masken	-	-
Teilautomatisierung	o	-
manuelle Nachbearbeitung	-	+
flexible Klassenanzahl	-	+
Bedienbarkeit	-	+

Nach Tabelle 5.2 gibt es bei beiden Ansätzen mehrere Nachteile. Zwar schneidet das Programm Labelme insgesamt besser ab, jedoch fehlen auch hier wichtige Funktionen. Keines der Programme bietet die Möglichkeit eines Datenmanagements. Es werden keine Informationen zu den bereits erstellten Labelmaps gespeichert, was die Übersicht über die Trainings-, Validierungs- und Testdaten sowie die spätere Erstellung von Referenzdatensätzen zu Testzwecken erschwert. Darüber hinaus bietet keines der Programme die Möglichkeit der Verwendung von Masken, beispielsweise für die Klasse ‚Hintergrund', durch welche die Erstellung von Labelmaps für Bilder der selben Kameraposition deutlich beschleunigt wird. Ein Vorteil des Region-Growing Algorithmus ist, dass dieser im Gegensatz zu Labelme kein in sich geschlossenes Programm ist, wodurch sich der Code anpassen und erweitern lässt. Außerdem bietet der Region-Growing Algorithmus durch das verwendete Verfahren einen gewissen Grad der Automatisierung, da bei Auswählen eines Pixels im Bild benachbarte Pixel mit einem ähnlichen Grauwert automatisch klassifiziert werden. Labelme bietet eine solche Funktion nicht, da in diesem Programm die Pixel über manuelle Bereichsmarkierungen klassifiziert werden. Dafür ist eine manuelle Nachbearbeitung der Klassifizierung nur im Programm Labelme möglich. Ein weiteres Problem des Region-Growing Algorithmus ist die Klassenanzahl. Ohne eine Codeanpassung kann in diesem Fall während der Klassifizierung nur eine Klasse genutzt werden. Die Bedienbarkeit des Programms Labelme hebt sich durch die grafische Oberfläche und die Möglichkeit des Zoomens deutlich von der des Region-Growing Algorithmus ab, welcher keinerlei Bedienungshilfen enthält.
Die Entwicklung eines eigenen Programms zur Erstellung der Trainings-, Validierungs- und Testdaten bietet vor dem Hintergrund der Ergebnisse aus Tabelle 5.2 einige Vorteile.

Die mögliche Einführung eines Datenmanagements ist dabei gerade für große Datensätze sehr wichtig. So können die Daten für das Training der künstlichen neuronalen Netze deutlich besser verwaltet und Datensätze für verschiedene Tests leichter wiederverwendet werden. Außerdem ermöglicht ein eigens entwickeltes Programm die Verwendung von Masken. Des Weiteren können die Vorteile beider vorhandener Lösungen in einem neu entwickelten Programm sinnvoll kombiniert werden. So kann eine Teilautomatisierung der Pixel-Klassifizierung über die Grauwerte der Bilder mit der Möglichkeit der manuellen Nachbearbeitung der Klassifizierung verbunden werden. Zudem können die für diese Datenaufbereitungsaufgabe wichtigen Bedienelemente implementiert werden, die außerdem mögliche Eingabe- und Logikfehler bei der Datenaufbereitung verhindern.
Die hier genannten Aspekte überwiegen durch ihre Relevanz den Aufwand für die Entwicklung und Implementierung eines solchen Programms, weshalb die Datenaufbereitung durch ein eigens erstelltes Programm im weiteren Verlauf präferiert wird.

5.3 Optimierung und Validierung

In diesem Abschnitt werden die für die Optimierung und Validierung zu verwendenden Parameter und Funktionen ausgearbeitet. Eingangs erfolgt eine Analyse möglicher Bewertungskriterien. Anschließend wird ein Konzept zur Auswahl geeigneter Optimizer in Kombination mit Loss-Functions sowie der Batchsize und der Learning-Rate behandelt.

5.3.1 Auswertungsparameter

Eine aussagekräftige Ergebnisdarstellung erfordert eine geeignete Wahl von Referenzparametern, die für die Tests herangezogen werden. Eine Auswahl möglicher Parameter ist in Tabelle 5.3 dargestellt.

Tabelle 5.3: Bewertung der Auswertungsparameter

Bewertungskriterium	**Acc**	**mIoU**	**F1-Score**
Invarianz gegenüber Klassenverteilung	-	+	+
Fehlergewichtung der Segmentierung	-	o	o
Übertragbarkeit auf Klassifizierung	+	o	o

Wie Tabelle 5.3 zu entnehmen, schneiden die Parameter mIoU und F1-Score gleichermaßen ab. Dies ist durch die sehr ähnliche Berechnung zu begründen. Durch die höhere Gewichtung der korrekt segmentierten Datenpunkte belohnt der F1-Score diese stärker als der mIoU. Schlussendlich sind beide Parameter robust gegenüber ungleicher Klassenverteilung, da jede Klasse für sich betrachtet und anschließend der Mittelwert berechnet wird. Entsprechend schwer fallen Fehler bei Klassen mit geringer Verfügbarkeit im Testbild ins Gewicht. Für die Validierung der Segmentierungsergebnisse wird der F1-Score gewählt, um einzelne Pixelfehler bei Klassen mit sehr geringer Verfügbarkeit nicht derart stark zu gewichten, wie es bei dem mIoU der Fall ist. Dennoch wird die Robustheit gegenüber ungleicher Klassenverteilung gewährleistet.
Die Pixelwise Accuracy stellt hingegen die Anzahl der korrekt klassifizierten Datenpunkte zur Gesamtanzahl der Datenpunkte ins Verhältnis. Fehler bei Klassen, die im Vergleich zum Gesamtbild nur gering vertreten sind, haben daher nur sehr geringe Auswirkungen auf die Pixelwise Accuracy, auch wenn diese bei der Klassifizierung des Bedeckungsgrades einen großen Einfluss haben sollten. Da die verfügbaren Datensätze ebenfalls Bilder mit annähernd homogener Klassenverteilung enthalten, wird die Pixelwise Accuracy zusätzlich zur Validierung der Architekturen verwendet.

5.3.2 Optimizer und Loss-Function

Sowohl für den Optimizer als auch für die Loss-Function sind diverse Optionen verfügbar. Um das bestmögliche Ergebnis für die Anforderungen A1 und A2 zu erzielen und dabei die Genauigkeit aus A12 zu erreichen, werden verschiedene Kombinationen betrachtet. Nach [Yaqub u. a., 2020] eignet sich der Adam Optimizer besonders gut für Segmentierungsaufgaben. Dieser soll inklusive der Erweiterung auf den Nadam Optimizer (vgl. [Dozat, 2016]) in die Testszenarien einbezogen werden.
Eine Auswahl von Loss-Functions wird in [Jadon, 2020] analysiert. Unter der Annahme, dass innerhalb der Aufnahmen eine große Varianz der Schnee- bzw. Wolkenverteilung vorherrscht, sich diese jedoch über einen Mini-Batch relativiert, werden der Dice-Loss (auch F1-Loss) und der Cross-Entropy-Loss zur genaueren Betrachtung ausgewählt. Der F1-Loss basierend auf dem F1-Score eignet sich besonders bei ungleicher Klassenverteilung innerhalb der Trainingsdaten. Der Cross-Entropy-Loss hingegen eignet sich bei homogenen Klassenverteilungen. Als dritte Alternative wird der Log-Cosh-Dice-Loss nach [Jadon, 2020] gewählt. Dieser ist eine Erweiterung des Dice-Loss und verbessert nach [Jadon, 2020] die Genauigkeit bei unausgeglichener Klassenverteilung zusätzlich.

Für die Auswahl werden alle Kombinationen der ausgewählten Optimizer und Loss-Functions mit dem selben Datensatz und konstanter Learning-Rate sowie Batchsize mit einer U-Net Architektur nach [Ronneberger u. a., 2015] getestet.

5.3.3 Batchsize und Learning-Rate

Zusätzlich zum Optimizer und der Loss-Function sind die Batchsize und die Learning-Rate zu definieren. Diese Parameter sind nach [Kandel u. Castelli, 2020] stark voneinander abhängig. Unter Berücksichtigung der erhöhten Generalisierungsleistung bei einer kleinen Batchsize (vgl. [Goodfellow u. a., 2016]) wird zunächst die Batchsize über einen Test von $\mathcal{B} = 1$ bis $\mathcal{B} = 50$ bei konstanter Learning-Rate eingestellt.
Ein anschließender Test mit gestaffelter Learning-Rate und der zuvor festgelegten Batchsize soll die Optimierung der Batchsize auf die voreingestellte Learning-Rate validieren. Die Spanne der zu validierenden Lerning-Rate wird auf 10^{-1} bis 10^{-6} festgesetzt. In diesem Bereich sind nach [Yaqub u. a., 2020] die gravierensten Änderungen der Genauigkeit zu verzeichnen.
Unabhängig davon wird eine Strategie zur Reduzierung der Learning-Rate im Trainingsverlauf vorgesehen. Diese soll das Training bei Instabilität und Stagnierung unterstützen. Dabei gibt es die Möglichkeit, eine variable, automatisierte Anpassung der Learning-Rate vorzunehmen, welche den Trainingsverlauf überwacht und die Learning-Rate bei Bedarf anpasst. Die Alternative zu diesem Verfahren ist das Learning-Rate-Scheduling, bei dem die epochale Planung der Anpassungsschritte im Vorfeld festgelegt wird. In dieser Arbeit wird die variable, automatisierte Anpassung genutzt, da diese nur dann wirksam wird, wenn tatsächlich eine Stagnierung oder Instabilität des Trainings vorliegt und das Training nicht durch verfrühte oder verspätete Reduzierung der Learning-Rate negativ beeinflusst wird.

5.4 Auswahl von Architekturen

Für Anforderung A7 wird ein Programm zum Trainieren und Testen künstlicher neuronaler Netze entwickelt. Dieses Kapitel behandelt die Auswahl der geeigneten Netz-Architekturen, die im Rahmen dieser Arbeit trainiert und getestet werden. Dafür werden die Netz-Architekturen aus Kapitel 3 herangezogen und die Ergebnisse der dort

beschriebenen Ausarbeitungen gegenübergestellt. Da das Ziel dieser Arbeit die semantische Segmentierung ist, werden ausschließlich FCN genutzt. Diese erlauben gleichzeitig die Nutzung variabler Eingangsbildgrößen, wie es Anforderung A10 fordert.

In Tabelle 5.4 werden Accuracy und F1-Score des UX-Nets von [Komiyama u. a., 2018] mit denen eines U-Nets gleicher Parametrierung verglichen. Für den von [Komiyama u. a., 2018] verwendeten Datensatz ergibt sich eine deutliche Verbesserung des F1-Scores bei geringer Reduzierung der Accuracy durch die Nutzung des UX-Nets. Somit wird diese Netzarchitektur in der Gegenüberstellung aus 5.4 favorisiert.

Tabelle 5.4: Gegenüberstellung von U- und UX-Net (vgl. [Komiyama u. a., 2018])

Architektur	**Acc**	**F1-Score**
U-Net	98,1 %	81,5 %
UX-Net	97,7 %	90,8 %

Des Weiteren zeigt Tabelle 5.5 die Testergebnisse des um ein MFFM erweiterten U-Nets von [Li u. a., 2018] im Vergleich zu einem DeepLab-Algorithmus. Für den hier verwendeten Datensatz ist die U-Net Architektur inklusive eines MFFM in allen Bewertungskriterien führend, weshalb diese favorisiert wird.

Tabelle 5.5: Gegenüberstellung von DeepLab und U-Net inkl. MFFM (vgl. [Li u. a., 2018])

Architektur	**Acc**	**mIoU**	**F1-Score**
DeepLab	87,7 %	75,5 %	86,0 %
U-Net (MFFM)	95,0 %	89,5 %	94,5 %

Zuletzt wird in Tabelle 5.6 die DeepLab-basierte Architektur von [Hongcai u. a., 2019] mit einem DeepLab v3+ (vgl. [Chen u. a., 2018]) verglichen. Bei der Schnee- und Wolkenerkennung auf Basis eines Trainingsdatensatzes mit Satellitenbildern zeigt sich durch die modifizierte DeepLab Architektur von [Hongcai u. a., 2019] eine Verbesserung gegenüber dem DeepLab v3+, weshalb diese im weiteren Verlauf der Arbeit genutzt wird.

Tabelle 5.6: Gegenüberstellung von DeepLab v3+ und der modifizierten DeepLab Architektur von [Hongcai u. a., 2019]

Architektur	**Acc**	**mIoU**
DeepLab v3+	94,9 %	88,9 %
DeepLab (MFFM / CSAM)	95,1 %	89,3 %

Die drei hier ausgewählten Architekturen können alle potentiell die Anforderungen A1 und A2 für die semantische Segmentierung der Bilder für die anschließende Ermittlung der Bedeckungsgrade erfüllen. Zwei der drei ausgewählten Architekturen sind dabei auch für die Erkennung von Schnee beziehungsweise Wolken entwickelt (vgl. [Li u. a., 2018], [Hongcai u. a., 2019]). Da die Ergebnisse der drei ausgewählten Architekturen aufgrund der unterschiedlichen Testdatensätze nicht direkt vergleichbar sind, werden alle drei Architekturen implementiert und anhand der Datensätze aus Kapitel 3.4.1 trainiert und getestet.

Die oben genannten Architekturen werden um eine weitere Architektur ergänzt, welche im Rahmen dieser Arbeit auf Basis der Ergebnisse der bestehenden Architekturen konzeptioniert wird (siehe Abb. 5.1). Die Basis dieser Architektur bildet ein U-Net nach [Ronneberger u. a., 2015] mit der Erweiterung zum UX-Net nach [Komiyama u. a., 2018], um räumliche Informationen an den Decoder-Eingang und Feature-Informationen an den Decoder-Ausgang zu transportieren. Nach den Erkenntnissen von [Hongcai u. a., 2019] werden diese Verbindungen durch ein CAM und ein SAM unterstützt. Diese Bausteine sollen durch sinnvolle Gewichtung der Featuremaps, beziehungsweise der räumlichen Informationen das Segmentierungsergebnis verbessern. Darüber hinaus wird in der obersten Encoder-Decoder Verbindung ein CSAM verwendet, um die Informationen, die von dem Input-Layer direkt in den Output-Layer fließen, sinnvoll zu gewichten. Außerdem werden die Up-Sampling-Operationen im Decoder, wie von [Tong u. a., 2021] zur Verbesserung der Segmentierungsgenauigkeit empfohlen, durch Transposed Convolutions (learned Up-Sampling) ersetzt. Nachfolgend wird die hier erstellte Architektur als ‚KMTX-Net' bezeichnet.

Das KMTX-Net wird den zuvor ausgewählten Architekturen ebenfalls gegenübergestellt. Der moderate Implementierungsaufwand der verschiedenen Architekturen rechtfertigt diese Vorgehensweise.

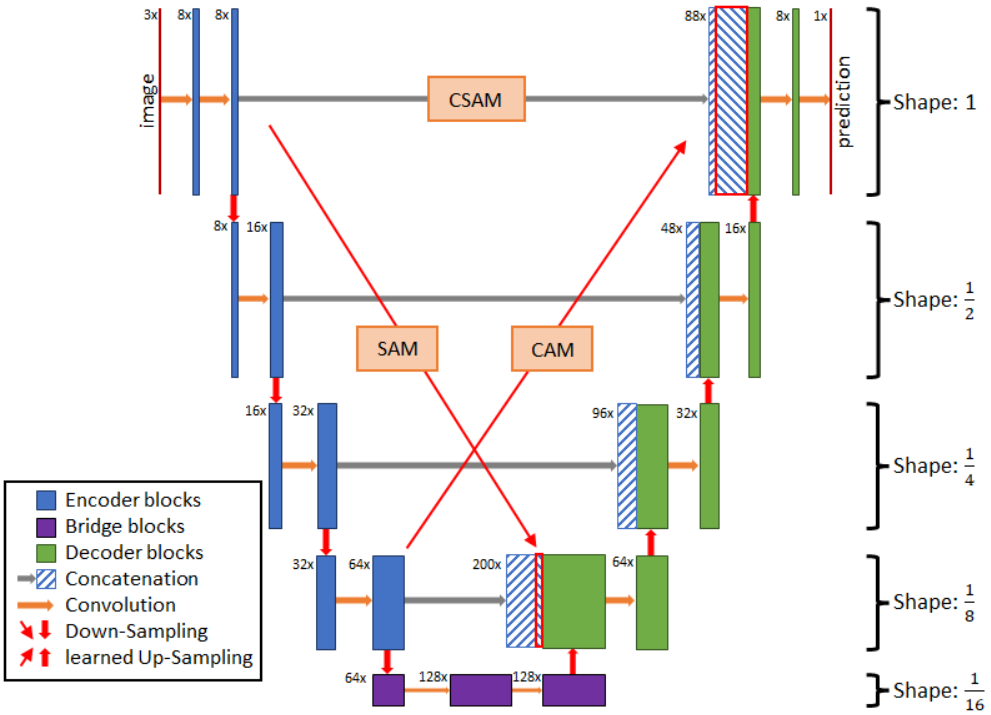


Abbildung 5.1: Beispielhafter Aufbau der KMTX-Architektur

5.5 Methoden des Domain Transfers

Ein weiteres relevantes Thema bei der Schnee- und Wolkenerkennung ist der Domain Transfer, wobei es darum geht, die zu erstellenden neuronalen Netze für verschiedene Domains nutzbar zu machen. In dieser Arbeit wird der Begriff Domain sowohl für Szenerien, als auch für Kameras genutzt. Ein Domain Transfer kann hier folglich sowohl ein Transfer auf neue Szenerien wie beispielsweise neue Wetterstationen, als auch ein Transfer auf eine andere Kamera in der selben Szenerie sein. Ein Kamerawechsel stellt durch unterschiedliche Kameraeigenschaften wie beispielsweise Kontrastverhältnis, Bildformat oder Verzerrung ebenfalls einen Domain Transfer dar.
Es gibt verschiedene Herangehensweisen für diese Problemstellung. Zunächst kann ein für eine Domain trainiertes Netz mit Trainingsdaten einer neuen Domain weitertrainiert werden. Außerdem kann ein Netz initial mit Trainingsdaten verschiedener Domains trainiert werden. Diesen Verfahren gegenüber stehen die DATL-Verfahren, welche von [Scannell u. a., 2020] und [Brion u. a., 2021] beschrieben werden. Von den DATL-Verfahren wird für

diese Arbeit der Domain Transfer auf Merkmalsebene gewählt, da [Scannell u. a., 2020] mit einem so adaptierten U-Net bei unbekannten Domains Verbesserungen im F1-Score um bis zu 10 % gegenüber einem vergleichbaren nicht angepassten U-Net erzielen. Überdies lassen sich die für diese Arbeit ausgewählten Netz-Architekturen dementsprechend anpassen, wogegen bei Image- und Label-basierten DATL-Verfahren zusätzlich GAN benötigt werden.
Das merkmalsbasierte DATL wird in dieser Arbeit dem Training mit Daten aller Domains ohne DATL, sowie dem Weitertrainieren von Netzen, die mit einer Domain vortrainiert werden, gegenübergestellt. Dabei werden Segmentierungsergebnisse sowohl auf Domains, die für das Training genutzt werden, als auch auf unbekannten Domains ausgewertet. Der Domain-Klassifikator der Netzarchitektur von [Scannell u. a., 2020] wird zur Erfüllung von Anforderung A10 mit den Erkenntnissen von [He u. a., 2014] angepasst, um die Größe der Eingangsbilder variabel zu halten und Informationsverlust durch Veränderung der Bildgröße vorzubeugen. Hierzu werden durch Faltungsoperationen Features extrahiert, die mittels Global-Pooling zu einem Feature-Vektor zusammengefasst werden. Dieser wird dem Dense Layer für die Klassifizierung bereitgestellt (siehe Abb. 5.2). Das nach der Architekturanalyse ausgewählte neuronale Netz ist für das DATL um den Domain-Klassifikator zu erweitern.

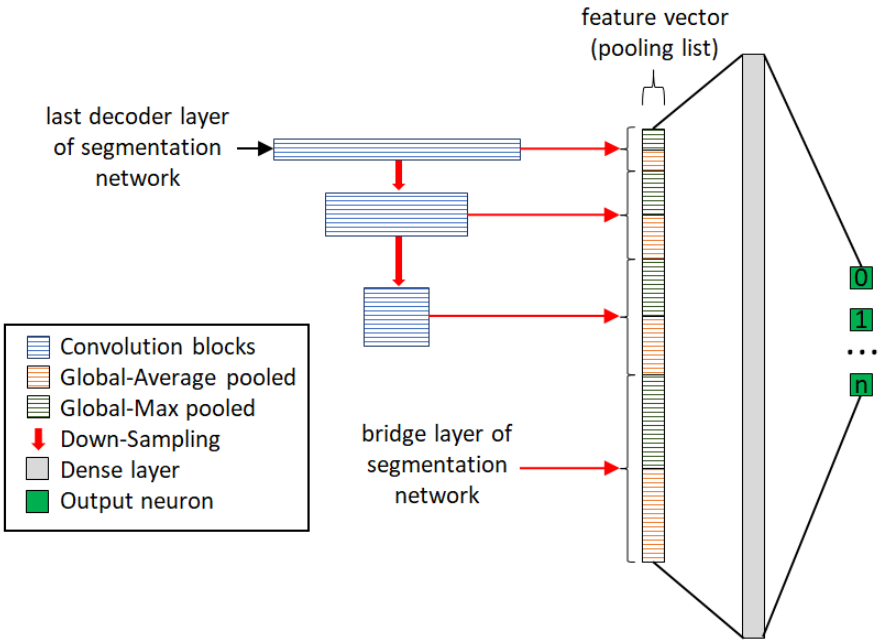


Abbildung 5.2: Beispielhafter Aufbau des Domain-Klassifikators

5.6 Schneeerkennung

In diesem Abschnitt wird das Konzept der Schneeerkennung erarbeitet. Auf Basis der Anforderung A1 ist das Ziel, schneebedeckte Flächen in Kameraaufnahmen semantisch zu segmentieren, um anschließend den Schneebedeckungsgrad zu ermitteln. Dafür wird im Folgenden der Umgang mit Bildaufnahmen bei Tag und Nacht konzipiert. Des Weiteren wird ein Konzept für eine möglichst standortunabhängige Schneeerkennung aufgestellt. Abschließend wird die Berechnung des Schneebedeckungsgrades sowie die Zuordnung der Kameraaufnahmen in die Referenzklassen definiert.

5.6.1 Verarbeitung von Tag- und Nachaufnahmen

In diesem Abschnitt wird die in Anforderung A14 geforderte Verarbeitung von Tag- und Nachtaufnahmen bei der Schneerkennung definiert. Die in Kapitel 3.4.1 eingeführten Datensätze enthalten die hier zu verarbeitenden Bilder. Während der Datensatz des Flughafenvorfeldes für Tag und Nacht ausschließlich Farbbilder enthält und durch die Ausleuchtung nahezu kein Unterschied ersichtlich ist, besteht der Datensatz der Wasserkuppe aus Farbbildern am Tag und IR-Bildern bei Nacht (siehe Abb. 5.3).

Abbildung 5.3: Farbaufnahme der Wasserkuppe um 16:34 Uhr (links) und IR-Aufnahme der Wasserkuppe um 17:04 (rechts) des selben Tages

In Tabelle 5.7 sind drei mögliche Ansätze zur Verarbeitung der verschiedenen Formate aufgezeigt. Die Segmentierung der Farb- und IR-Bilder kann getrennt oder gemeinsam erfolgen. Außerdem ist eine künstliche Bildeinfärbung der IR-Aufnahmen nach [Limmer u. Lensch, 2016] möglich.

Tabelle 5.7: Gegenüberstellung von Segementierungsstrategien für IR-Aufnahmen

Bewertungskriterium	**getrennte Segmentierung**	**gemeinsame Segmentierung**	**Bild-einfärbung**
Implementierungsaufwand	+	+	-
Domain Transfer	o	o	-
Trainingsdatenerstellung	-	-	o

Nach Tabelle 5.7 ist der Implementierungsaufwand sowohl für die getrennte als auch für die gemeinsame Segmentierung gering, da sich die Netz-Architektur diesbezüglich nicht ändert und die Definition des Netzeingangs in die Konfigurationsdatei ausgelagert ist. Bei der Bildeinfärbung hingegen ist eine zusätzliche Netz-Architektur zu implementieren. Des Weiteren wird auch der Domain Transfer für die Bildeinfärbung als aufwendiger gegenüber den reinen Segmentierungsaufgaben bewertet. Dies fußt auf der Tatsache, dass hierbei sowohl das neuronale Netz für die semantische Segmentierung als auch jenes für die Bildeinfärbung auf die neue Domain adaptiert oder bereits bei der Erstellung domainunabhängig konzipiert werden muss.
Das letzte Bewertungskriterium betrifft die Anzahl der benötigten Trainingsdaten. Sowohl die getrennte als auch die gemeinsame Segmentierung benötigen jeweils einen Trainingsdatensatz für Farb- und IR-Bilder. Bei der künstlichen Bildeinfärbung genügt ein Trainingsdatensatz mit Farbbildern für die semantische Segmentierung, da die IR-Bilder in Farbbilder übersetzt und somit als selbe Domain gehandhabt werden können. Das Training des neuronalen Netzes zur Einfärbung kann durch die Interpretation der Zeitstempel der Trainingsbilder erfolgen. In Abbildung 5.3 ist die Schneedecke innerhalb von 30 Minuten nahezu unverändert, sodass in einer definierten Zeitspanne um den Umschaltzeitpunkt von Farb- auf IR-Bilder die Farbbilder als Ground Truth der IR-Bilder beim Training des neuronalen Netzes zur Bildeinfärbung verwendet werden können.
Durch den hohen Implementierungsaufwand sowie den Zusatzaufwand für den Domain Transfer wird die Bildeinfärbung für die weitere Umsetzung ausgeschlossen. Weiterhin ist bereits bei einem zeitlichen Versatz von 30 Minuten (siehe Abb. 5.3), durch das Umschalten auf die IR-Kamera, ein deutlicher Unterschied bei der Bildaufnahme durch den Wegfall der Farbinformationen zu verzeichnen. Dieser wird bei größeren Zeitunterschieden zunehmend gravierender, sodass Tag- und Nachtaufnahmen als zwei Domains behandelt werden (siehe Abb. 5.4). Dies spricht gegen eine gemeinsame Verarbeitung. Um

die Schneerkennung dennoch robust gegenüber Farb- und IR-Aufnahmen zu gestalten, wird nachfolgend die getrennte Segmentierung forciert.

Abbildung 5.4: Farbaufnahme der Wasserkuppe um 15:34 Uhr (links) und IR-Aufnahme der Wasserkuppe um 19:34 (rechts) des selben Tages

5.6.2 Standortunabhängige Schneeerkennung

Für die standortunabhängige Verwendung der Schneeerkennung werden die in Kapitel 5.5 definierten Methoden des Domain Transfers angewendet und die Resultate verglichen. Um die Robustheit des Domain Transfers noch weiter zu steigern, werden im Folgenden Randbedingungen für das Training der neuroalen Netze bei dem Domain Transfer definiert.
Das DATL stützt sich auf die Extraktion von domaininvarianten Merkmalen. Um möglichst viele dieser Merkmale zu gewinnen, wird ein Trainingsdurchlauf mit allen verfügbaren Domains initiiert. Eine Herausforderung dabei ist die Separation von Merkmalen für Flächen, auf denen Schnee liegen könnte und denen, wo dies nicht möglich ist. Wird beispielsweise breites Buschwerk von oben abgelichtet, kann dies mit Schnee bedeckt werden. Wird anschließend das selbe Buschwerk seitlich abgelichetet bietet dieser Teil keinen Raum für Schneeflächen. Die extrahierten Merkmale werden sich dennoch sehr ähneln. Lokale Merkmale spielen somit eine erheblich Rolle. Eine Lösung für das Problem bietet die Reduzierung der Klassenzahl von drei auf zwei Klassen. Hierbei wird zwischen den Klassen ‚schneebedeckter Boden' und ‚unbedeckter Boden' unterschieden. Bei dieser Art von Training muss das neuronale Netz lediglich den Schnee erkennen. Hintergrundbereiche müssen in diesem Fall anhand von Masken segmentiert werden. Eben dieser Ansatz wird sowohl für den Domain Transfer als auch für alleinstehende Domains evaluiert.

Die Nutzung von Augmentations zur Bildmanipulation soll diesen Prozess zusätzlich unterstützen. Dies geschieht in zwei Iterationen. Die erste Iteration enthält Augmentations, bei deren Verwendung das neuronale Netz ausschließlich Eingangsbilder erhält, die auch ein realistisches Eingangsszenario darstellen. Die Augmentations *Vertical-Flip*, bei der das Bild über die vertikale Achse gespiegelt wird, *Translate*, bei der das Bild horizontal verschoben wird und *Rotate*, bei der das Bild um den Mittelpunkt rotiert wird, erfüllen diese Vorraussetzung, da durch horizontale Bewegung oder Neupositionierung der Kamera auf dem Messfeld eben diese Szenarien entstehen können. Die Rotation wird auf Winkel bis 15 Grad eingeschränkt, um eine nicht optimal ausgerichtete Kamera zu simulieren.
Die zweite Iteration manipuliert die Eingangsbilder stärker. Hier wird den zuvor verwendeten Augmentations der *Horizontal-Flip* hinzugefügt, bei dem das Bild über die horizontale Achse gespiegelt wird. Darüber hinaus wird eine Bildzerstückelung, bei der das Bild mit Überlappung zerschnitten wird, verwendet. Weiterhin werden bei der Rotation Winkel größer als 15 Grad zugelassen. Diese Art der Bildmanipulation kann durch die Reduzierung auf zwei Klassen erfolgen, da in diesem Fall die Separation von Hintergrund und unbedecktem Boden und somit die lokale Anordnung nicht relevant ist. Abschließend werden diese Ergebnisse jenen ohne Verwendung von Augmentations gegenübergestellt.

5.6.3 Ermittlung des Schneebedeckungsgrades

Für die Ermittlung des Schneebedeckungsgrades aus Anforderung A3 werden die zuvor semantisch segmentierten Kameraaufnahmen verwendet. Hierbei wird die schneebedeckte Fläche zur gesamten Bildfläche abzüglich des Hintergrundes ins Verhältnis gesetzt. Um diese Flächen nach Anforderung A11 möglichst genau berechnen zu können, werden in Tabelle 5.8 zwei Optionen aufgezeigt, die perspektivische Verzerrungen durch die Kameraposition sowie deren Ausrichtung und somit Unterschiede im relativen Maßstab von nahen und fernen Bildbereichen berücksichtigen. Diese werden der Nichtnutzung jeglicher Verzerrungskorrekturen gegenübergestellt.

Tabelle 5.8: Berücksichtigung perspektivischer Verzerrung bei der Berechnung des Schneebedeckungsgrades

Bewertungskriterium	**keine Beachtung**	**Pixel-gewichtung**	**geometrische Rektifizierung**[14]
Implementierungsaufwand	+	o	-
Domain Transfer	+	o	-
Fehler bei der Berechnung des Bedeckungsgrades	-	o	+

Der Implementierungsaufwand ist am geringsten, sofern keine Beachtung von perspektivischer Verzerrung erfolgt. Hier müssen lediglich die Pixel der jeweiligen Klassen gezählt und ins Verhältnis gesetzt werden, da alle Pixel als gleichwertig in Bezug auf die abgebildete Fläche angenommen werden. Bei der Pixelgewichtung hingegen unterscheiden sich die Pixelwertigkeiten je nach Position des Pixels im Bild. Bedingt durch die Bildtiefe wird Pixeln, die weiter von der Kamera entfernte Bereiche abbilden, eine höhere Gewichtung zugeteilt als Pixeln im Vordergund. Durch Berücksichtigung der Verzerrung lediglich in der vertikalen Bildachse ist der abzuleistende Implementierungsaufwand moderat gegenüber der vollständigen geometrischen Rektifizierung, welche nach [Kraus, 2004] Passbeziehungsweise Referenzpunkte für die Bildtransformation benötigt. Diese müssen zuvor definiert und aus den Bildern extrahiert werden.
Die Aufwände für den Domain Transfer korrelieren mit den jeweiligen Implementierungsaufwänden. Während bei der Nichtberücksichtigung der perspektivischen Verzerrung keine Aufwände entstehen, muss bei der Pixelgewichtung eine angepasste Referenzgewichtung erstellt werden. Die geometrischen Rektifizierung fordert eine erneute Definition von Referenzpunkten, die geeignet extrahiert werden müssen.
Gegensätzlich verhält sich der zu erwartende Fehler bei der Berechnung des Bedeckungsgrades. Während bei einer optimalen geometrischen Rektifizierung alle Pixel die selbe Bildfläche darstellen, werden bei der Pixelgewichtung die Gewichte linear entlang der vertikalen Bildachse angenähert. Dies stellt eine Vereinfachung der in [Schreer, 2005] definierten Approximation der perspektivischen Transformation dar, da Verzerrungen in der horizontalen Bildachse vernachlässigt werden.

[14] Kompensation von Verzerrungen, die auf Grund von Bildneigungen und Geländehöhenunterschieden zustandekommen.

Nach Analyse der Bewertungskriterien aus Tabelle 5.8 werden bei der Verwendung der geometrischen Rektifizierung die genauesten Ergebnisse für die Berechnung des Bedeckungsgrades durch die Berücksichtigung der Bildverzerrung aus Anforderung A11 erwartet. Im Gegensatz dazu steht die einfache Portierbarkeit auf neue Domains aus Anforderung A19, welche bei der geometrischen Rektifizierung nicht gegeben ist. Durch die Notwendigkeit der einfachen Portierbarkeit wird die Pixelgewichtung für die Ermittlung des Schneebedeckungsgrades im weiteren Verlauf forciert. Zusätzlich werden diese Ergebnisse denen bei Nichtbeachtung der perspektivischen Verzerrung gegenübergestellt. Hierbei wird der Einfluss der Berücksichtigung von perspektivischer Verzerrung bei den vorhandenen Aufnahmen evaluiert.
Die Einordnung der Aufnahmen anhand des zuvor berechneten Bedeckungsgrades in fünf beziehungsweise elf Referenzklassen (siehe Anf. A4, A5) soll eine Accuracy von mindestens 90 % erreichen (siehe Anf. A12). Dabei wird, je nach Szenerie, der für einen Beobachter relevante Bereich von 100 Metern (vgl. [DWD, 2021]) überschritten, um die Accuracy des gesamten Bildbereiches zu ermitteln. Zusätzlich wird untersucht, ob die Accuracy durch die Nutzung des 2-Klassensystems mit Hintergrundmaskierung verbessert werden kann.

5.7 Wolkenerkennung

Das folgende Kapitel befasst sich mit der Konzeptionierung der in Anforderung A2 und A3 geforderten semantischen Segmentierung von Wolken in Kameraaufnahmen des oberen Halbraums, sowie der anschließenden Ermittlung des Wolkenbedeckungsgrades. Die Methoden und Erkenntnisse aus der Konzeptionierung der Schneeerkennung werden dabei berücksichtigt. Zunächst wird eine Wolkendefinition festgelegt und der Beobachtungshorizont definiert. Anschließend wird diskutiert, wie sich ein Training von neuronalen Netzen für die Wolkenerkennung mit möglichst geringer Anzahl an Trainingsdaten realisieren lässt. Der Domain Transfer wird ebenfalls für die Wolkenerkennung in Betracht gezogen.
Für die Wolkenerkennung stehen für diese Arbeit zwei Bilddatensätze des selben Standorts unter Verwendung verschiedener Kameras zur Verfügung. Bei den Kameras handelt es sich um eine Mobotix Q26 und eine Vivotek FE9381-EHV (siehe Abb. 5.5).

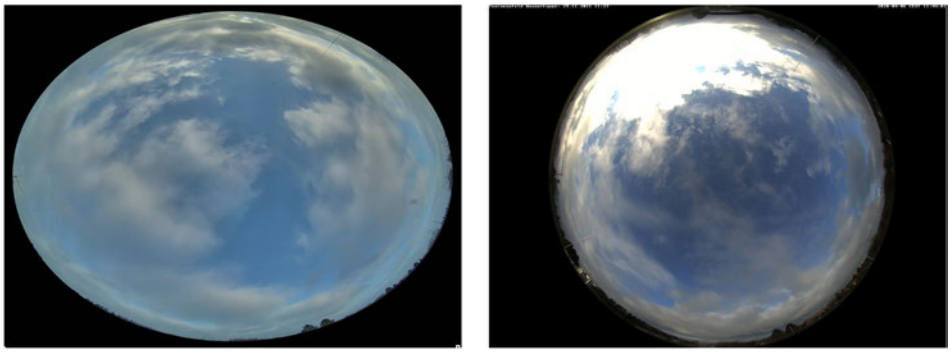


Abbildung 5.5: Wolkenaufnahmen Vivothek FE9381-EHV (links), Mobotix Q26 (rechts)

Nachtaufnahmen können bei diesem Datensätzen nicht in die Ermittlung des Bedeckungsgrades einbezogen werden, da diese lediglich dunkles Bildrauschen enthalten und keine Rückschlüsse auf die Wolkenbedeckung zulassen.

5.7.1 Wolkendefinition und Beobachtungshorizont

Die Wolkendefinition des DWD lautet nach [Kuner, 2015] wie folgt: „Eine Wolke ist eine sichtbare, in der Luft schwebende Anhäufung von atmosphärischen Kondensations- und Sublimationsprodukten des Wasserdampfes (Wassertröpfchen und/oder Eisteilchen), die den Erdboden im Gegensatz zum Nebel nicht berührt“. Demnach werden in dieser Arbeit nach Möglichkeit alle sichtbaren Wasserdampfpartikel in der Luft, die kein Bodennebel sind, als Wolke klassifiziert. Dabei wird keine Unterscheidung zwischen verschiedenen Wolkentypen oder Wolkenhöhen getroffen. Weiterhin werden laut [DWD, 1967] die horizontnahen Wolken in Bezug auf den Bedeckungsgrad überschätzt. Hier ist kein schwerwiegender Fehler bei der Ermittlung des Bedeckungsgrades zu erwarten, wenn der Beobachtungshorizont etwas angehoben wird. Da anhand der zur Verfügung gestellten Kameraaufnahmen nicht ermittelbar ist, inwiefern Wolkenlücken im horizontnahen Bereich gegebenenfalls verdeckt werden, werden in dieser Arbeit lediglich 90 % des Bildradius der Wolkenbilder während der Messungen betrachtet. Zudem wird die Möglichkeit geben, diesen Bereich bei der Wolkenbedeckungsmessung einzustellen.

5.7.2 Training mit geringer Trainingsdatenanzahl

Für die Wolkenerkennung wird untersucht, wie ein effektives Training mit einer verhältnismäßig geringen Anzahl an Trainingsdaten durchgeführt werden kann, um dadurch den Arbeitsaufwand bei der Datenaufbereitung möglichst gering zu halten. Eine Alternative zur Verwendung untrainierter neuronaler Netze ist das Weitertrainieren bereits bestehender Modelle, deren ursprüngliche Trainingsdaten Parallelen zu den neuen Trainingsdaten aufweisen. So kann das Training durch die bereits initialisierten Gewichte positiv beeinflusst werden. Wolken weisen in ihrer Form und Farbe häufig Ähnlichkeiten zu Schneeflecken auf (siehe Abb. 5.6). Dahingegen wird der Himmelsraum in den Schneebildern durchgehend als Hintergrund klassifiziert, während es bei der Wolkenerkennung die Unterteilung in unbedeckten und wolkenbedeckten Himmelsraum gibt. Wenn davon ausgegangen wird, dass die Initialgewichte für die Klassifizierung des Himmelsraumes als Hintergrund das Ergebnis nicht stärker negativ beeinflussen als völlig zufällige Initialgewichte, ist eine Verbesserung des Trainings durch die Nutzung von vortrainierten Modellen aus der Schneeerkennung denkbar und wird dementsprechend getestet.

Abbildung 5.6: Vergleich von Wolken (links) und Schneeflecken (rechts)

Darüber hinaus gibt es verschiedene Möglichkeiten der künstlichen Erzeugung weiterer Trainingsdaten, beispielsweise durch Bildzerstückelung mit Überlappung und Augmentations. [Komiyama u. a., 2018] nutzen diese Verfahren in Kombination, um aus fünf Bildern insgesamt 7344 Trainingsdaten zu erzeugen. Damit wird für den spezifischen Anwendungsfall eine Pixelwise Accuracy von über 98 % sowie eine Class Average Accuracy von über 90 % erreicht. Wolkenbilder des oberen Halbraums eignen sich zudem sehr gut für verschiedene Augmentations, da sie nach der Definition in dieser Arbeit keine fest definierten Formen, Ausprägungen, Positionen oder Orientierungen aufweisen. In diesem Fall werden bei der Bildzerstückelung, starken Rotationen oder Verkippen keine für die

Wolkenerkennung wichtigen Informationen eliminiert. Die Nutzung von Augmentations wie *Rotate* in verschiedenen Winkeln sowie *Horizontal-* und *Vertical-Flip* ist daher durchaus sinnvoll und unterstützt voraussichtlich die Generalisierungsleistung der neuronalen Netze. Die Bildzerstückelung befreit die Trainingsdaten zudem von störenden Kontextinformationen bedingt durch die runde beziehungsweise ovale Bildform. Dieses Verfahren setzt jedoch voraus, dass das neuronale Netz, wie in Anforderung A10 gefordert, eine variable Eingangsbildgröße unterstützt.

5.7.3 Domain Transfer in der Wolkenerkennung

Der Domain Transfer wird ebenfalls bei der Wolkenerkennung betrachtet. Dieser beschränkt sich in der vorliegenden Arbeit auf die Nutzung verschiedener Kameras. Der Transfer auf andere Standorte wird nicht betrachtet, da Hintergrundinformationen wie beispielsweise Gebäude meist eine geringe oder gar keine Rolle beim Domain Transfer spielen, sofern horizontnahe Bereiche von der Messung ausgenommen sind. Ändern kann sich dieser Sachverhalt, wenn die Kamera beispielsweise von hohen Gebäuden, Bäumen oder Bergen flankiert wird, die sich bis in den für die Messung relevanten Bereich erstrecken. Die vorhandenen Datensätze bilden solche Fälle jedoch nicht ab. Nach [Scannell u. a., 2020] sind Unterschiede bezüglich Bildrauschen und Konstrast verschiedener Kameras neben standortbezogenen Aspekten ebenfalls Probleme, die ein DATL rechtfertigen. In Abbildung 5.5 sind Kameraunterschiede bei Helligkeit und Kontrast bei der Darstellung des unbedeckten Himmels sichtbar. Das Ziel ist, die Robustheit der neuronalen Netze gegenüber der Verwendung verschiedener Kameras bei der Wolkenerkennung zu verbessern.

5.7.4 Ermittlung des Wolkenbedeckungsgrades

Sobald der Beobachtungshorizont für die Wolkenbedeckungsmessung eingestellt ist, soll die Wolkenbedeckung in dem verbleibenden Teil des Himmelsraums bestimmt werden. Dafür müssen entweder die Eingangs- oder die Ausgangsbilder des neuronalen Netzes entsprechend maskiert werden, damit die überflüssigen Bereiche bei der Berechnung des Bedeckungsgrades nicht berücksichtigt werden.
Bei der Nutzung von Fischaugenobjektiven, wie sie hier verwendet werden, muss mit Bildverzerrungen gerechnet werden. Dabei gibt es verschiedene Projektionstypen von

Fischaugenobjektiven mit unterschiedlichen Skalierungsfehlern[15] bis hin zu keinem nennenswerten Skalierungsfehler im Fall von flächentreuen Projektionen. Folglich lässt sich eine solche Bildentzerrung nicht einheitlich für jede Kamera umsetzen. Zu beachten ist auch, dass die Fehlerkompensation ausschließlich Fehler durch die kamerainterne Projektion berücksichtigt. Perspektivische Fehler können dadurch nicht eliminiert werden. Diese sind nach [Strauss, 2007] bei hinreichender Homogenität der Bewölkung als gering zu bewerten. Darüber hinaus werden perspektivische Effekte in [DWD, 1967] bei der Wetterbeobachtung nicht thematisiert und werden daher als vernachlässigbar eingestuft. Tabelle 5.9 stellt der Nichtnutzung von Korrekturverfahren mathematische Näherungen der Skalierungsfehler für eine geeignete Pixelgewichtung sowie die vollständige Bildentzerrung gegenüber.

Tabelle 5.9: Berücksichtigung von Verzerrungsfehlern durch die Kameraobjektive bei der Berechnung des Wolkenbedeckungsgrades

Bewertungskriterium	**keine Bearbeitung**	**Pixel-gewichtung**	**Bildentzerrung**
Implementierungsaufwand	+	o	-
Domain Transfer	+	o	-
Informationsverlust	+	+	-
Skalierungsfehler	-	o	+

Für die Bildentzerrung gibt es verschiedene Verfahren. [Uras, 2015] nutzt die Verfahren nach [Zhang, 2000] und [Scaramuzza u. a., 2006] für die Bildentzerrung und berichtet bei diesen Verfahren von Informationsverlusten und Bildunschärfen in den äußeren Bildbereichen. Auch bei CNN-gestützten Verfahren wie von [Xue u. a., 2019] gibt es einen nicht zu vernachlässigenden Informationsverlust in den äußeren Bildbereichen. Je nach eingestelltem Beobachtungshorizont kann das Ergebnis der Bedeckungsmessung dadurch beeinflusst werden.
Bei der Nutzung von Gewichtungsmatrizen, welche durch mathematische Näherung der Pixelskalierung erstellt werden, gibt es keinen Informationsverlust im Bild, da das Bild nicht entzerrt wird. Da es sich in diesem Fall nur um eine Näherung und nicht um eine exakte Kamerakalibrierung handelt, wird der Skalierungsfehler weniger genau herausge-

[15] Objekte gleicher Größe werden in verschiedenen Bildbereichen unterschiedlich groß dargestellt.

rechnet als bei der Bildentzerrung. Der Arbeitsaufwand für die Umsetzung des Domain Transfers auf eine andere Kamera wird bei der Pixelgewichtung als geringer gegenüber der Bildentzerrung eingeschätzt, da keine aufwendige Ermittlung extrinsischer und intrinsicher Parameter und Kalibrierung der neuen Kamera durchgeführt werden muss. Für eine mathematische Näherung muss jedoch das Projektionsmodell der genutzten Optik ermittelt und eine geeignete Näherung der Skalierung implementiert werden.
Die in der vorliegenden Arbeit genutzten Kameras besitzen beide laut der vom Hersteller bereitgestellten Kamerahandbücher[16] bereits softwareseitige Bildentzerrungen. Darüber hinaus wird durch den geänderten Beobachtungshorizont der am stärksten verzerrte Bildbereich bereits ausgeblendet. Nach [Baierl, 2008] ist die flächentreue Abbildungsfunktion, bei der Objektflächen relativ zum Raumwinkel korrekt abgebildet werden, die am weitesten verbreitete Funktion und ermöglicht beispielsweise bei Wolkenaufnahmen eine korrekte Ablesung des Bedeckungsgrades. Unter Berücksichtigung dieser Aspekte wird im weiteren Verlauf keine zusätzliche Pixelgewichtung beziehungsweise Bildentzerrung genutzt.

5.8 Identifikation nicht nutzbarer Kameraaufnahmen

Durch Umwelteinflüsse können in den Kameraaufnahmen Störeffekte wie beispielsweise Überlichtung, starker Nebel oder Regentropfen auf dem Objektiv auftreten. Da diese Bilder bereits bei der Datenaufbereitung zu Problemen führen und teilweise nicht segmentierbar sind, werden diese entsprechend nicht für das Training der neuronalen Netze verwendet. Um das Gesamtsystem dennoch robust gegenüber diesen Aufnahmen zu gestalten, soll deren Nutzbarkeit vor der Segmentierung evaluiert werden.
In Tabelle 5.10 werden drei Ansätze für diese Selektierung gegenübergestellt. Ein Ansatz der klassischen Bildverarbeitung für die Nebelerkennung wird von [Padberg, 2021] umgesetzt. Dieser Ansatz wird einem Xception-Klassifikator, wie er von [Perkovic, 2022] genutzt wird, gegenübergestellt. Als dritter Ansatz wird die Verwendung des Domain-Klassifikators aus Kapitel 5.5 in Betracht gezogen.

[16] Handbuch Mobotix Q26: https://www.mobotix.com/sites/default/files/2019-10/mx_ML_Q26_de_20190521.pdf (zul. aufgerufen am 25.03.2022).
Handbuch Vivothek FE9381-EHV: https://webapi.vivotek.com/api/DownloadCenter/Download/global?p1=Y0QptKu1tPRlcS7KZFA5NQ==&p2=j199RdC0wOvZ8pPmTck8Tg== (zul. aufgerufen am 25.03.2022).

Tabelle 5.10: Gegenüberstellung von Selektierungsstrategien zur Identifikation von nicht nutzbaren Kameraaufnahmen

Bewertungskriterium	**klassische Bildverarbeitung**	**Xception**	**Domain Klassifikator**
Einbindung neuer Domains	-	o	o
Implementierungsaufwand	-	o	+
Verarbeitung variabler Bildgröße	o	-	+

Nach Tabelle 5.10 erfordert die Einbindung neuer Domains in allen drei Fällen zusätzliche Aufwände. Die Realisierung anhand eines Xception-Klassifikators nach [Perkovic, 2022] sowie über den Domain-Klassifikator erfordert die Erstellung eines Datensatzes mit nicht nutzbaren Bildern für das Training. Die Verwendung von Augmentations für das Training von CNN, welche beispielsweise Überlichtungung oder Regentropfen imitieren, reduziert den Aufwand bei der Datensatzerstellung. Das für die Nutzung klassischer Bildverarbeitung notwendige Feature Engineering ist dagegen wesentlich arbeitsintensiver, da in diesem Fall für jede Form der Störung in den Kameraaufnahmen eigens zugeschnittene Lösungen entwickelt werden müssen.
Im Fall der klassischen Bildverarbeitung korreliert der Implementierungsaufwand mit dem Aufwand für die Einbindung neuer Domains. Die Infrastruktur für die Erstellung künstlicher neuronaler Netze ist bereits vorhanden. Der Implementierungsaufwand für den Domain-Klassifikator ist am geringsten, da dieser für das DATL implementiert wird und somit direkt verwendet werden kann.
Als letztes Bewertungskriterium wird die Verarbeitung von Bildern variabler Größe aufgeführt. Während der in dieser Arbeit implemetierte Domain-Klassifikator verschiedene Eingangsgrößen verarbeiten kann, ist dies bei der Implementierung des Xception-Klassifikators durch die Vektorisierung der Featuremaps nicht vorgesehen. Bei der klassischen Bilderarbeitung ist dies grundsätzlich möglich, setzt jedoch die Verwendung von skalierungsinvarianten Merkmalen voraus.
Nach Analyse der Tabelle 5.10 wird im Folgenden der Domain-Klassifikator für die Selektierung von nicht nutzbaren Bildern gewählt.

6 Entwicklung und Implementierung

Das nachfolgende Kapitel fasst die Entwicklung und Implementierung der im Rahmen dieser Arbeit erarbeiteten Lösungen zusammen. Dabei wird die konkrete Umsetzung der Konzepte zur Erfüllung der Anforderungen erläutert. Es wird zunächst die eigens erstellte Softwarelösung für die effiziente Aufbereitung von Trainings-, Test- und Validierungsdaten behandelt. Anschließend befasst sich dieses Kapitel mit der Implementierung von ausgewählten Netzarchitekturen und der dazugehörigen Infrastruktur zum Trainieren und Testen der erstellten künstlichen neuronalen Netze. Für das Training der neuronalen Netze werden anschließend geeignete Trainingsparameter ermittelt. Schließlich wird die Anwendung für Endnutzer*innen thematisiert, welche die Ermittlung des Schnee- und Wolkenbedeckungsgrades ermöglicht.

6.1 Datenaufbereitung

In diesem Abschnitt wird die konzeptionell erforderliche Softwarelösung für die Datenaufbereitung entwickelt. Hierzu werden zunächst die Funktionsweise sowie die Schnittstellen dargstellt. Eine Betrachtung der Einbindung von Projekten und Benutzer*innen sowie der für die Erstellung der Labelmaps nötigen Funktionen erfolgt anschließend. Schlussendlich folgt eine Erläuterung des Programmablaufs.

6.1.1 Funktionsweise

Das zu entwickelnde Programm, welches nachfolgend als ‚Labelmizer' bezeichnet wird, ermöglicht das Erstellen von Labelmaps. Diese können für das Training künstlicher neuronaler Netze zur semantischen Segmentierung verwendet werden. Die Einbindung verschiedener Benutzer*innen und Projekte mit variabler Klassenzahl wird umgesetzt. Weiterhin sorgen eine geordnete Menüführung und ein optionales Zoomen beim Erstellen der Labelmaps für ein möglichst genaues Ergebnis. Funktionen zur Teilautomatisierung der

Datenaufbereitung, die auf Basis der Eingangsbilder und Eingaben von Benutzer*innen eine Vorabsegmentierung durchführen, sowie die Verwendung von Maskierungen erhöhen die Effizienz bei der Erstellung der Labelmaps. Zudem erhält das Programm eine Funktion zur manuellen Korrektur der Vorabsegmentierung über eine Bereichsmarkierung. Ein in den Labelmizer integriertes Datenmanagement speichert die Bilder mit den dazugehörigen Labelmaps ab und ermöglicht den späteren Zugriff auf die verarbeiteten Dateien durch die Befüllung einer Statusdatei, welche als Datensatz mit Metadaten für alle aufbereiteten Bildpaare des jeweiligen Projektes fungiert. Weiterhin wird der Labelmizer robust gegen fehlerhafte Eingaben der Benutzer*innen ausgelegt, sodass diese eine wiederholte Eingabe fordern und nicht zum Programmabsturz führen.

6.1.2 Schnittstellen

Eine robuste Entwicklung der Datenaufbereitung erfordert einen strukturierten Aufbau sowie definierte Schnittstellen, um eine einheitliche Verwendung der generierten Daten zu ermöglichen. Die Schnittstellen des hier entwickelten Labelmizers sind in Abbildung 6.1 dargestellt. Eingangsparameter sind die zu segmentierenden Bilder, eine Konfigurationsdatei für Informationen zu Projekten und Benutzer*innen sowie manuelle Eingaben, die den Ablauf des Programms steuern. Ausgegeben werden die erstellten Labelmaps und Statusdateien mit den Metadaten der verarbeiteten Bildpaare.

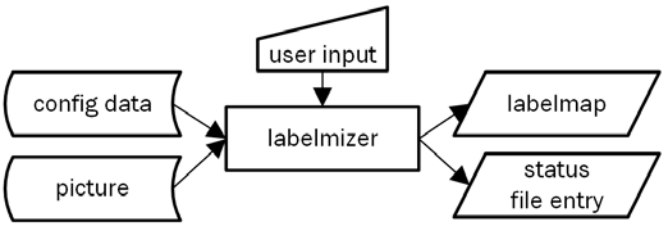


Abbildung 6.1: Schnittstellen der Datenaufbereitung

Die Labelmaps werden als png-Datei mit dem zugehörigen Bild in einem eigens erstellten Unterordner gespeichert. Die Statusdatei enthält unter anderem Informationen zu den Speicherorten der Bilder und Labelmaps. Zur einfachen Auswertung wird für jedes Projekt eine eigenständige Datei erzeugt. Für die Verwendung des Labelmizers durch mehrere Benutzer*innen wird für alle Nutzer*innen ebenfalls eine separate Statusdatei erzeugt. Dies verhindert Zugriffsprobleme bei gleichzeitiger Nutzung des Programms und beugt Datenverlust durch Überschreiben vor. Als Dateiformat wird hier, abweichend zu

Tabelle 5.1, das csv-Format verwendet. Durch strukturell gleichbleibende Einträge in Tabellenform ist dieses Format hier zweckmäßig. Ein zusätzliches Kontrollprogramm bietet die Möglichkeit, alle Statusdateien einzulesen und die Verfügbarkeit der referenzierten Bildpaare zu prüfen sowie eventuelle Speicherfehler aufzuzeigen.

6.1.3 Einbindung von Projekten und Benutzer*innen

Durch die Verwendung des Labelmizers für die Schnee- und Wolkenerkennung muss dieser die Möglichkeit bieten, zwischen den Projekten zu unterscheiden. Um dies zu gewährleisten, werden projektspezifische Daten in eine Konfigurationsdatei ausgelagert. Codeausschnitt 6.1 zeigt die Einbindung des Projektes Schneeerkennung in den Labelmizer.

Codeausschnitt 6.1: Labelmizer: Projekteinbindung Labelmizer

```
"name": "snow",
"shortcut": "s",
"visualization_factor": 100,
"distance_factor": 1.1,
"border_correction": 0.01,
"classes": [
      {"label": "background", "value": 0},
      {"label": "no_snow", "value": 1},
      {"label": "snow", "value": 2}
]
```

An dieser Stelle werden die Parameter Visualisierungsfaktor, Distanzfaktor und Randkorrektur, welche im nachfolgenden Kapitel erläutert werden, festgelegt. Außerdem werden die möglichen Klassen für die semantische Segmentierung über diese Datei definiert.
Darüber hinaus enthält die Konfigurationsdatei Informationen über eingebundene Standorte beziehungsweise Datensätze und Informationen zu Benutzer*innen wie Namen und Speicherpfade der Bilddaten. Dadurch wird die Verwendbarkeit auf verschiedenen Endgeräten sichergestellt.

6.1.4 Erstellung der Labelmaps

Um die Hauptaufgabe des Labelmizers – die Erstellung der Labelmaps – zu realisieren, werden verschiedene Funktionen entwickelt. Eine dieser Funktionen ist die Schwellwertsegmentierung. Diese extrahiert die Klasse des Bedeckungsgrades aus dem Ordnernamen,

dem das Bild zugeordnet ist und berechnet den mittleren Grauwert. Auf Basis dieser Werte ermittelt die Funktion den Schwellwert, über dem alle Pixel der Klasse Schnee oder Wolken zugeordnet werden. Eine manuelle Korrektur des Schwellwertes ist anschließend möglich.
Die Segmentierung der Hintergrundklasse kann durch die Verwendung einer zuvor erstellten Hintergrundmaske unterstützt werden. Neben den Hintergrundmasken wird auch die manuelle Verwendung von Masken für die restlichen Klassen implementiert. Die Aktivierung von Masken wird ebenfalls in die Konfigurationsdatei ausgelagert.
Eine weitere Segmentierungsoption wird mit dem ‚ColorGrabber' implementiert. Hierbei besteht die Möglichkeit, durch Klicken in das angezeigte Bild einen Pixel zu extrahieren und Pixel mit ähnlichen Grauwertverhältnissen zwischen den Farbkanälen mit einer zuvor ausgewählten Klasse zu segmentieren. In Abbildung 6.2 ist die Berechnung dieser Verhältnisse abgebildet[17]. Der Distanzfaktor gibt dabei den Grad der zu erreichenden Ähnlichkeit der Grauwertverhältnisse an. Ein Wert von $1,1$ stellt eine maximale Abweichung der Farbverhältnisse von 10 % dar. Eine anschließende Auswertung aller Bildpixel nach diesen Verhältnissen bildet die Grundlage der Segmentierung.

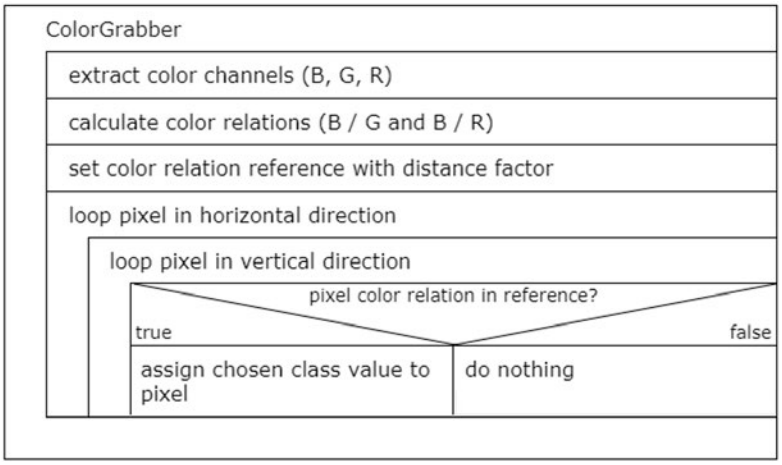


Abbildung 6.2: Funktionsweise der Klasse ColorGrabber

Eine weitere Kernfunktion des Labelmizers ist die manuelle Segmentierung über eine Bereichsmarkierung. Mit dieser Funktion wird eine Korrektur der Ergebnisse der vorangegangenen Funktionen ermöglicht. Überdies kann ein Bild mit dieser Funktion auch gänzlich manuell segmentiert werden. Hierbei wird ein Bildbereich durch Mausklicks

[17] Der Codeausschnitt ist in Anhang B.1 zu finden.

markiert, welcher dann mit der zuvor gewählten Klasse segmentiert wird. Je nach Projekteinstellung werden randnahe Pixel durch die Randkorrektur genau an den Bildrand gesetzt. Der Wert für die Randkorrektur gibt den Abstand zu dem Rand des Bildes an, in dem die Funktion aktiv ist. Der Wert $0,01$ steht für einen Abstand von 1 % der jeweiligen Bilddimension.
Um diese manuelle Nacharbeit gering zu halten, falls sich beispielsweise die Schneebedeckung zwischen den Bildern nur geringfügig ändert, wird in einer Sitzung die zuvor gespeicherte Labelmap als Vorschlag für die Segmentierung des darauffolgenden Bildes angezeigt. Diese kann entweder direkt gespeichert oder mit den oben genannten Funktionen bearbeitet werden. Die Labelmaps werden beim Speichern zusätzlich mit einem Visualisierungsfaktor, welcher der Konfigurationsdatei entnommen wird, multipliziert. Dadurch wird eine optische Auswertung ermöglicht, da die Labelmaps sonst nur Grauwerte enthalten, die den jeweiligen Nummern der Klassen entsprechen und daher visuell nicht unterscheidbar sind.

6.1.5 Programmablauf

Für eine geordnete Führung der Benutzer*innen während des Programmablaufs wird ein Objekt erstellt, welches die Variablen für die Steuerung des Programmablaufs enthält, die nicht direkt die Segmentierung betreffen. Die Nutzung des Objektes erfolgt im Hauptmenü. Hier werden außerdem die Funktionen zum Bearbeiten der Labelmap ausgeführt (siehe Abb. 6.3).

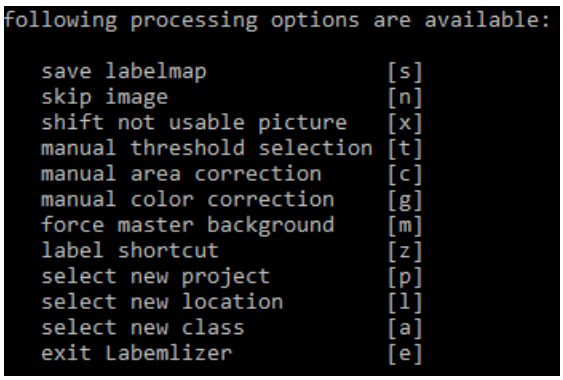


Abbildung 6.3: Hauptmenü des Labelmizers

Der gesamte Programmablauf des Labelmizers ist in Abbildung 6.4 dargestellt. Nach dem Start des Labelmizers wird zunächst die Konfigurationsdatei eingelesen. Anschließend

werden Projekt und Datensatz gewählt sowie ein Bild zur Segmentierung automatisch bestimmt. Über das Ablaufobjekt wird die Labelmap bearbeitet und das Ergebnis gespeichert. Alternativ werden neue Konfigurationen gewählt oder das Bild als nicht nutzbar aussortiert. Dieser Vorgang wird wiederholt, bis der Labelmizer beendet wird.

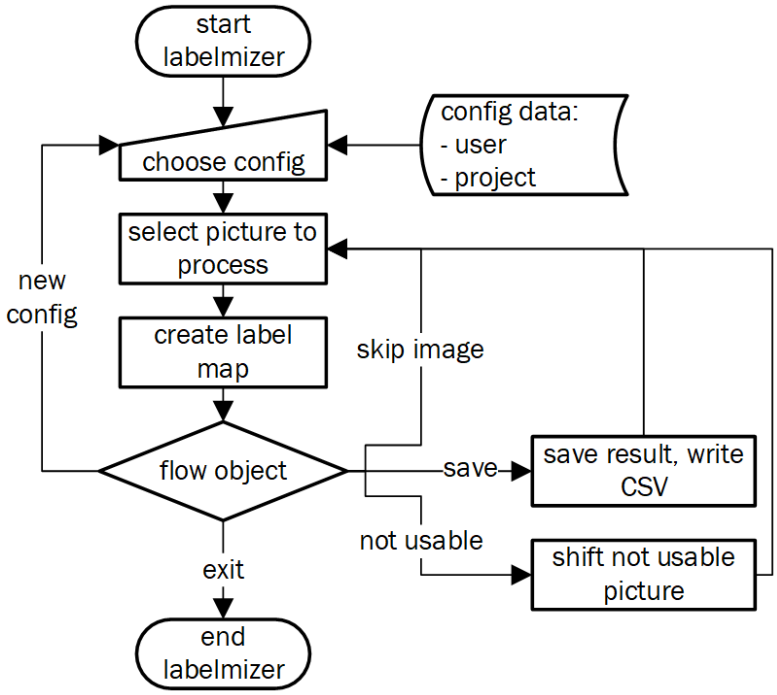


Abbildung 6.4: Programmablauf des Labelmizers

6.2 Netzimplemetierung

Dieses Kapitel behandelt die Entwicklung und Umsetzung der Softwarelösung zur Implementierung der künstlichen neuronalen Netze. Diese umfasst im Wesentlichen ein Programm für das Training sowie ein Programm für das Testen der neuronalen Netze. Außerdem ist die Implementierung der in Kapitel 5.4 ausgewählten Netz-Architekturen Bestandteil dieses Programms.
Zunächst wird die Funktionsweise des Programms erläutert. Zusätzlich werden die verschiedenen Schnittstellen des Programms erklärt. Außerdem wird die Trainingskonfiguration inklusive der Einbindung von Projekten und Benutzer*innen dargestellt. Des Weiteren werden wichtige Programmfunktionen sowie der detaillierte Programmablauf gezeigt.

6.2.1 Funktionsweise

Die Softwarelösung für die Netzimplementierung erhält neben den Funktionen für Training und Prediction auch verschiedene Funktionen für das Datenmanagement, die Architekturimplementierung und die Ergebnisauswertung.
Das Datenmanagement ermöglicht die Erstellung und Speicherung von Referenzdatensätzen zum Trainieren und Testen sowie deren Wiederverwendung. Diese Datensätze sind für die Vergleichbarkeit verschiedener Tests notwendig. Zudem wird so eine Trainingsfortsetzung mit bestehenden oder neuen Datensätzen ermöglicht. Außerdem werden verschiedene Daten während des Trainierens und Testens gespeichert. Der Trainingsablauf wird epochenweise in einer Datei ausgegeben. Dabei können Rückgabewerte wie beispielsweise Loss oder Validation-Loss gespeichert werden. Auf diese Weise wird eine bessere Nachvollziehbarkeit des Trainings ermöglicht, wenn dieses nicht permanent überwacht wird. Ebenfalls in einer Datei gespeichert werden der Aufbau des neuronalen Netzes mit allen Informationen zu den einzelnen Netzschichten und deren Parameteranzahl sowie die genutzten Trainingsparameter.
Während des Testens der neuronalen Netze werden die prädizierten Labelmaps der Netze hinsichtlich der Bewertungskriterien wie Pixelwise Accuracy oder F1-Score ausgewertet und die Ergebnisse in einer Datei gespeichert. Für das Training der Netze wird ein Batch-Generator entwickelt, der die Trainings- und Validierungsdaten gemischt in Batches aufteilt und Funktionen für die Erhöhung der Anzahl der Trainings- und Validierungsdaten durch Augmentations und Bildzerstückelung enthält. Die Implementierung der Architekturen erfolgt hier modular, da bestimmte Netz-Bausteine wie beispielsweise der CAM oder der SAM von mehreren Architekturen genutzt werden.

6.2.2 Schnittstellen

Die Schnittstellen des Programms sind in Abbildung 6.5 abstrahiert dargestellt. Die Programmteile für Training und Test der neuronalen Netze sind getrennt voneinander zu betrachten und auch getrennt voneinander ausführbar. Die beiden Programmteile haben jeweils eine Schnittstelle zu den Benutzer*innen, über die Eingaben zur Trainings- und Testkonfiguration abgefragt werden. Das Trainingsprogramm kann neue Datensätze in Form einer json-Datei nach den Eingaben der Benutzer*innen erzeugen. Diese enthalten nur Metadaten der Bilder, über welche die ausgewählten Bilder geladen werden können. Außerdem wird die ausgelagerte Trainingskonfiguration verwendet.

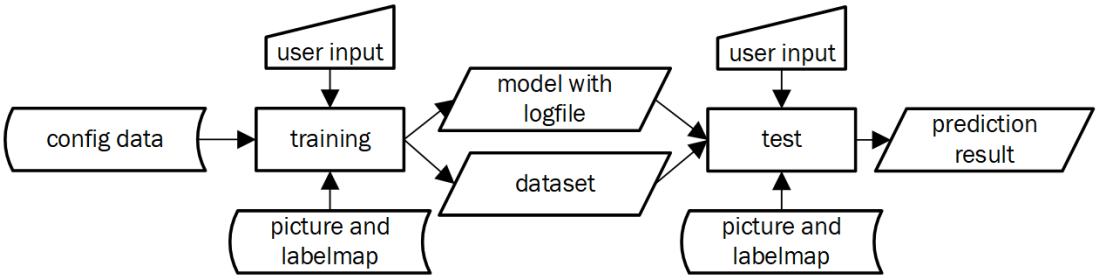


Abbildung 6.5: Schnittstellen der Netzimplementierung

Das Testprogramm erhält Informationen über die Testdaten aus dem Datensatz, der von dem Trainingsprogramm erstellt wird. Außerdem nutzt es für die Prediction das von dem Trainingsprogramm erzeugte Modell des neuronalen Netzes. Über manuelle Eingaben wird abgefragt, für welches Projekt und welche Standorte ein Test durchgeführt werden soll. Das Testprogramm gibt die prädizierten Labelmaps zusammen mit den Testbildern aus und errechnet für die Testdaten die ausgewählten Bewertungskriterien, welche in einer separaten Datei gespeichert werden.

6.2.3 Trainingskonfiguration außerhalb des Programmcodes

Die Konfiguration des Trainings findet außerhalb des Programmcodes in json-Dateien statt. Dafür werden Informationen zu Nuter*innen wie Namen und Pfade zu den Bilddaten ähnlich wie bei der Datenaufbereitung aus Kapitel 6.1.3 zusammen mit Informationen zu Projekten und dazugehörigen Standorten in einer Datei konfiguriert.
In der json-Datei aus Codeausschnitt 6.2 werden die Trainingskonfiguration sowie die Konfigurationen der Netz-Architekturen am Beispiel der semantischen Segmentierung außerhalb des Programmcodes gezeigt. Neben der semantischen Segmentierung werden in dieser Datei das DATL sowie die Klassifikatoren konfiguriert. Zur Trainingskonfiguration gehören Informationen zu Bildzerstückelung, Augmentations sowie Trainingsparameter wie Batchsize, Learning-Rate und Epochenanzahl. Die Architekturkonfigurationen werden in einer Liste organisiert, in der jede Architektur eigene Parameter zugewiesen bekommt. Darüber können die Architekturen variabel bezüglich ihrer Netztiefe, Filter Kernel, Pooling Kernel, Parameteranzahl und weiterer spezifischer Parameter eingestellt werden. An dieser Stelle wird außerdem eingestellt, ob die Eingangsbilder des Netzes für das Training eine feste oder variable Größe erhalten. Diese Datei kann bei Bedarf um neue Trainings- und Architekturkonfigurationen erweitert werden.

Codeausschnitt 6.2: Netzimplementierung: json-Datei für Trainingsparameter und Architekturkonfiguration

```
"name": "semantic-segmentation",
"shortcut": "ss",
"slice_factor": {"snow": 1, "cloud": 2},
"data_augmentation": [
      {
            "operation": "Rotate",
            "value": 180,
            "active": false,
            "quadrant": null
      }
]
"hyperparameter": {
      "learning_rate": 0.0002,
      "batch_size": 8,
      "epochs": 100,
      "min_lr": 0.00000001
},
"architecture_list": [
      {
            "name": "KMTX_net",
            "shortcut": "km",
            "shape_factor": null,
            "variable_shape": true,
            "CSAM": {"c_unit_factor": 5, "s_kernel": 9},
            "bridge_parameter": {...},
            "output_layer": {...},
            "pooling_layer": [...]
      }
]
```

6.2.4 Erstellung künstlicher neuronaler Netze

In diesem Abschnitt wird die Implementierung künstlicher neuronaler Netze fokussiert. Hierzu wird zunächst der grundsätzliche Modellaufbau, gefolgt von der Handhabung der zu verwendenen Bildgröße beziehungsweise des Shape-Factors beschrieben. Anschließend wird die Umsetzung der DATL-Erweiterung dargestellt.

Netzaufbau

Der Aufbau von künstlichen neuronalen Netzen in Python unter der Verwendung von Keras und Tensorflow erfolgt schrittweise. In Anhang B.2 ist ein solcher Aufbau mit den dazugehörigen Bibliotheken vereinfacht dargestellt. Dort werden die Netz-Layer anhand der Architekturkonfiguration aus Codeausschnitt 6.2 aufgebaut. Zusammen mit Input- und Output-Layer wird das Gesamtmodell erstellt.
Ein konkretes Beispiel für einen Netzbaustein, der für den Übergang zwischen zwei Netz-Layern genutzt werden kann, ist das CAM aus Abbildung 6.6 und Anhang B.3. Dieses gewichtet die Featuremaps des CAM-Eingangs nach Relevanz. Umgesetzt wird dies durch ein Global-Average- und Global-Max-Pooling mit anschließender Anwendung eines shared MLP. Der ‚c_unit_factor' aus der Konfigurationsdatei gibt hierbei die Anzahl der Neuronen des Hidden-Layers in Bezug auf die Anzahl der Featuremaps am Eingang an. Die schlussendliche Gewichtung der Featuremaps erfolgt durch die Multiplikation des shared MLP-Ausgangs mit dem CAM-Eingang.

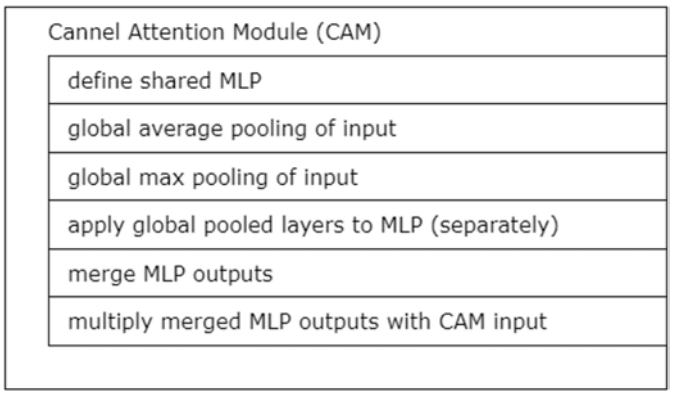


Abbildung 6.6: Funktionsweise des Channel Attention Module (CAM)

Das SAM gewichtet anders als das CAM räumliche Merkmale in den Eingangsbildern. Hierzu wird ein Max-Channel- und Average-Channel-Pooling auf den Eingangsdaten durchgeführt (siehe Abb. 6.7 / Anhang B.4). Eine anschließende Gewichtung der räumlichen Merkmale durch eine Faltung sowie die Multiplikation der gewichteten Featuremap mit den Eingangsdaten des SAM bildet den SAM-Ausgang.

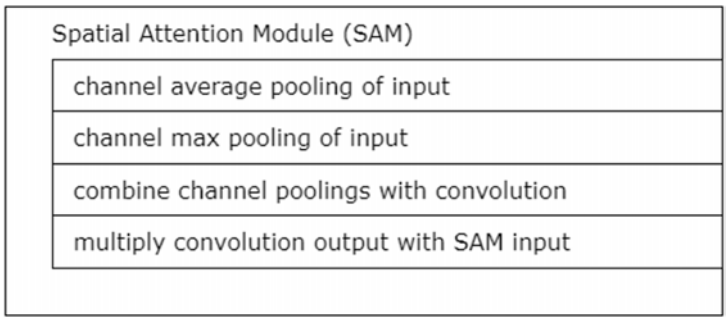


Abbildung 6.7: Funktionsweise des Spatial Attention Module (SAM)

Anpassung der Bildgröße

Um Dimensionsproblemen während des Trainings und der späteren Anwendung der neuronalen Netze vorzubeugen, ist teilweise eine Anpassung der Eingangsbildgröße notwendig. Dies wird durch die Bestimmung des Shape-Factors gewährleistet. Der Shape-Factor wird, wie in Anhang B.5 dargestellt, der Konfigurationsdatei entnommen oder alternativ über die Anzahl und Größe der verwendeten Pooling-Kernel berechnet. Um die spätere Anwendung von trainierten Modellen zu vereinfachen, werden die Informationen bezüglich des Shape-Factors während der Trainingsphase im Modell gespeichert. Bei der Prediction wird der Shape-Factor dann aus dem Modell extrahiert und alle Eingangsbilder auf die maximal zulässige Bildgröße reduziert.

DATL-Erweiterung

Das DATL auf Merkmalsebene erfordert die Erweiterung einer Netzarchitektur für die semantische Segmentierung um einen Domain-Klassifikator. Die Implementierung des Domain-Klassifikators wird in Abbildung 6.8 und Anhang B.6 dargestellt. Vor dem FCL werden die Ausgänge aller Pooling-Layer des Domain-Klassifikators sowie der Ausgang der Bridge des Segmentierungsnetzwerkes mittels Global-Poolings bezüglich ihrer Dimensionen Höhe und Breite angeglichen und konkateniert. Auf diese Weise wird die Verwendung von Eingangsbildern variabler Größe gewährleistet und die Parameteranzahl des Klassifikators begrenzt. Anhand der in der angeglichenen Featuremaps erfolgt schließlich die Klassifizierung durch den FCL.

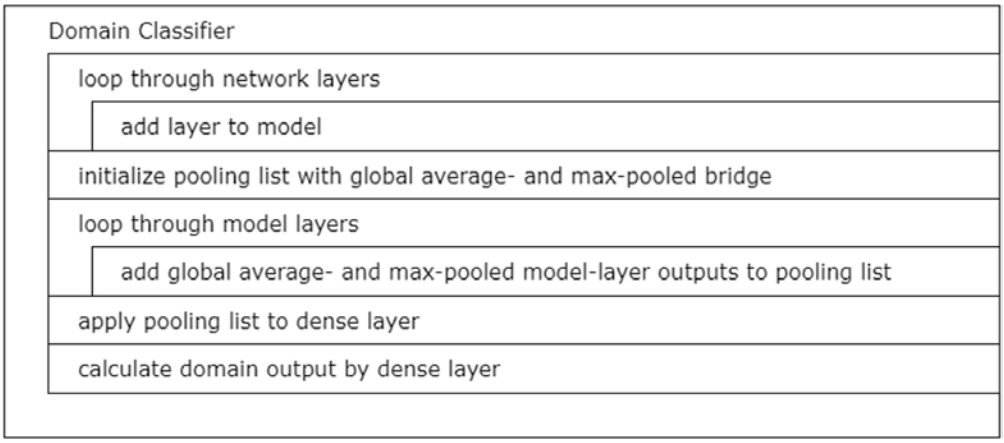


Abbildung 6.8: Funktionsweise des Domain-Klassifikators

6.2.5 Programmablauf

Der folgende Abschnitt beschreibt den Programmablauf für das Trainieren und Testen der neuronalen Netze. Der in Abbildung 6.9 dargestellte Trainingsablauf startet mit der Abfrage von Informationen für die Trainings- und Architekturkonfiguration. Die erforderlichen Eingaben während des Trainings beinhalten außerdem den zu verwendenden Modelltyp[18], die Architektur und Angaben bezüglich der Verwendung oder Neuerstellung des Datensatzes. Diese Informationen werden beim Testen automatisch den vorhandenen Dateien entnommen. Abhängig von der manuellen Eingabe wird anschließend ein neuer Datensatz erstellt und gespeichert oder ein vorhandener Datensatz geladen. Daraufhin werden dem Datensatz die voreingestellten Informationen zu Augmentations und der Bildzerstückelung hinzugefügt. Falls das Weitertrainieren eines vorhandenen Netzes verlangt wird, wird ein vorhandenes Netzmodell geladen, ansonsten wird ein neues Modell basierend auf der Architekturkonfiguration erstellt. Anschließend folgt die Erstellung der Trainings- und Validierungs-Batches mit dem gewählten Datensatz unter Berücksichtigung der in der Konfigurationsdatei voreingestellten Batchsize. Mit den Batches erfolgt das Training des Modells mit der voreingestellten Learning-Rate und Epochenanzahl. Schließlich wird die Datei für die Trainings- und Netzbeschreibung gespeichert.
Das Testprogramm testet ein zuvor trainiertes neuronales Netz mit dem dafür erstellten Datensatz, woraufhin die Ergebnisse hinsichtlich der Bewertungskriterien gespeichert werden.

[18] hier: semantische Segmentierung, DATL oder Klassifikation.

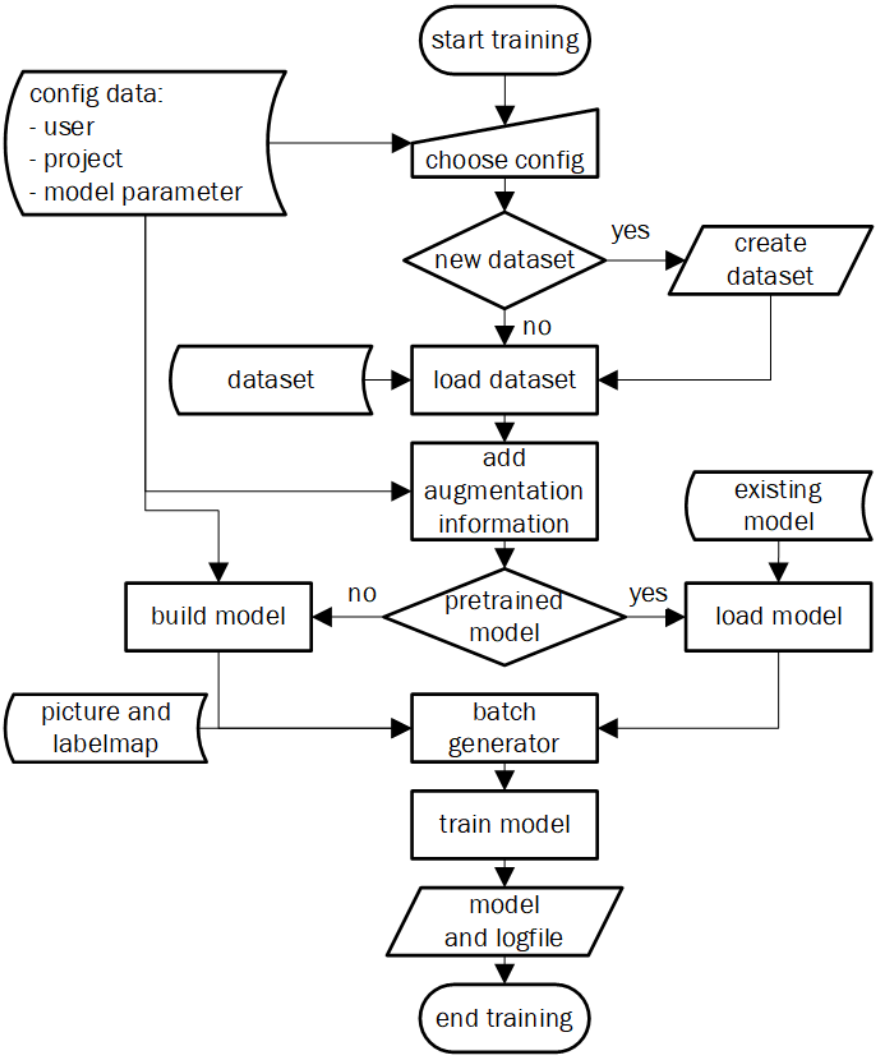


Abbildung 6.9: Programmablauf des Netz-Trainings

6.3 Einstellen der Trainingsparameter

Für das Training der künstlichen neuronalen Netze zur Schnee- und Wolkenerkennung werden die Trainingsparameter Batchsize und Learning-Rate festgelegt. Außerdem wird ein geeigneter Optimizer und eine Loss-Function ausgewählt. Für die Ermittlung der geeigneten Trainingsparameter wird ein U-Net sowie die Testdatensätze D1 (*waterday*) und D2 (*garden*) verwendet. Die in der vorliegenden Arbeit verwendeten Datensätze sind

in Anhang A aufgelistet. Die Aufteilung der Referenzdatensätze ist dabei in jedem Test identisch.

6.3.1 Optimizer und Loss-Function

Zunächst wird eine geeignete Kombination aus Optimitzer und Loss-Function ermittelt. Dafür werden, wie in Tabelle 6.1 dargestellt, die Loss-Functions Cross-Entropy- und F1-Loss mit den Optimizern Adam und Nadam in allen Kombinationen getestet. Neben diesen Loss-Functions wird ein von [Jadon, 2020] mit dem Logarithmus und dem Cosinus Hyperbolicus modifizierter Dice-Loss implementiert und getestet. Für den Test werden die Datensätze D1 (*waterday*) und D2 (*garden*) unabhängig voneinander verwendet und der Mittelwert der Ergebnisse in Tabelle 6.1 abgebildet. Erkennbar ist, dass der Adam Optimizer in Verbindung mit dem F1-Loss mit $0,8$ % Abstand den höchsten F1-Score erreicht. Die beste Pixelwise Accuracy erreicht der Nadam Optimizer zusammen mit dem Cross-Entropy-Loss, wobei diese lediglich um $0,2$ % höher ist als bei der Kombination Adam Optimizer mit F1-Loss. Aufgrund des geringen Unterschieds in der Pixelwise Accuracy und weil der F1-Score in dieser Arbeit wegen der teils ungleichen Klassenverteilung in den Bilddaten den dominanten Bewertungsparameter darstellt, wird im weiteren Verlauf dieser Arbeit der Adam Optimizer in Verbindung mit dem F1-Loss verwendet.

Tabelle 6.1: Auswahl der optimalen Kombination aus Loss-Function und Optimizer (Datensätze D1 (*waterday*) und D2 (*garden*), U-Net, Learning-Rate von $2 \cdot 10^{-4}$, Batchsize von 25)

Loss-Function	Adam		Nadam	
	Acc	**F1-Score**	**Acc**	**F1-Score**
Cross-Entropy-Loss	$95,6$ %	$74,3$ %	$95,7$ %	$73,3$ %
F1-Loss	$95,5$ %	$75,1$ %	$94,9$ %	$73,4$ %
Log-Cosh-Dice-Loss	$73,7$ %	$51,7$ %	$94,7$ %	$69,2$ %

Auffällig ist, dass die Ergebnisse des Dice-Loss mit Logarithmus und Cosinus Hyperbolicus nicht mit den Ergebnissen von [Jadon, 2020] korrelieren. Vor allem in Verbindung mit dem Adam Optimizer liegen die Accuracy und der F1-Score mit jeweils über 20 % Unterschied deutlich unterhalb der Ergebnisse der anderen verwendeten Loss-Functions.

6.3.2 Batchsize und Learning-Rate

Die Auswahl der Batchsize und der Learning-Rate erfolgt in zwei aneinandergereihten Tests. Aufgrund der starken Abhängigkeit der beiden Parameter voneinander wird zunächst eine Learning-Rate aus der Mitte des von [Yaqub u. a., 2020] als relevant dargestellten Bereichs ausgewählt. Damit erfolgt unter Anwendung von Datensatz D1 (*waterday*) nach Tabelle 6.2 eine sukzessive Auswahl der für diese Learning-Rate am besten geeigneten Batchsize zwischen 1 und 50. Die besten Ergebnisse sowohl beim F1-Score, als auch bei der Pixelwise Accuracy ergeben sich bei der Verwendung einer Batchsize von 10.

Tabelle 6.2: Ermittlung der geeigneten Batchsize bei fester Learning-Rate von $2 \cdot 10^{-4}$ (Datensatz D1 (*waterday*), U-Net, F1-Loss, Adam Optimizer)

Batchsize	**Acc**	**F1-Score**
1	88,7 %	66,9 %
2	97,6 %	70,9 %
5	98,4 %	86,0 %
10	98,5 %	87,5 %
25	97,4 %	76,0 %
50	96,5 %	68,9 %

Um die optimale Batchsize noch genauer bestimmen zu können, wird wie in Abbildung 6.10 dargestellt der F1-Score aus Tabelle 6.2 in einem Graph über der Batchsize aufgetragen und die Datenpunkte über eine Spline-Interpolation miteinander verbunden. Die Auswertung des Maximums der Spline-Interpolation ergibt eine theoretisch optimale Batchsize von 7,5 für die ausgewählte Learning-Rate von $2{\cdot}10^{-4}$. Da die Kurve der Spline-Interpolation in Richtung der nächst größeren Batchsize flacher abfällt als in Richtung der nächst kleineren Batchsize, wird der Optimalwert der Batchsize auf 8 aufgerundet.

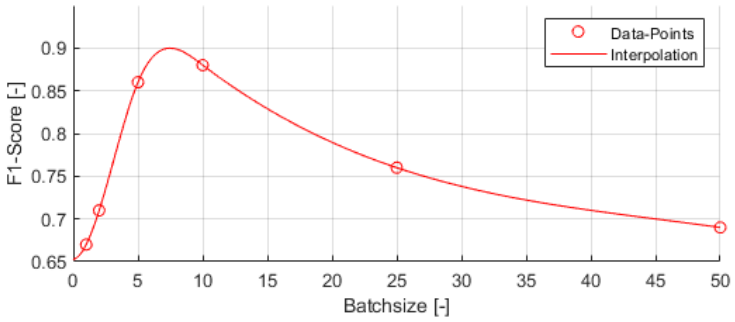


Abbildung 6.10: Ermittlung der optimalen Batchsize mittels Spline-Interpolation

Im letzten Schritt wird die, in Tabelle 6.2 praktisch ermittelte und in Abbildung 6.10 theoretisch optimierte, Batchsize validiert. Dafür wird der Test aus Tabelle 6.2 mit festgelegter Batchsize von 8 und variabler Learning-Rate durchgeführt (siehe Tab. 6.3). Dabei wird der gesamte nach [Yaqub u. a., 2020] relevante Bereich der Learning-Rate getestet. Der Bereich um die ausgewählte Learning-Rate von $2 \cdot 10^{-4}$ wird zudem in feineren Abstufungen getestet, um zu zeigen, dass die Optimierung der Batchsize auf die ausgewählte Learning-Rate der Realität entspricht. Ein direkter Vergleich des Ergebnisses bei einer Batchsize von 10 mit Learning-Rate $2 \cdot 10^{-4}$ aus Tabelle 6.2 mit dem Ergebnis bei einer Batchsize von 8 und gleicher Learning-Rate aus Tabelle 6.3 zeigt, dass auch die theoretische Optimierung der Batchsize in der Praxis valide ist. Daher wird diese Kombination aus Learning-Rate und Batchsize für den weiteren Verlauf der Arbeit genutzt.

Tabelle 6.3: Validierung der ermittelten Batchsize von 8 (Datensatz D1 (*waterday*), U-Net, F1-Loss, Adam Optimizer)

Learning-Rate	**Acc**	**F1-Score**
10^{-1}	$90,4\ \%$	$65,5\ \%$
10^{-2}	$97,5\ \%$	$78,7\ \%$
10^{-3}	$98,5\ \%$	$72,5\ \%$
10^{-4}	$98,8\ \%$	$92,0\ \%$
$2 \cdot 10^{-4}$	$98,9\ \%$	$92,2\ \%$
$3 \cdot 10^{-4}$	$98,9\ \%$	$90,6\ \%$
10^{-5}	$97,4\ \%$	$70,5\ \%$
10^{-6}	$91,0\ \%$	$65,6\ \%$

6.4 Anwendung für Endnutzer*innen

In diesem Kapitel wird die Entwicklung und Implementierung der Anwendung für Endnutzer*innen dargestellt. Hierzu wird zunächst die Funktionsweise gefolgt von den Programmschnittstellen sowie die Anwendung auf verschiedene Wetterstationen erläutert. Außerdem wird die Hauptaufgabe des Programms – die Berechnung der Schnee- und Wolkenbedeckungsgrade – sowie der Programmablauf vorgestellt.

6.4.1 Funktionsweise

Die zu entwickelnde Anwendung ermöglicht unter Verwendung der erstellten künstlichen neuronalen Netze eine Berechnung des Schnee- bezieungsweise Wolkenbedeckungsgrades. Der Bedeckungsgrad soll hierbei nach Anforderung A18 in einer csv-Datei ausgegeben werden. Ein optionales Speichern der semantisch segmentierten Bilder wird nach Anforderung A20 ermöglicht. Die Parametrierung und Zuweisung von Referenzklassen für die Klassifizierung erfolgt in einer Konfigurationsdatei, um eine einfache Portierung auf weitere Wetterstationen und Anwendungsfälle nach Anforderung A17 zu ermöglichen. Um eine robuste Bildauswertung zu gewährleisten, wird das Eingangsbild auf Eignung für die Segmentierung evaluiert.
Die Robustheit der Segmentierung selbst wird durch die optionale Verwendung von Masken aus Anforderung A22 erhöht. Die Aktivierung der Maskierung erfolgt ebenfalls in der Konfigurationsdatei. Darüber hinaus wird für neue Wetterstationen das Erstellen von Referenzmasken ermöglicht. Außerdem ist bei der Schneebedeckungsmessung die Möglichkeit der Pixelgewichtung zur Kompensation der perspektivischen Verzerrung gegeben. Nach Anforderung A21 werden zwei verschiedene Möglichkeiten der Prediction implementiert. Hierbei wird zwischen der Prediction eines kompletten Bild-Ordners und der automatischen Prediction bei der Bilderstellung unterschieden. Hierzu wird ein Referenzordner angegeben, aus dem entweder einmalig alle enthaltenen Bilder prädiziert werden oder alternativ alle hier neu abgespeicherten Bilder nach Erstellung automatisch prädiziert werden.

6.4.2 Schnittstellen

Das Programm erhält eine Schnittstelle für Eingaben der Benutzer*innen, über die der Programmablauf gesteuert wird. Darüber werden Informationen zu den genutzten

Wetterstationen und die Methode für die Prediction abgefragt. Außerdem erhält das Programm eine Datenschnittstelle, über die es auf die Konfigurationsdatei sowie die neuronalen Netze für die semantische Segmentierung und die Klassifizierung der Bildnutzbarkeit zugreift (siehe Abb. 6.11).

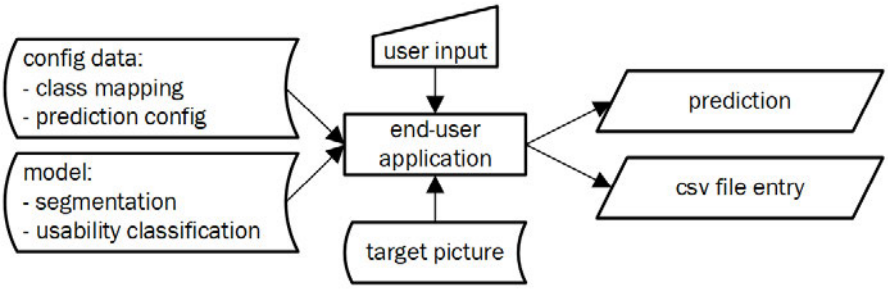


Abbildung 6.11: Schnittstellen der Anwendung für Endnutzer*innen

Nach der Segmentierung erstellt das Programm für jedes segmentierte Bild einen Eintrag in einer csv-Datei und gibt das segmetierte Bild optional aus. Eine beispielhafte Ausgabe in der csv-Datei ist in Tabelle 6.4 dargestellt.

Tabelle 6.4: Beispielhafte Ausgabe der Anwendung für Endnutzer*innen

image	value_percent	class_id	class_name
image_0.jpg	null	-1	nicht nutzbar
image_1.jpg	0	0	unbedeckt
image_2.jpg	7	1	schneereste
image_3.jpg	25	2	schneeflecken
image_4.jpg	75	3	durchbrochen
image_5.jpg	100	4	geschlossen

6.4.3 Einbindung neuer Wetterstationen

Nachfolgend wird die Einbindung von Wetterstationen in das Programm erläutert. Diese geschieht außerhalb des Programmcodes über die Konfigurationsdatei. Diese json-Datei ist in einen globalen und einen stationsbezogenen Teil aufgeteilt. Die Stationseinbindung

wird analog zu der Einbindung von Projekten und Benutzer*innen bei der Datenaufbereitung aus Codeausschnitt 6.1 in einer Liste realisiert und enthält allgemeine sowie Ordnerinformationen für die verschiedenen Stationen. Der globale Teil ist in Codeausschnitt 6.3 dargestellt.

Codeausschnitt 6.3: Anwendung für Endnutzer*innen: Stationserweiterung

```
"formats_to_predict": ["jpg", "png"],
"max_resulution": [480, 640],
"class_mapping": [
      {
            "reference": "snow",
            "values": [
                  {
                        "name": "unbedeckt",
                        "id": 0,
                        "lower_value": 0,
                        "upper_value": 1
                  }
            ]
      }
],
"default_values": {...}
```

Hier werden die für die Prediction zu berücksichtigenden Bildformate sowie eine maximale Auflösung definiert, auf die die Eingangsbilder bei Überschreitung angepasst werden. Weiterhin wird hier die Zuordnung der Bilder zu den Referenzklassen des Bedeckungsgrades konfiguriert.

6.4.4 Klassifizierung der semantisch segmentierten Kameraaufnahmen

Dieser Abschnitt behandelt die Klassifizierung der semantisch segmentierten Kameraaufnahmen in die vorgegebenen Referenzklassen. Basis hierfür bietet das in Abbildung 6.13 und Codeausschnitt 6.3 eingeführte ‚class_mapping'. Um diese Zuordnung durchführen zu können, muss zuvor der Bedeckungsgrad in Prozent berechnet werden.
Für die genauere Klassifizierung der Schneebedeckung wird es ermöglicht, eine Gewichtungsmatrix zur Kompensation von Skalierungsfehlern durch perspektivische Verzerrung zu erstellen. In Abbildung 6.12 ist exemplarisch eine Gewichtungsmatrix dargestellt, wobei schwarz einer geringeren und weiß einer stärkeren Gewichtung entspricht.

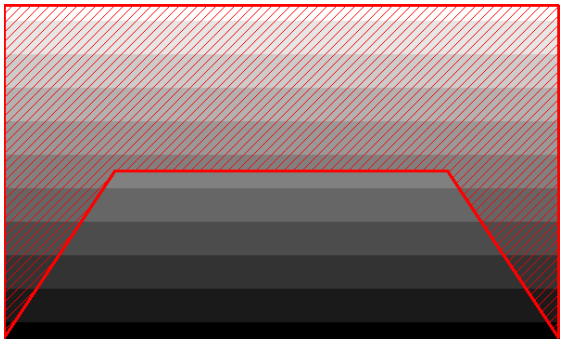


Abbildung 6.12: Visualisierte Gewichtungsmatrix zur Kompensation von perspektivischer Verzerrung mit beispielhafter Bildmaskierung (rot: maskierter Bildbereich) und Spreizung der Werte zur Veranschaulichung

Die Dimensionen der Gewichtungsmatrizen müssen den Dimensionen der semantisch segmentierten Bilder entsprechen. Jeder enthaltene Wert ist das Gewicht des jeweiligen Pixels, wobei die Summe aller Gewichte den Wert eins aufweist.
Die Kompensation der perspektivischen Verzerrung ist abhängig von der Position und Neigung der Kamera. Um die für die Kompensation nötige Gewichtungsmatrix zu ermitteln, wird ein Referenzbild des jeweiligen Standorts verwendendet, in dem zwei Objekte gleicher Ausdehnung bodennah positioniert werden. Die Positionierung der Referenzobjekte soll dabei ohne horizontalen und möglichst großem vertikalen Versatz in der horizontalen Bildmitte erfolgen. Durch Markierung der maximalen Ausdehnung der Referenzobjekte wird deren Skalierung berechnet. Unter Berücksichtigung des Pixelabstandes zwischen den Referenzobjekten im Bild wird ein Gewichtungsvektor linear auf die vertikale Bildachse skaliert. Eine anschließende Anwendung dieses Vektors auf jede Bildspalte erstellt die Gewichtungsmatrix.
Durch Runden der Werte vor dem Speichern ist es möglich, eine gröbere Bereichsunterteilung zu erhalten. Somit können in Gewichtungsmatrizen zur Kompensation perspektivischer Verzerrung Bereiche gleicher Gewichtung erzeugt werden (siehe Abb. 6.12).
Auf die Berechnung des Bedeckungsgrades folgt die Zuordnung der Bilder in die jeweiligen Referenzklassen. Die Realisierung dieser Zuordnung wird in Abbildung 6.13 exemplarisch sowie in Anhang B.7 detailiert dargestellt. Um nicht nutzbare Bilder zu separieren wird zunächst geprüft, ob ein berechneter Prozentwert vorhanden ist. Sofern dies nicht der Fall ist, wird das Bild entsprechend der Referenzklassen als nicht nutzbar markiert. Bei vorhandenem Prozentwert wird die entsprechende Klasse zugewiesen.

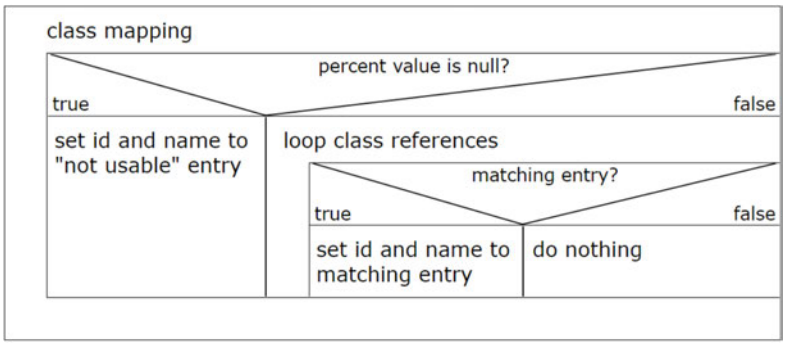


Abbildung 6.13: Funktionsweise der Klassenzuordnung

6.4.5 Programmablauf

In Abbildung 6.14 ist der Programmablauf dargestellt. Beim Programmstart wird zunächst die Konfigurationsdatei eingelesen. Über eine manuelle Eingabe wird anschließend die zu verwendende Station gewählt und damit die für diesen Programmdurchlauf gültigen Randbedingungen. Sofern in der selektierten Station das Verwenden einer Maske und einer Pixelgewichtung vorgesehen ist, werden diese eingelesen. Andernfalls wird dieser Schritt übersprungen. Ist die Maske oder die Gewichtungsmatrix nicht vorhanden oder beschädigt, wird diese erstellt. Die Erstellung erfolgt hierbei durch die Anwender*innen. Bei der Erstellung von Masken kann zwischen einer runden Maske, bei der lediglich der Radius vorgegeben werden muss, und einer durch Bereichsmarkierungen im Bild erstellten Maske gewählt werden. Wird bei der Schneeerkennung eine Pixelgewichtung gefordert, ist die Auswahl beziehungsweise Erstellung einer geeigneten Gewichtungsmatrix notwendig. Diese muss durch die Anwender*innen über Selektierung der horizontalen Ausdehnung zweier bodennaher Objekte gleicher Größe im Bild generiert werden.
Nach dem optionalen Einlesen beziehungsweise Erstellen der Gewichtungsmatrix sowie der Maske erfolgt das Laden der benötigten Netz-Modelle. Je nach Parametrierung der Station wird zusätzlich zum Modell für die semantische Segmentierung ein Modell zur Selektierung von nicht nutzbaren Bildern geladen. Je nach voreingestelltem Modus wird anschließend die einmalige Prediction aller Bilddaten eines Ordners durchgeführt oder ein Watchdog aktiviert, der Bilder beim Abspeichern in den eingestellten Ordner automatisch prädiziert. Die Ausgabe der Ergebnisse erfolgt abschließend in der für die Station definierten csv-Datei. Ein optionals Speichern der prädizierten semantisch segmentierten Bilder ist ebenfalls möglich.

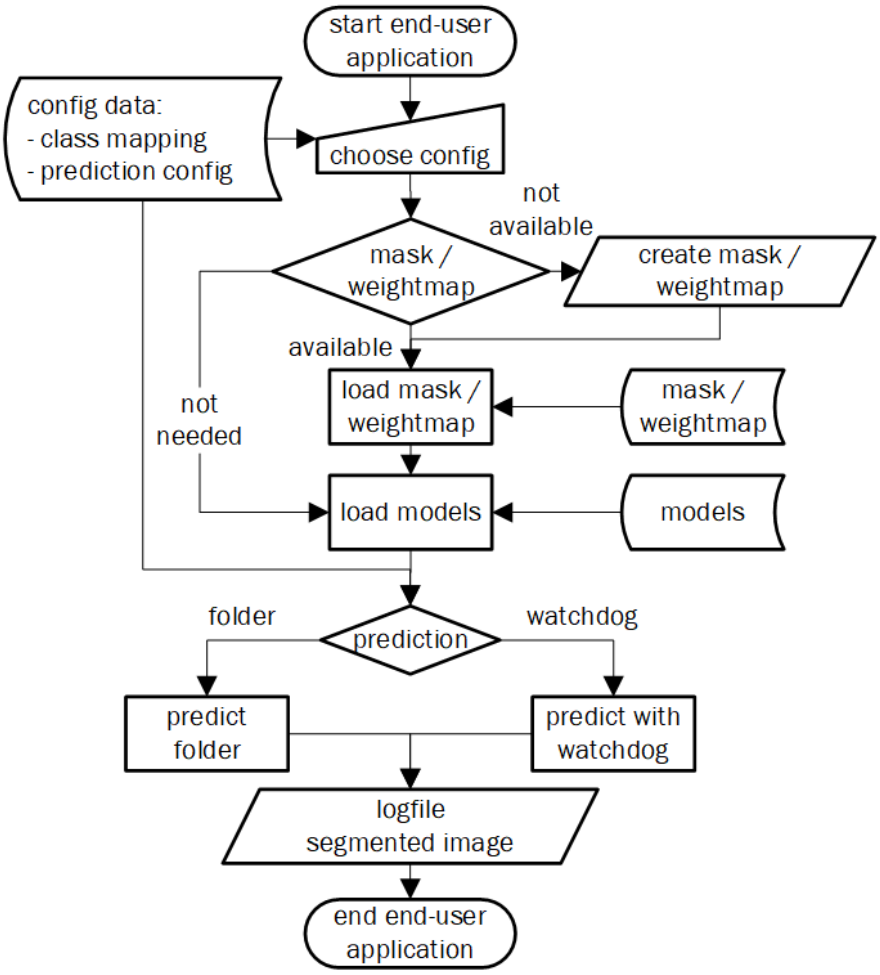


Abbildung 6.14: Programmablauf der Anwendung für Endnutzer*innen

7 Validierung

Dieses Kapitel behandelt die Ergebnisauswertung der Konzepte und Implementierungen der vorliegenden Arbeit. Dabei werden die im Konzept festgelegten Bewertungsparameter verwendet. Zunächst wird die Performance der ausgewählten Netz-Architekturen unter Verwendung übereinstimmender Datensätze verglichen, um objektiv eine Architektur für die weiteren Tests auszuwählen. Anschließend folgt die Validierung der Schnee- und Wolkenerkennung. Darüber hinaus wird eine selbst erhobene empirische Analyse zur Objektivierung der Ergebnisse der Schnee- und Wolkenerkennung ausgewertet. Darauf folgt die Validierung des Klassifikators für die Erkennung nicht nutzbarer Aufnahmen sowie die Laufzeitanalyse für die Segmentierung und Bedeckungsmessung. Abschließend erfolgt eine Auswertung der Anforderungserfüllung. Die Rahmenbedingungen für das Training der in diesem Abschnitt verwendeten neuronalen Netze sind in Kapitel 6.3 definiert.

7.1 Vergleich der Netz-Architekturen

Nachfolgend werden die ausgewählten Netz-Architekturen mit den Datensätzen der Schneeerkennung getestet, um festzulegen, welche der Architekturen im weiteren Verlauf der Arbeit verwendet wird. Die resultierende Netz-Architektur wird aufgrund der bereits thematisierten Ähnlichkeiten ebenfalls für die Wolkenerkennung verwendet.
Die Gegenüberstellung der vier Architekturen wird in Tabelle 7.1 veranschaulicht. Die Pixelwise Accuracy aus Tabelle 7.1a weist durchweg hohe Werte von mindestens 96 % auf. Die besten Ergebnisse erreicht mit 98,4 % im Durchschnitt das U-Net mit der MFFM Erweiterung sowie das eigens entwickelte KMTX-Net.
Der F1-Score aus Tabelle 7.1b ist insgesamt geringer als die Pixelwise Accuracy. Den geringsten Wert erzielt an dieser Stelle mit 70,8 % das U-Net mit der MFFM Erweiterung bei Datensatz D6 (*waternight*). Mit einem Durchschnitt von 89,7 % schneidet das KMTX-Net am besten ab. Es ist zu beachten, dass die Datensätze teilweise Testbilder enthalten, die Schneebedeckungen von 100 % oder 0 % aufweisen. Da in diesen Fällen jeweils

die Klasse ‚unbedeckter Boden' beziehungsweise ‚schneebedeckter Boden' nicht vertreten sind, gehen einzelne Pixelfehler in diesen Klassen beim F1-Score mit großer Gewichtung ein. Wird beispielsweise in einem Bild mit einer Schneebedeckung von 100 % ein Pixel mit der Klasse ‚unbedeckter Boden' prädiziert, fällt der micro F1-Score für diese Klasse unmittelbar auf 0 %, wodurch der macro F1-Score dementsprechend einen Maximalwert von 66,7 % im 3-Klassensystem erreichen kann. An dieser Stelle kommt es folglich auf eine möglichst exakte, klassentreue Prediction an.

Tabelle 7.1: Vergleich der ausgewählten Netz-Architekturen unter Verwendung der Datenstätze D3 (*muccam01*) bis D6 (*waternight*)

(a) Vergleich der Accuracy

Datensatz	**U-Net (MFFM)**	**UX-Net**	**DeepLab (MFFM / CSAM)**	**KMTX-Net**
D3 (muccam01)	98,0 %	97,8 %	97,2 %	97,8 %
D4 (garden)	99,1 %	99,0 %	98,4 %	98,9 %
D5 (waterday)	98,6 %	98,5 %	96,8 %	98,6 %
D6 (waternight)	97,9 %	97,8 %	98,4 %	98,1 %
Durchschnitt	98,4 %	98,3 %	97,7 %	98,4 %

(b) Vergleich des F1-Scores

Datensatz	**U-Net (MFFM)**	**UX-Net**	**DeepLab (MFFM / CSAM)**	**KMTX-Net**
D3 (muccam01)	90,0 %	87,6 %	82,2 %	90,1 %
D4 (garden)	92,3 %	91,1 %	87,5 %	90,2 %
D5 (waterday)	79,0 %	85,0 %	77,7 %	92,8 %
D6 (waternight)	70,8 %	86,1 %	89,3 %	85,7 %
Durchschnitt	83,0 %	87,4 %	84,2 %	89,7 %

Basierend auf diesen Ergebnissen wird das KMTX-Net als Netz-Architektur für das weitere Vorgehen in dieser Arbeit verwendet. Der Vergleich der Ergebnisse des UX-Nets mit den Ergebnissen des KMTX-Nets, welches auf dem UX-Net basiert, bestätigt die Vorhersage bezüglich der positiven Effekte der CSAM-Algorithmen von [Hongcai u. a., 2019].

7.2 Auswertung der Schneerkennung

Dieser Abschnitt behandelt die Auswertung der Schneerkennung. Für die Auswertung der semantischen Segmentierung werden zunächst Methoden des Domain Transfers untersucht. Eine nachfolgende Analyse auf mögliche Verbesserungen der Ergebnisse durch Reduzierung der Klassenanzahl komplettiert die Tests zur semantischen Segmentierung. Abschließend wird die Berechnung des Bedeckungsgrades inklusive Einordnung in die Referenzklassen validiert.

7.2.1 Domain Transfer bei der Schneeerkennung

Für den Domain Transfer bei der Schneeerkennung wird zunächst der Transfer auf unbekannte Szenerien getestet. Anschließend wird die Möglichkeit analysiert, ein neuronales Netz mit Trainingsdaten aller verfügbaren Domains zu trainieren. Abschließend wird die Einbindung neuer Domains in ein bestehendes System untersucht.

Transfer auf unbekannte Szenerien

Nachfolgend werden die Ergebnisse eines Transfers von bekannten auf unbekannte Szenerien ausgewertet. Hierbei werden in Tabelle 7.2 die F1-Scores der Tests mit den Datensätzen D3 (*muccam01*), D5 (*waterday*) sowie D4 (*garden*) dargestellt[19].
Sofern kein aktiver Domain Transfer stattfindet, wird lediglich mit *muccam01* trainiert und das resultierende neuronale Netz auf *waterday* und *garden* angewendet. Die Fortsetzung des mit *muccam01* durchgeführten Trainings mit der Szenerie *waterday* wird ebenfalls untersucht. Weiterhin wird ein neuronales Netz mit und ohne Domain-Klassifikator mit den Datensätzen *muccam01* und *waterday* trainiert. Der Datensatz *garden* stellt für jeden Testfall die unbekannte Domain dar und wird lediglich beim DATL für das Training des Domain-Klassifikators verwendet.
Tabelle 7.2 ist zu entnehmen, dass der F1-Score von unbekannten Domains in jedem Test geringer ist als jener bekannter Domains. Ein direkter Vergleich von *muccam01* und *waterday* im ersten Testfall zeigt eine Reduzierung um $29,3$ % auf $60,8$ %. Bei initialer Verwendung der Domains *waterday* und *muccam01* für das Training steigt der F1-Score von *waterday* um bis zu $25,9$ % auf $86,7$ %, während der F1-Score von *muccam01* um maximal $1,4$ % auf $88,7$ % sinkt. Bezogen auf die Ergebnisse des KMTX-Net aus Tabelle

[19] Die entsprechende Pixelwise Accuracy ist im Anhang in Tabelle C.1 dokumentiert.

7.1b, bei der jede Domain ein eigenes neuronales Netz erhält, ist eine Reduzierung des F1-Scores für *muccam01* und *waterday* von 1,1 % bis 7,1 % erkennbar.

Tabelle 7.2: F1-Scores der standortunabhängigen Schneeerkennung

Trainingsmethode/ Trainingsdaten	**D3 (muccam01)**	**D5 (waterday)**	**D4 (garden)**	**Durchschnitt**
D3 (muccam01)	90,1 %	60,8 %	57,1 %	69,3 %
D3 (muccam01), Trainingsfortsetzung mit D5 (waterday)	67,8 %	91,7 %	65,7 %	75,0 %
D3 (muccam01) und D5 (waterday)	88,7 %	85,7 %	65,1 %	79,9 %
DATL D3 (muccam01) und D5 (waterday)	89,0 %	86,7 %	71,7 %	82,4 %

Ein Vergleich des Trainings mit und ohne DATL in den unteren beiden Testfällen aus Tabelle 7.2 zeigt, dass das mit DATL trainierte neuronale Netz in den bekannten Domains geringe Steigerungen des F1-Scores von bis zu 1,0 % und in der unbekannten Domain eine Steigerung von 6,6 % erfährt. Dies lässt auf die erfolgreiche Extraktion von domaininvarianten Merkmalen schließen. In diesem Test wird außerdem der durchschnittlich höchste F1-Score von 82,4 % erreicht.
Zusätzlich wird der F1-Score der unbekannten Domain *garden* durch alle Methoden des Domain Transfers um mindestens 8,0 % gegenüber des Verzichts auf einen aktiven Domain Transfer verbessert. Durch die Verwendung von Trainingsdaten verschiedener Domains sind die antrainierten Merkmale robuster gegenüber unbekannter Domains.
Ebenfalls auffällig ist der hohe F1-Score des Datensatzes *waterday* bei der Trainingsfortsetzung. Dieser liegt in der Größenordnung des eigens für die Domain *waterday* trainierten Netzes aus Tabelle 7.1b. Das neuronale Netz verlernt bei der Trainingsfortsetzung jedoch größtenteils die Merkmale der Ursprungsdomain, weshalb der F1-Score der Ursprungsdomain ähnlich niedrig ist, wie der der unbekannten Domain. Dies korreliert mit den Ergebnissen aus [Perkovic, 2022]. Dort wird bei einer Trainingsfortsetzung ebenfalls ein Rückgang der Ergebnisse der Prediction bei der Ursprungsdomain dokumentiert.

Verwendung eines neuronalen Netzes für alle verfügbaren Domains

Ein weiteres Szenario des Domain Transfers ist die Verwendung eines neuronalen Netzes für alle verfügbaren Domains. Hierzu wird ein DATL-Netz mit allen verfügbaren Datensätzen trainiert. Die Ergebnisse der Pixelwise Accuracy und des F1-Scores werden in Tabelle 7.3 dargestellt.

Tabelle 7.3: Auswertung der Pixelwise Accuracy und des F1-Scores der Schneerkennung unter Nutzung sämtlicher verfügbarer Trainingsdaten aller Domains

Datensatz	**mit DATL**		**ohne DATL**	
	Acc	**F1-Score**	**Acc**	**F1-Score**
D3 (muccam01)	96,9 %	82,3 %	97,4 %	83,1 %
D4 (garden)	98,7 %	89,5 %	98,9 %	90,7 %
D5 (waterday)	97,7 %	77,2 %	98,2 %	79,8 %
D6 (waternight)[20]	98,6 %	78,7 %	99,3 %	74,5 %

Werden die Ergebnisse der Tabelle 7.3 mit denen aus Tabelle 7.1 verglichen, ist eine durchnittliche Reduzierung des F1-Scores um 7,8 % zu verzeichnen. Besonders auffällig ist die starke Reduzierung bei den Datensätzen *waterday* und *waternight* von bis zu 15,6 %. Das neuronale Netz ist somit nicht in der Lage, die Komplexität der Trainingsdaten in ausreichender Qualität zu erfassen.
Weiterhin lässt sich aus den Ergebnissen aus Tabelle 7.3 schlussfolgern, dass das DATL gegenüber aller in das Training eingebundenen Domains keine relevante Verbesserung mit sich bringt. Wie die unteren beiden Testfälle aus Tabelle 7.2 zeigen, beschränken sich die Verbesserungen durch das DATL auf die Ergebnisse bei unbekannten Domains.

Einbindung unbekannter Szenerien

In diesem Abschnitt werden die zuvor gewonnenen Erkenntnisse des Domain Transfers gegenübergestellt, um die verschiedenen Möglichkeiten der Einbindung unbekannter Szenerien darzustellen. Hierbei ist zu unterscheiden, ob aufbereitete Trainingsdaten für die

[20] Die Bilder werden für das Training dreikanalig eingelesen.

unbekannte Szenerie vorhanden sind, oder die Einbindung ohne den Aufwand der Trainingsdatenaufbereitung erfolgen soll.
Sofern für die neu einzubindende Domain keine annotierten Trainingsdaten vorhanden sind, liefert das DATL die vielversprechendsten Ergebnisse. Hierbei ist darauf zu achten, dass die Parameterzahl des neuronalen Netzes ausreichend für die Anzahl und Komplexität der Domains gewählt wird, um Underfitting zu vermeiden.
Sind Trainingsdaten für die unbekannte Domain vorhanden, werden die besten Ergebnisse bei der Verwendung eines neuronalen Netzes pro Domain erreicht. Aufgrund der ähnlichen Ergebnisse des Trainings mit nur einer Domain und der Trainingsfortsetzung werden deren Trainingsverläufe in Abbildung 7.1 gegenübergestellt. Weiterhin wird die Auswirkug der Verwendung eines Trainingsdatensatzes reduzierter Größe analysiert.

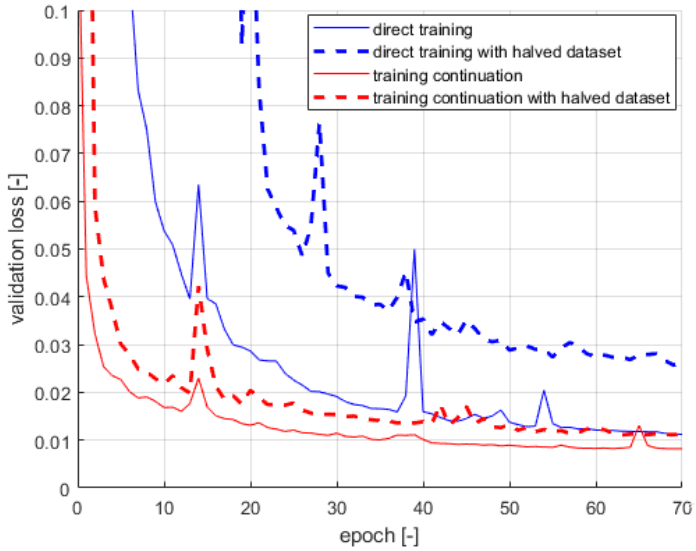


Abbildung 7.1: Trainingsverläufe der Domain *waterday* mit und ohne Nutzung eines für *muccam01* vortrainierten Modells

Abbildung 7.1 zeigt durch die initialisierten Gewichte eine deutlich schnellere Abnahme des Loss bei der Trainingsfortsetzung gegenüber dem direkten Training eines Modells mit einer Domain. Das Training kann entsprechend schneller durchgeführt werden, da eine Verringerung der Epochenanzahl ohne Qualitätseinbußen möglich ist.
Weiterhin sind in Abbildung 7.1 die Trainingsverläufe der Trainingsfortsetzung und des direkten Trainings mit halbierter Datensatzgröße dargestellt. Während bei der Trainingsfortsetzung der Loss ähnlich schnell sinkt wie bei der Verwendung des kompletten Daten-

satzes, ist die Reduzierung des Loss bei einem direkten Training mit halber Datensatzgröße deutlich langsamer und der Loss stagniert bei einem höheren Wert. Durch Halbierung des Datensatzes bei der Trainingsfortsetzung sinkt der F1-Score um $1,7$ % auf $90,0$ %, wobei durch ein direktes Training eine Reduktion um $4,9$ % auf $87,9$ % zu verzeichnen ist[21]. Folglich sind die Leistungseinbußen eines neuronalen Netzes durch Verringerung der Trainingsdatenanzahl wesentlich niedriger, sofern ein bestehendes Modell der Schneeerkennung für eine Trainingsfortsetzung genutzt wird.

7.2.2 Einfluss der Klassenanzahl bei der semantischen Segmentierung

In diesem Abschnitt wird untersucht, inwieweit die semantische Segmentierung durch die Verwendung von zwei statt drei Klassen verbessert werden kann. Hierbei muss das künstliche neuronale Netz ausschließlich zwischen den Klassen ‚schneebedeckter Boden' und ‚unbedeckter Boden' unterscheiden. Die Ergebnisse eines Trainings mit nur einer Domain sind in Tabelle 7.4 dargestellt.

Tabelle 7.4: Auswertung der semantischen Segmentierung bei Reduzierung auf das 2-Klassensystem anhand des Datensatzes D5 (*waterday*)

Klassensystem	**Acc**	**F1-Score**
3-Klassensystem	$98,6$ %	$92,8$ %
2-Klassensystem	$98,6$ %	$95,1$ %
2-Klassensystem mit Augmentations	$98,9$ %	$91,8$ %

Der direkte Vergleich des 2- und 3-Klassensystems aus Tabelle 7.4 zeigt einen Anstieg des F1-Scores um $2,3$ % auf $95,1$ %. Durch das Wegfallen der Klasse ‚Hintergrund' können sämtliche Parameter des neuronalen Netzes für die Unterscheidung von schneebedecktem Boden und unbedecktem Boden verwendet werden. Die Vorteile dieser Methode werden besonders während eines Domain Transfers ersichtlich. Dort stellt durch die fehlenden räumlichen Merkmale unbekannter Domains die Unterscheidung zwischen den Klassen ‚Hintergrund' und ‚unbedeckter Boden' eine größere Herausforderung dar als die Selektierung schneebedeckter Flächen (siehe Abb. 7.2).

[21] Die Ergebnisse sind in Tabelle C.2 im Anhang dokumentiert.

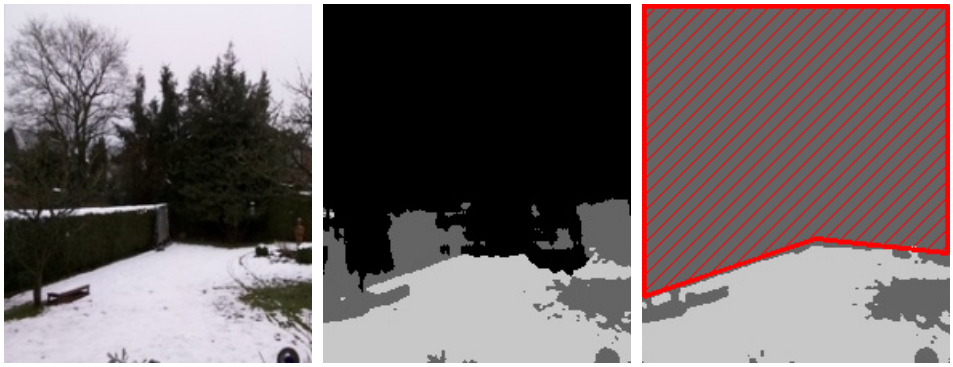


Abbildung 7.2: Vergleich der Prediction eines Bildes aus dem unbekannten Datensatz D4 (*garden*) (links) im 3-Klassensystem (mitte) und 2-Klassensystem (rechts) mit beispielhafter Maske (rot) unter Anwendung des Domain Adversarial Transfer Learning

Die Prediction der Schneeflächen sind im 3- und 2-Klassensystem ähnlich. Während im 2-Klassensystem eine Maske für die anschließende Berechnung des Bedeckungsgrades verwendet werden muss, verfälscht die fehlerhafte Segmentierung des Hintergrundes im 3-Klassensystem den resultierenden Bedeckungsgrad. Somit wird für das Beispiel aus Abbildung 7.2 im 3-Klassensystem bei einer Pixelwise Accuracy von $87,1$ % und einem F1-Score von $76,4$ % ein Bedeckungsgrad von $62,4$ % berechnet. Im 2-Klassensystem wird bei einer Pixelwise Accuracy von $98,8$ % und einem F1-Score von $93,7$ % ein Bedeckungsgrad von $87,7$ % ermittelt.
Abschließend soll festgehalten werden, dass bei einer Reduzierung der Klassenanzahl die Qualität der Segmentierungsergebnisse steigt, jedoch für die Berechnung des Bedeckungsgrades Masken erforderlich sind, sofern ein Bereich als Hintergrund gewertet werden muss. Eine Erweiterung des Datensatzes durch Augmentations bringt sowohl bei einer einzelnen Domain, als auch bei dem Domain Transfer keine Vorteile, solange ein Trainingsdatensatz ausreichender Größe vorhanden ist (siehe Tab. 7.4)[22].

7.2.3 Analyse der Bildklassifizierung durch Berechnung des Schneebedeckungsgrades

Für die Einordnung der semantisch segmentierten Bilder in die definierten Referenzklassen wird der Bedeckungsgrad der Prediction sowohl dem Bedeckungsgrad der Ground

[22] Die Ergebnisse des Domain Transfers unter Verwendung von Augmentations sind in Tabelle C.3 im Anhang dokumentiert.

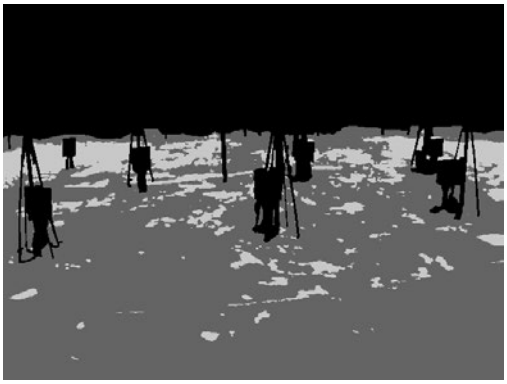
Truth (G.T.), als auch der manuellen Einordnung in die entsprechenden Referenzklassen (Ref.) gegenübergestellt und die Accuracy ausgewertet. Weiterhin wird der MAE zwischen den Bedeckungsgraden der Prediction und Ground Truth analysiert. In Tabelle 7.5 sind die Ergebnisse der Klassifizierung in die fünf Referenzklassen aus [DWD, 2021] dargestellt. Hierbei ist zu beachten, dass die Aufnahmen von *muccam01* und *garden* vorab in elf Referenzklassen eingeordnet sind und dadurch keine Accuracy für die Einordnung in fünf Referenzklassen ermittelbar ist.

Tabelle 7.5: Klassifizierungsergebnis von Accuracy (G.T. und Ref.) sowie MAE in fünf Referenzklassen mit und ohne Pixelgewichtung

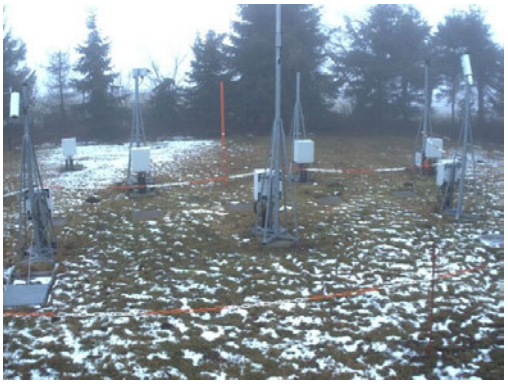
Datensatz	**ohne Pixelgewichtung**			**mit Pixelgewichtung**		
	G.T.	**Ref.**	**MAE**	**G.T.**	**Ref.**	**MAE**
D3 (muccam01)	94,9 %	–	1,7 %	94,9 %	–	1,9 %
D4 (garden)	97,7 %	–	1,1 %	97,7 %	–	1,1 %
D5 (waterday)	100,0 %	100,0 %	1,2 %	100,0 %	100,0 %	1,1 %
D6 (waternight)	96,6 %	93,1 %	2,5 %	96,6 %	96,6 %	2,5 %

Nach Tabelle 7.5 liegen die Klassifizierungsergebnisse bei allen Datensätzen gegenüber der Ground Truth gerundet zwischen 95 % und 100 %. Alle Aufnahmen des Datensatzes *waterday* können den richtigen Klassen zugeordnet werden. Bei *waternight* hingegen wird ein Bild fehlerhaft klassifiziert. Die Accuracy der manuellen Einordnung in die Referenzklassen ist bei *waterday* und *waternight* ähnlich gegenüber der Ground Truth. Die Klassifizierung weist einen MAE von maximal 2,5 % auf. Weiterhin ist auffällig, dass sich durch die Pixelgewichtung bei der Accuracy und dem MAE bezogen auf die Ground Truth kein und im Hinblick auf die manuelle Einordnung in die Referenzklassen nahezu kein Unterschied ergibt. Dies ist auf eine meist homogene Schneeverteilung zurückzuführen. Bei einer inhomogenen Schneeverteilung können sich Fehler der semantischen Segmentierung durch die Pixelgewichtung verschieden stark auswirken, was auch eine Rückkopplung auf die Klassifizierungsgenauigkeit hat. In Abbildung 7.3 wird der Einfluss der Pixelgewichtung auf den Bedeckungsgrad dargestellt.

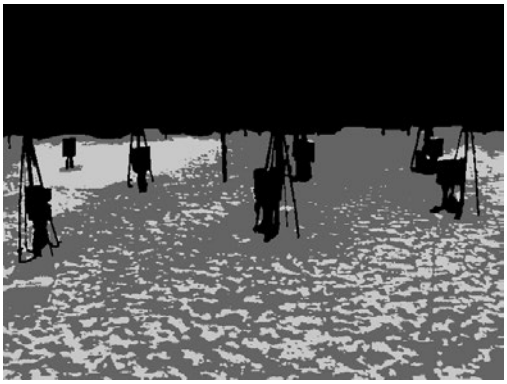
(a) Aufnahme der Wasserkuppe (Klassifiziert mit 10 % bis < 50 % Schneebedeckungsgrad)



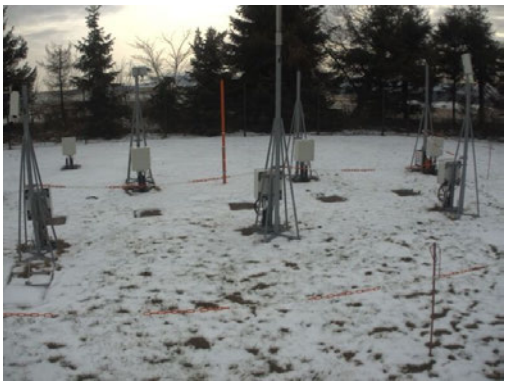
(b) Prediction zu (a) mit Schneebedeckungsgrad von 33,3 % mit und 25,6 % ohne Pixelgewichtung



(c) Aufnahme der Wasserkuppe (Klassifiziert mit 10 % bis < 50 % Schneebedeckungsgrad)



(d) Prediction zu (c) mit Schneebedeckungsgrad von 28,3 % mit und 29,6 % ohne Pixelgewichtung



(e) Aufnahme der Wasserkuppe (Klassifiziert mit 50 % bis < 100 % Schneebedeckungsgrad)

(f) Prediction zu (e) mit Schneebedeckungsgrad von 91,1 % mit und ohne Pixelgewichtung

Abbildung 7.3: Gegenüberstellung der Bedeckungsgradberechnung mit und ohne Pixelgewichtung bei unterschiedlicher Schneeverteilung

Abbildung 7.3 zeigt, dass die Unterschiede der Bedeckungsgrade bei annähernd homogener Schneeverteilung (hier: < 2 %) kleiner sind als bei inhomogener Schneeverteilung (hier: $7,7$ %).
Nach Analyse der vorliegenden Datensätze ist die Schneeverteilung meist annähernd homogen. Diese Aussage wird durch einen MAE von $0,3$ % bis $1,9$ % zwischen den Bedeckungsgraden mit und ohne Pixelgewichtung gestützt[23]. Somit kann die These aus [Strauss, 2007], dass bei ausreichender Homogenität der Bewölkung perspektivische Skalierung für die Berechnung des Wolkenbedeckungsgrades vernachlässigt werden kann, auf die Berechnung des Schneebedeckungsgrades übertragen werden.
Werden anschließend die Klassifizierungsergebnisse für elf Referenzklassen analysiert, sind diese bei der Ground Truth durchschnittlich um $7,1$ % geringer als die der Klassifizierung in fünf Referenzklassen (siehe Tab. 7.6). Hierbei ist zu beachten, dass die Aufnahmen der Wasserkuppe manuell in fünf Referenzklassen eingeordnet sind und dadurch keine Accuracy für die Einordnung in elf Referenzklassen ermittelbar ist.

Tabelle 7.6: Klassifizierungsergebnis der Accuracy (G.T. und Ref.) in elf Referenzklassen mit und ohne Pixelgewichtung

Datensatz	**ohne Pixelgewichtung**		**mit Pixelgewichtung**	
	G.T.	**Ref.**	**G.T.**	**Ref.**
D3 (muccam01)	$84,6$ %	$48,7$ %	$84,6$ %	$48,7$ %
D4 (garden)	$90,7$ %	$53,5$ %	$90,7$ %	$48,8$ %
D5 (waterday)	$92,3$ %	–	$92,3$ %	–
D6 (waternight)	$93,1$ %	–	$93,1$ %	–

Die reduzierte Accuracy ist durch die feinere Abstufung der Klassen zu begründen. So wird beispielsweise ein Bild mit einem Referenzbedeckungsgrad von $69,9$ % und einem berechneten Bedeckungsgrad von $70,1$ % bei fünf Referenzklassen korrekt und bei elf Referenzklassen fehlerhaft klassifiziert.
Abschließend werden die Klassifizierungsergebnisse des Domain Transfers mittels DATL analysiert. Die Ergebnisse bei Verwendung von fünf Referenzklassen und Pixelgewichtung sind in Tabelle 7.7 dargestellt. Die Datensätze D3 (*muccam01*) und D5 (*waterday*)

[23] Der MAE aller Domains ist in Tabelle C.4 im Anhang dokumentiert.

sind während des Trainings als bekannt und der Datensatz D4 (*garden*) als unbekannt deklariert[24].

Tabelle 7.7: Gegenüberstellung der Accuracy (G.T. und Ref.) sowie des MAE der Einordnung in fünf Referenzklassen unter Anwendung des Domain Adversarial Transfer Learning im 2- und 3-Klassensystem mit Pixelgewichtung

Datensatz	**3-Klassensystem**			**2-Klassensystem**		
	G.T.	**Ref.**	**MAE**	**G.T.**	**Ref.**	**MAE**
D3 (muccam01)	94,9 %	–	1,7 %	94,9 %	–	1,6 %
D4 (garden)	74,4 %	–	6,8 %	74,4 %	–	5,9 %
D5 (waterday)	100,0 %	100,0 %	1,2 %	100,0 %	100,0 %	1,1 %

Die Aufnahmen aus *waterday* können sowohl im 2-, als auch im 3-Klassensystem trotz reduziertem F1-Score beim DATL korrekt zugeordnet werden. Die Accuracy der Klassifizierung des Datensatzes *muccam01* ist bezogen auf die Ground Truth mit 94,9 % ebenfalls unverändert gegenüber der Klassifizierung mittels eines eigens für *muccam01* trainierten Netzes (siehe Tab. 7.5). Die Klassifizierung der unbekannten Domain *garden* erreicht eine Accuracy von 74,4 % bezogen auf die Ground Truth. Der MAE ist im 2-Klassensystem durchgehend geringer als im 3-Klassensystem. Dies bestätigt die verbesserten Segmentierungsergebnisse durch die Verwendung des 2-Klassensystems mit Maskierungen.

7.3 Auswertung der Wolkenerkennung

Für die Wolkenerkennung wird zunächst der Einfluss der Trainingsdatenanzahl auf die Ergebnisse der semantischen Segmentierung untersucht. Darüber hinaus wird der Domain Transfer getestet und schließlich die Klassifizierung des Bedeckungsgrades sowohl mit, als auch ohne DATL gegenübergestellt.

[24] Die entsprechenden Ergebnisse ohne Pixelgewichtung sind in Tabelle C.5 im Anhang dokumentiert. Weiterhin sind die Klassifizierungsergebnisse des DATL für alle Domains sowie die Zuordnung in elf Referenzklassen in Tabelle C.6 und C.7 im Anhang festgehalten.

7.3.1 Einfluss der Trainingsdatenanzahl

In diesem Kapitel wird der Einfluss der Trainingsdatenanzahl auf die Pixelwise Accuracy und den F1-Score analysiert. In Tabelle 7.8 wird zunächst der Einfluss der Trainingsdatenvervielfältigung durch Augmentations und Bildzerstückelung untersucht, um festzulegen, mit welcher der Methoden die weiteren Tests durchgeführt werden. Die Bildzerstückelung erfolgt mit einer relativen Bildgröße von 50 % der Originalgröße und 50 % Überlappung. Als Augmentations werden *Vertical-Flip*, *Horizontal-Flip* sowie *Rotate* in zwei verschiedenen zufälligen Winkeln genutzt. Dabei wird zur besseren Veranschaulichung die absolute Anzahl der Trainingsdaten angegeben. Es ist erkennbar, dass sich die Ergebnisse nicht proportional zur Anzahl der Trainingsdaten verbessern. Dennoch erzielt die Kombination aus Bildzerstückelung und Augmentations mit 89,2 % den höchsten F1-Score. Die Pixelwise Accuracy ist bei der Bildzerstückelung ohne Augmentations um 0,3 % höher.

Tabelle 7.8: Einfluss von Bildzerstückelung und Augmentations bei der Wolkenerkennung (Datensatz: D8 (*Vivothek*), Augmentations, Bildzerstückelung)

Verfahren	**Anzahl Trainingsdaten**	**Acc**	**F1-Score**
keine Datenvervielfältigung	32	85,6 %	70,0 %
Augmentations	160	95,5 %	83,8 %
Bildzerstückelung	288	97,2 %	86,7 %
Bildzerstückelung und Augmentations	1440	96,9 %	89,2 %

Wird der Trainings-Loss für die unteren drei Testfälle aus Tabelle 7.8 in die Betrachtung mit einbezogen, ist eine weitere Entwicklung zu erkennen. Der Loss während des Trainings ist mit Minimalwerten von 4,64 % mit Augmentations, 5,21 % mit Bildzerstückelung und 3,96 % mit der Kombination beider Verfahren ähnlich niedrig. Bei der Bildzerstückelung ist der Loss höher als bei der Nutzung von Augmentations, obwohl sich der F1-Score bei den Testbildern genau entgegengesetzt verhält. Das hängt damit zusammen, dass die Generalisierungsleistung der neuronalen Netze nicht zwangsweise mit den Ergebnissen der Trainingsdaten korreliert. Die Bildzerstückelung sowie die Augmentations wirken hier dem Overfitting des neuronalen Netzes entgegen und verhindern, dass das Netz den Wolkenvorkommen lokale Bezüge zuordnen kann. Auf diese Weise

wird bei minimalen Änderungen des Trainings-Loss die Generalisierungsleistung verbessert und so der F1-Score bei den Testbildern gesteigert. Aufgrund des verhältnismäßig hohen F1-Scores bei der Kombination von Bildzerstückelung und Augmentations wird diese Methode für die weiteren Tests angewendet.
Der Einfluss der Anzahl der aufbereiteten Trainingsbilder auf die Prediction der Testbilder bei gleichbleibender Augmentation und Bildzerstückelung wird in Abbildung 7.4 dargestellt[25]. Für diesen Test werden beide Datensätze der Wolkenerkennung genutzt, um eine möglichst große Reichweite bei der Trainingsdatenanzahl zu erreichen. Der Testdatensatz ist in jedem Fall identisch. Im dargestellten Bereich ist ein klarer Aufwärtstrend sowohl beim F1-Score, als auch bei der Pixelwise Accuracy erkennbar. Der F1-Score steigt dabei insgesamt schneller an. Daraus lässt sich in diesem Bereich ein deutliches Potential bei der Verbesserung der Prediction der neuronalen Netze durch Aufbereitung weiterer Trainingsbilder ableiten.

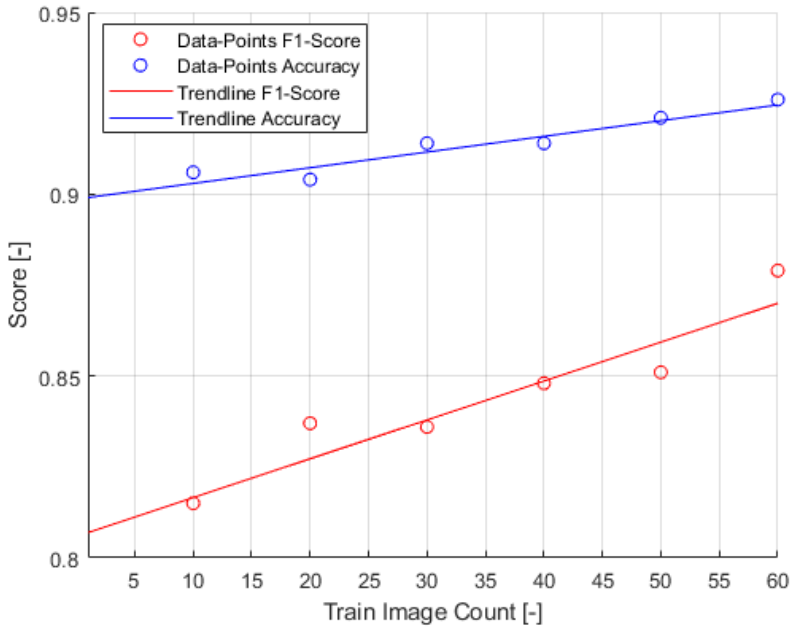


Abbildung 7.4: Entwicklung der Pixelwise Accuracy und des F1-Scores über die Anzahl der Trainingsbilder

[25] Die zugrunde liegenden Daten sind in Tabelle C.8 im Anhang dokumentiert.

7.3.2 Domain Transfer bei der Wolkenerkennung

Der Domain Transfer wird standortunabhängig auf Kameraebene untersucht. Dafür werden sowohl neuronale Netze mit beiden Trainingsdatensätzen trainiert, als auch der Transfer auf unbekannte Domains untersucht, indem für das Training nur aufbereitete Trainingsdaten eines Datensatzes verwendet werden.
In den beiden ersten Tests aus Tabelle 7.9 stellt der Datensatz D7 (*Mobotix*) für das Segmentierungsnetzwerk eine unbekannte Domain dar. Das mit Datensatz D8 (*Vivothek*) trainierte DATL-Netz besitzt lediglich einen mit Bildern beider Domains trainierten Domain-Klassifikator. Erkennbar ist, dass sich die Ergebnisse der Accuracy und des F1-Scores bei der unbekannten Domain *Mobotix* durch das DATL deutlich verbessern. Gleichzeitig ist jedoch ein Rückgang der Segmentierungsleistung bei der bekannten Domain *Vivothek* zu verzeichnen, sodass sich im Durchschnitt lediglich eine Verbesserung der Pixelwise Accuracy von $1,6$ % und des F1-Scores von $0,4$ % ergibt. Folglich wirkt sich die Unterdrückung domainspezifischer Merkmale in diesem Fall negativ auf das Training der Domain *Vivothek* aus.
Stehen aufbereitete Trainingsdaten für beide Domains zur Verfügung, lassen sich die Ergebnisse des Domain Transfers noch weiter verbessern. Wird für den Domain Transfer ein mit Datensatz D8 (*Vivothek*) trainiertes Modell mit Datensatz D7 (*Mobotix*) weitertrainiert, werden Merkmale der Domain *Vivothek* teilweise wieder verlernt, was zu einem Rückgang der Pixelwise Accuracy um $4,7$ % und des F1-Scores um $6,2$ % führt. Bei der Domain *Mobotix* steigt jedoch die Pixelwise Accuracy um $8,0$ % und der F1-Score um $14,5$ %, wodurch sich das Gesamtergebnis bezogen auf das Training mit nur einer Domain um bis zu $1,6$ % bei der Pixelwise Accuracy und $4,1$ % beim F1-Score verbessert. Bei Berücksichtigung der guten Ergebnisse bezogen auf die Domain *Mobotix* und eine durch die initialisierten Gewichte geringe Trainingsdauer (siehe Abb. 7.1 der Schneeerkennung) ist dieses Verfahren sehr gut geeignet für einen Domain Transfer, bei dem aufbereitete Trainingsdaten zur Verfügung stehen und das neuronale Netz ausschließlich für die neue Domain genutzt wird.

Tabelle 7.9: Ergebnisse verschiedener Tests zum Domain Transfer

Trainingsmethode / Trainingsdaten	**D8 (Vivothek)**		**D7 (Mobotix)**		**Durchschnitt**	
	Acc	**F1-Score**	**Acc**	**F1-Score**	**Acc**	**F1-Score**
D8 (Vivothek)	96,8 %	88,5 %	86,3 %	74,1 %	91,6 %	81,3 %
DATL D8 (Vivothek)	93,6 %	85,8 %	92,8 %	77,6 %	93,2 %	81,7 %
D8 (Vivothek), Trainingsfortsetzung mit D7 (Mobotix)	92,1 %	82,3 %	94,3 %	88,6 %	93,2 %	85,4 %
D7 (Mobotix) und D8 (Vivothek)	93,8 %	87,1 %	91,4 %	88,6 %	92,6 %	87,9 %
DATL D7 (Mobotix) und D8 (Vivothek)	94,4 %	85,4 %	91,9 %	89,8 %	93,1 %	87,6 %

Mit bis zu 87,9 % erzielen die neuronalen Netze, die initial mit beiden Domains trainiert werden, im Durchschnitt die besten F1-Scores. Dabei bringt das DATL keine Verbesserung des F1-Scores. Eine Verbesserung des F1-Scores durch das DATL bei unbekannten Domains ist dennoch zu erwarten. Begründen lässt sich diese These durch die sichtbare Verbesserung der Ergebnisse bei der unbekannten Domain *Mobotix* bei Nutzung des DATL im zweiten Testfall aus Tabelle 7.9 und die Erkenntnisse des Domain Transfers der Schneeerkennung in Kapitel 7.2.1. Die Pixelwise Accuracy ist bei einem Training mit aufbereiteten Daten beider Domains mit einer maximalen Abweichung von 0,6 % nahezu konstant. Somit eignen sich diese Methoden für neuronale Netze, die nach dem Domain Transfer gleichermaßen für beide Domains genutzt werden.
Basierend auf der These aus Kapitel 5.7.2, dass Wolken Ähnlichkeiten zu Schneeflecken aufweisen, können die neuronalen Netze aus der Schneeerkennung als Basis für das Training der neuronalen Netze zur Wolkenerkennung genutzt werden. Abbildung 7.5 zeigt, dass sich bei einer Trainingsfortsetzung mit einem Modell der Schneeerkennung und den Datensätzen D7 (*Mobotix*) und D8 (*Vivothek*) bereits in der ersten Epoche ein sehr geringer Validation-Loss von unter 7 % einstellt. Darüber hinaus läuft das Training insgesamt stabiler, da der Validation-Loss während des Trainings in einzelnen Epochen weniger stark ansteigt.

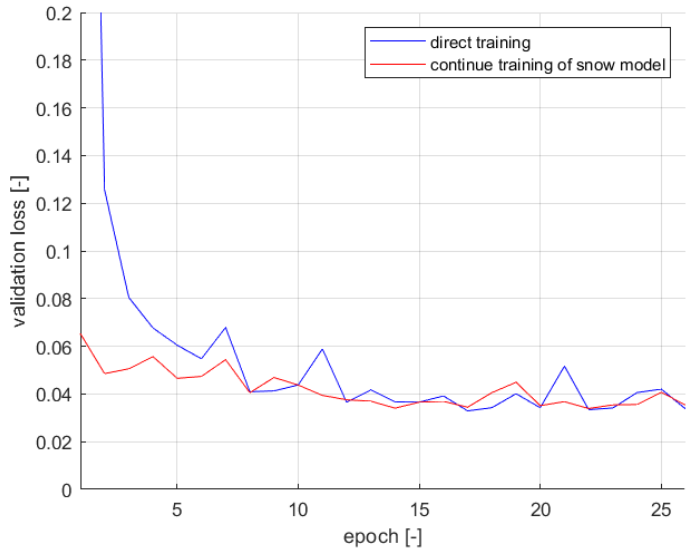
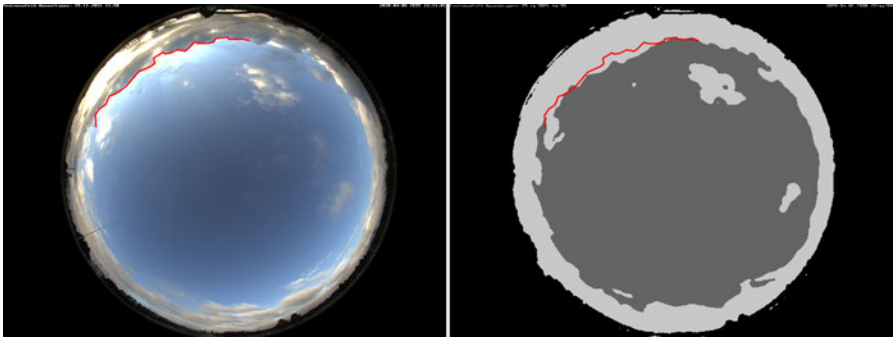


Abbildung 7.5: Gegenüberstellung des Trainingsverlaufs bei Nutzung eines untrainierten Modells und eines für die Schneeerkennung vortrainierten Modells

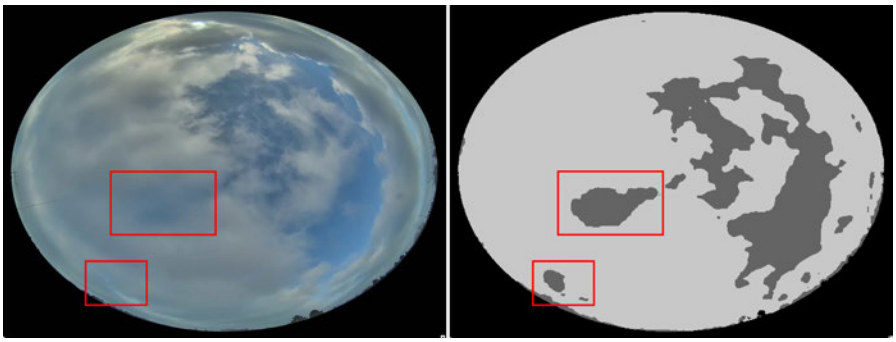
Stehen für das Training nur geringe Rechenkapazitäten zur Verfügung, kann so innerhalb weniger Trainingsepochen ein funktionierendes Modell für die Wolkenerkennung trainiert werden. Bei größerer Epochenanzahl gleichen sich die Trainingsverläufe jedoch an, was sich auch in den Ergebnissen der Testdaten widerspiegelt. Das so trainierte Modell erreicht ohne DATL eine Pixelwise Accuracy von $92,9$ % und einen F1-Score von $88,5$ %. Verglichen mit dem equivalenten Training ohne vortrainiertem Modell der Schneeerkennung aus Tabelle 7.9 wird eine minimale Verbesserung des F1-Scores um $0,6$ % erzielt. In Abbildung 7.6 werden beispielhafte Ergebnisse der Prediction eines DATL-Netzes gezeigt. Das mit der Kamera eines Mobiltelefons[26] aufgenommene Bild aus Abbildung 7.6c repräsentiert die unbekannte Domain. Dieses Bild unterscheidet sich zu den Trainingsbildern in Darstellung und Auflösung. Außerdem ist von Unterschieden bei Bildrauschen und Kontrastverhältnissen auszugehen. Das Ergebnis der Prediction erscheint optisch logisch, was auf einen erfolgreichen Domain Transfer schließen lässt. Abbildung 7.6a zeigt, dass noch Potential bei der Unterscheidung von direkter Sonneneinstrahlung und Wolken vorhanden ist. Dort werden in der oberen Bildhälfte durch die Sonneneinstrahlung Flächen fälschlicherweise als Wolken segmentiert. In Abbildung 7.6b werden in der linken Bildhälfte Flächen als unbedeckt segmentiert, bei denen Wolken erkennbar sind, jedoch

[26] Genutzt wird ein Huawei P20 Pro mit Leica Triple-Kamera.

der Himmel sichtbar durchscheint. Solche Bildflächen lassen einen gewissen Interpretationsspielraum. Bei einer strikten Durchsetzung der Wolkendefinition aus Kapitel 5.7.1 sind diese Bereiche jedoch fehlerhaft segmentiert.



(a) Mobotix Testbild (links) mit Prediction (rechts) (geschätzte Wolkengrenze Rot markiert)



(b) Vivothek Testbild (links) mit Prediction (rechts) (falsch klassifizierte Flächen Rot markiert)

(c) Testbild aufgenommen mit der Kamera des Mobiltelefons (links) mit Prediction (rechts)

Abbildung 7.6: Ergebnisse der Prediction eines mit den Datensätzen D7 (*Mobotix*) und D8 (*Vivothek*) trainierten Modells unter Verwendung des Domain Adversarial Transfer Learning

7.3.3 Analyse der Bildklassifizierung durch Berechnung des Wolkenbedeckungsgrades

Die Bewertung der Bedeckungsgradmessung erfolgt zum einen anhand der Accuracy der Einordnung in die fünf Referenzklassen der Wolkenbedeckung und zum anderen durch den klassenunabhängigen MAE. Da es für die Datensätze der Wolkenerkennung keine vorangegangene manuelle Einordnung in die Referenzklassen gibt, werden die Bewertungskriterien lediglich an der Ground Truth gemessen. Die Tests zur Bedeckungsgradmessung aus Tabelle 7.10 sind angelehnt an die Tests zum Domain Transfer aus Tabelle 7.9.

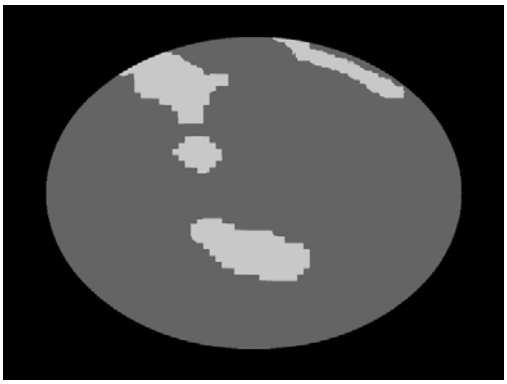
Tabelle 7.10: Accuracy und MAE der Wolkenbedeckungsmessung

Trainingsmethode / Trainingsdaten	**D8 (Vivothek)**		**D7 (Mobotix)**		**Durchschnitt**	
	Acc	**MAE**	**Acc**	**MAE**	**Acc**	**MAE**
D8 (Vivothek)	75,0 %	6,5 %	36,4 %	16,5 %	54,8 %	11,9 %
DATL D8 (Vivothek)	60,0 %	11,6 %	72,7 %	7,9 %	66,7 %	9,7 %
D8 (Vivothek), Trainingsfortsetzung mit D7 (Mobotix)	60,0 %	10,3 %	81,8 %	5,1 %	71,4 %	7,5 %
D7 (Mobotix) und D8 (Vivothek)	70,0 %	10,9 %	86,4 %	8,0 %	78,6 %	9,3 %
DATL D7 (Mobotix) und D8 (Vivothek)	60,0 %	10,4 %	86,4 %	6,7 %	73,8 %	8,5 %

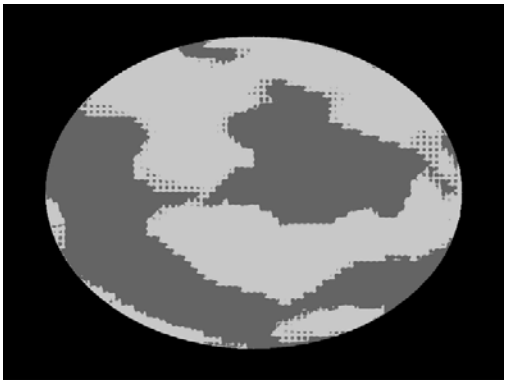
Für den ersten Test wird ein Modell lediglich mit Datensatz D8 (*Vivothek*) trainiert und auf beiden Domains getestet. Dabei fällt auf, dass durch das Training mit nur einer Domain bei dieser ein domainspezifischer MAE von 6,5 % erzielt wird. Hierbei stellt sich bei der unbekannten Domain *Mobotix* mit 16,5 % der von allen Tests höchste MAE ein. Auch die Accuracy ist mit 36,4 % sehr niedrig. Durch Verwendung des DATL verbessern sich die Ergebnisse bei der für die Segmentierung unbekannten Domain *Mobotix* signifikant. Der MAE sinkt um mehr als die Hälfte und die Accuracy wird verdoppelt. Wie bei der Auswertung des Domain Transfers bereits erläutert, führt das DATL dabei zu einer

Verschlechterung der Ergebnisse bei der für das Training genutzten Domain *Vivothek*. Im Durchschnitt führt das DATL jedoch zu einer deutlichen Verbesserung der Wolkenbedeckungsmessung.
Die besten Durchschnittsergebnisse der Bedeckungsmessung werden bei Nutzung von Trainingsdaten beider Domains erzielt. Auffällig ist, dass der niedrigste durchschnittliche MAE bei der Trainingsfortsetzung erreicht wird. Maßgeblich verantwortlich dafür ist der niedrige MAE der Domain *Mobotix* von $5,1\,\%$, welcher außerdem der niedrigste domainspezifische MAE in dieser Testreihe ist. Allgemein sind die Ergebnisse der Domain *Mobotix* in beiden Bewertungskriterien besser als die der Domain *Vivothek*. Mögliche Erklärungen dafür sind beispielsweise Abweichungen bei der Qualität der aufbereiteten Trainingsdaten oder gegebenenfalls leichter zu erlernende Merkmale durch geeignetere Kontrastverhältnisse bei den Bildern der Domain *Mobotix*.
Die höchste Accuracy erreicht im Durchschnitt das initial mit beiden Domains ohne DATL trainierte Modell. Die beste domainspezifische Accuracy wird mit $86,4\,\%$ in den letzten beiden Tests aus Tabelle 7.10 für die Domain *Mobotix* erreicht. Das ist auch die höchste Accuracy, die mit lediglich 60 Trainingsbildern in dieser Testreihe erreicht wird. Ein Vergleich der Durchschnittsergebnisse der letzten beiden Tests zeigt die Unabhängigkeit des MAE von den Referenzklassen. Dort sind die Entwicklungen der Accuracy und des F1-Scores nicht gegenläufig, wie sie es beispielsweise bei den Durchschnittsergebnissen der ersten beiden Tests sind. Treten Fehler häufig bei Bildern auf, deren Bedeckungsgrad nah an einer Grenze zwischen zwei Klassen liegt, kann die Accuracy genau wie bei der Schneeerkennung auch bei niedrigem MAE verhältnismäßig gering sein.
Alle Ergebnisse sind vor dem Hintergrund zu betrachten, dass auch Ungenauigkeiten bei der händischen Aufbereitung der Testdaten, an denen die Predictions der neuronalen Netze gemessen werden, auftreten können. Bei der Klassifizierung von Wolken spielt die subjektive Wahrnehmung der Person, die die Daten aufbereitet, ebenfalls eine Rolle. Abbildung 7.7 zeigt ein Beispiel, bei dem die Einschätzung der Wolkenbedeckung des neuronalen Netzes deutlich von der Einschätzung der Person, die das Testbild aufbereitet hat, abweicht. Das Problem ist, dass das Ziehen der Grenze zwischen wolkenbedecktem und unbedecktem Himmel bei weichen Übergängen sehr schwierig ist. Auch Faktoren wie beispielsweise Kontrastverhältnisse und Helligkeit des für die Datenaufbereitung verwendeten Bildschirms können einen Einfluss auf die Qualität der Testdaten haben. Der Einfluss dieses Problems kann voraussichtlich durch eine größere Menge aufbereiteter Trainingsbilder sowie die Objektivierung durch die Datenaufbereitung mehrerer Personen relativiert werden.

(a) Original Testbild Vivothek



(b) Händisch erstellte Ground Truth zu (a) mit Wolkenbedeckungsgrad 11,5 %



(c) Prediction des künstlichen neuronalen Netzes zu (a) mit Wolkenbedeckungsgrad 50,5 %

(d) Farblich gekennzeichneter Fehler zwischen (b) und (c)

Abbildung 7.7: Beispielhafte Fehlerdarstellung zwischen Prediction und Ground Truth

7.4 Empirische Validierung der semantischen Segmentierung

In diesem Abschnitt werden die experimentell ermittelten Ergebnisse der semantischen Segmentierung empirisch validiert. Durch die manuelle Erstellung der Ground Truth, die stets als Referenz bei der Auswertung der semantischen Segmentierung herangezogen wird, unterliegt diese nach [Jacob u. a., 2021] einer Fehlerquote. Sofern diese Fehler nicht in die Prediction der neuronalen Netze übertragen werden, wirkt sich dies negativ auf die Accuracy und den F1-Score aus. Um die Qualität der Prediction dennoch objektiv

validieren zu können, wurde eine Online-Umfrage[27] durchgeführt. In dieser Umfrage wurden im ersten Teil Fragen zur Person gestellt, um die Teilnehmer*innen mit relevanter Vorbildung separat analysieren zu können. Als relevante Vorbildung sind Kenntnisse in den Bereichen Bildverarbeitung, semantische Segmentierung und künstliche neuronale Netze definiert.
Im zweiten Teil wurde je ein Eingangsbild und die dazugehörige Ground Truth sowie das Ergebnis der Prediction gezeigt. Die Teilnehmer*innen mussten sich hier entscheiden, welche semantische Segmentierung das Eingangsbild besser repräsentiert oder ob alternativ beide gleichwertig sind. Die Umfrage umfasst 25 Bilder, von denen zehn dem Datensatz *waterday*, fünf dem Datensatz *waternight* und zehn den Datensätzen *Mobotix* und *Vivothek* angehören.
Die Anzahl der Umfragerückläufer beläuft sich auf 141 Teilnehmer*innen, von denen 41 Teilnehmer*innen mindestens eine relevante Vorbildung angegeben haben. Für eine objektive Auswertung der Umfrage wird die Fehlermarge ϵ auf Basis der Stichprobengröße n und der Sicherheitswahrscheinlichkeit z sowie der Populationsgröße P nach [Mossig, 2012] berechnet (siehe Gl. 7.1).

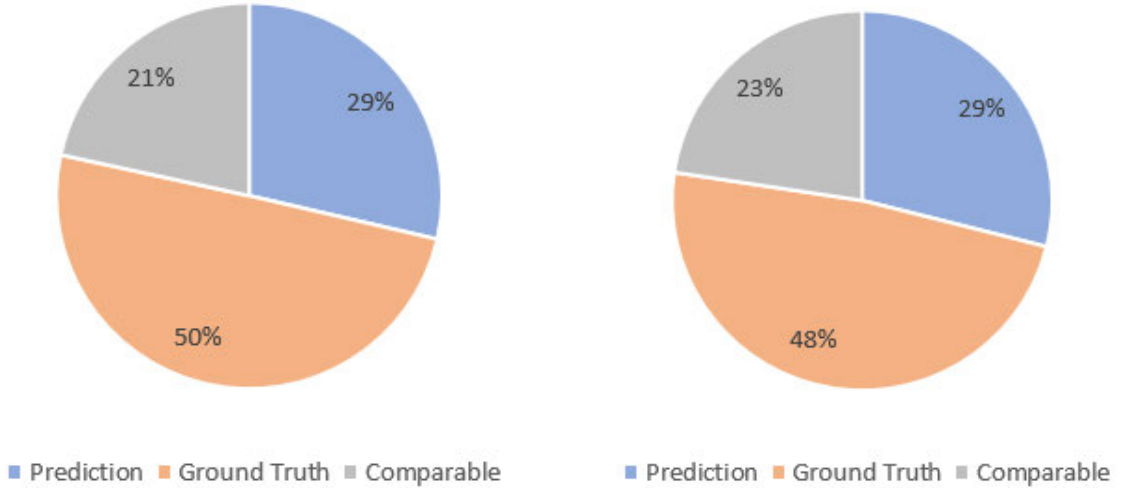
$$n \geq z^2 \cdot \frac{P \cdot (1 - P)}{\epsilon^2} \tag{7.1}$$

Der Wert für die Sicherheitswahrscheinlichkeit z wird zu $1,645$ gewählt. Hiermit wird festgesetzt, dass 90 % der Ergebnisse in der Fehlermarge ϵ liegen. Da die Populationsgröße in diesem Fall unbekannt ist, wird $P = 0,5$ gewählt, sodass n den größtmöglichen Wert annimmt, um auch im ungünstigsten Fall eine hinreichende Stichprobengröße zu erreichen (vgl. [Mossig, 2012]).
Es ergibt sich bei Auswertung aller Umfragerückläufer nach Gleichung 7.1, durch die Variation von n, eine Fehlermarge von 7 % beziehungsweise 13 % bei Auswertung der Teilnehmer*innen mit relevanter Vorbildung. Weiterhin erfüllen beide Gruppen die minimale Teilnehmeranzahl von 30 Personen (vgl. [Mossig, 2012]), sodass im Folgenden eine valide Auswertung beider Gruppen unter Annahme der berechneten Fehler möglich ist.

Die Gesamtergebnisse der Umfrage sind in Abbildung 7.8 dargestellt. Hier ist ersichtlich, dass die Verteilung der Ergebnisse zwischen den separat ausgewerteten Gruppen

[27] Umfrageportal: `https://www.umfrageonline.com/` mit einem Umfragezeitraum von 2 Kalenderwochen.

annähernd gleich ist. Sofern Comparable gewählt wird, repräsentieren sowohl die Prediction als auch die Ground Truth das Eingangsbild in gleicher Qualität.



(a) Ergebnis aller Teilnehmer*innen

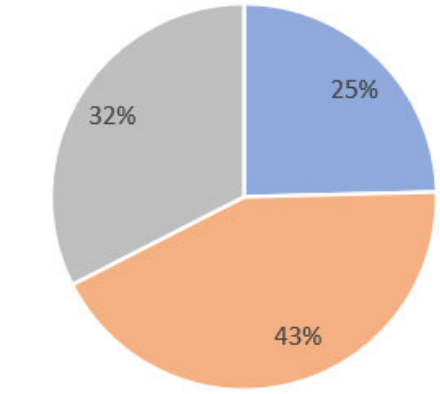
(b) Ergebnis der Teilnehmer*innen mit relevanter Vorbildung

Abbildung 7.8: Ergebnis der empirischen Analyse zur Schnee- und Wolkenerkennung

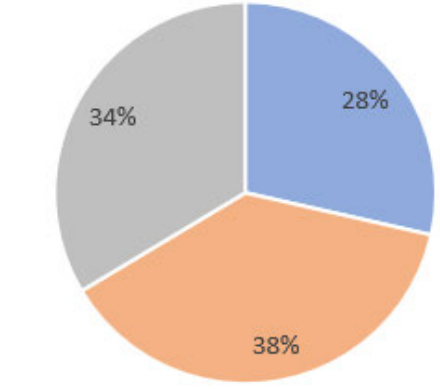
Weiterhin zeigt Abbildung 7.8, dass in maximal 50 % der Fälle die Ground Truth als repräsentativer bewertet wird. Folglich wird in mindestens 50 % der Fälle das Ergebnis der Prediction als besser beziehungsweise gleichermaßen präzise angesehen.

Werden anschließend die Ergebnisse der Schnee- und Wolkenerkennung in Abbildung 7.9 verglichen, ist ersichtlich, dass bei der Schneeerkennung die Ground Truth in geringerem Maße als repräsentativer gegnüber der Prediction eingestuft wird, als bei der Wolkenerkennung[28]. Dies korreliert mit den bei der Validierung der Schnee- und Wolkenerkennung ermittelten Ergebnissen der verwendeten neuronalen Netze, welche bei der Schneeerkennung einen F1-Score von 92,8 % und bei der Wokenerkennung einen F1-Score von 87,9 % aufweisen. Weiterhin ist Abbildung 7.9 zu entnehmen, dass trotz insgesamt höherer Repräsentativität der Ground Truth, der Anteil der Stimmen für die Prediction bei der Wolkenerkennung mit bis zu 35 % höher ist als bei der Schneeerkennung.

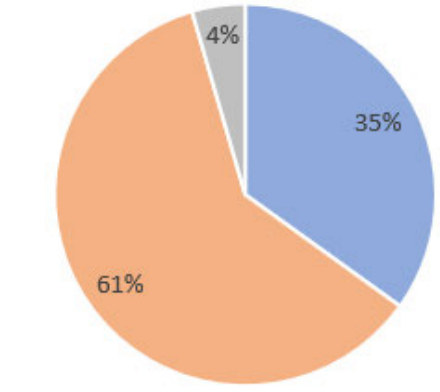
[28] Die Aufteilung in die Szenerien *waterday* und *waternight* ist in Anhang D dokumentiert.



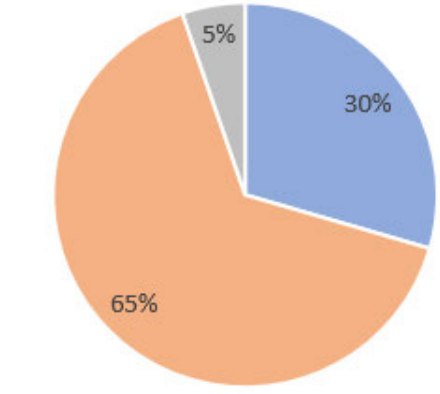
(a) Ergebnis aller Teilnehmer*innen bei der Schneeerkennung



(b) Ergebnis der Teilnehmer*innen mit relevanter Vorbildung bei der Schneeerkennung



(c) Ergebnis aller Teilnehmer*innen bei der Wolkenerkennung



(d) Ergebnis der Teilnehmer*innen mit relevanter Vorbildung bei der Wolkenerkennung

Abbildung 7.9: Ergebnis der empirischen Analyse unterteilt in Schnee- und Wolkenerkennung

Deutlich seltener wurden Ground Truth und Prediction bei der Wolkenerkennung als gleichwertig repräsentativ eingeordnet. Dies untersteicht die These aus Kapitel 7.3.3, dass eine einheitliche Wolkendefinition bei der Datenaufbereitung schwer umzusetzen ist. In Hinsicht auf das in Abbildung 7.9b dargestellte Umfrageergebnis wird geschlussfolgert, dass die Prediction des neuronalen Netzes der Schneeerkennung in nahezu vergleichbarer

Qualität zur manuellen semantischen Segmentierung arbeitet. Eine Nutzung der, in der vorliegenden Arbeit entwickelten, Lösungen für die Ermittlung der Bedeckungsgrade kann somit als Alternative zu der manuellen Ermittlung herangezogen werden.

7.5 Auswertung der Klassifikation nicht nutzbarer Aufnahmen

Die Klassifikatoren zur Aussortierung von Bildern, die für die semantische Segmentierung nicht nutzbar sind, werden unabhängig für die Schnee- und Wolkenerkennung trainiert und getestet. Bei der Schneeerkennung wird der Datensatz D9 der Szenerie *muccam01* genutzt, welcher zusätzlich zu nutzbaren Aufnahmen überlichtete Bilder enthält. Der Klassifikator für die Wolkenerkennung wird mit einem Datensatz D10 trainiert, der Bilder der Kamera Mobotix Q26 sowie der Kamera Vivothek FE9381-EHV enthält. Die nicht nutzbaren Aufnahmen sind hier zum einen Bilder bei Dunkelheit und zum anderen Bilder mit Regentropfen auf dem Kameraobjektiv. Zusätzlich wird der Testdatensatz D11 mit Bildern der Kamera des verwendeten Mobiltelefons genutzt. Dieser Datensatz besteht aus nutzbaren Tagaufnahmen sowie aus Aufnahmen, die für die semantische Segmentierung zu dunkel sind. Bei dem Training ist es wichtig, dass alle Trainingsdaten einer Domain die selbe Vorverarbeitung aufweisen. Andernfalls werden dem Klassifikator für die Unterscheidung nutzbarer und nicht nutzbarer Aufnahmen Merkmale bedingt durch Vorverarbeitungen wie beispielsweise Maskierungen antrainiert, was die Funktion des Klassifikators bei Anwendung auf gleichermaßen vorverarbeitete oder unvorverarbeitete Aufnahmen beeinträchtigt.

Tabelle 7.11 zeigt die Ergebnisse der Klassifikation bei einem Schwellwert für die Nutzbarkeit der Aufnahmen von $0,5$ (50 %). Bei diesem Schwellwert wird in jedem Test eine Accuracy von 100 % erreicht. Den niedrigsten MAE erreicht der Klassifikator der Wolkenerkennung. Diesem stehen dabei deutlich mehr Trainingsdaten zur Verfügung, als dem Klassifikator der Schneeerkennung. Der Klassifikator der Wolkenerkennung funktioniert darüber hinaus auch bei den Aufnahmen des Mobiltelefons, welche eine unbekannte Domain darstellen. Durch die Nutzung der Bildzerstückelung für das Training des Klassifikators ist dieser in der Lage, auch Bilder ohne die dunklen Rahmen der Trainingsbilder erfolgreich zu klassifizieren.

Tabelle 7.11: Auswertung des Klassifikators zur Selektion von nicht nutzbaren Bildern

Datensatz	Acc	MAE
D9 (muccam01)	100,0 %	11,9 %
D10 (cloud)	100,0 %	0,9 %
D11 (cloud Cellphone)	100,0 %	12,2 %

7.6 Laufzeitanalyse für die Bedeckungsgradberechnung

In diesem Abschnitt wird die benötigte Verarbeitungszeit für die entwickelte Berechnung des Schnee- und Wolkenbedeckungsgrades analysiert. Da diese Berechnung ein semantisch segmentiertes Bild benötigt, wird die entsprechende Verarbeitungszeit für die semantische Segmentierung ebenfalls ausgewertet. In Tabelle 7.12 ist die durchschnittliche Laufzeit der semantischen Segmentierung (Pred.) und der Berechnung des Bedeckungsgrades (Calc.) in Millisekunden dargestellt. Die Laufzeiten werden bei der Verarbeitung von Bildern der verschiedenen Datensätze auf zwei Systemen aufgenommen.

Tabelle 7.12: Auswertung der durchschnittlichen Laufzeit für die Berechnung des Bedeckungsgrades sowie der semantischen Segmentierung in Millisekunden

Datensatz	Bildgröße	System 1[29]		System 2[30]	
		Pred.	Calc.	Pred.	Calc.
D3 (muccam01)	$(600 \times 360 \times 3)$	588,6 ms	4,1 ms	530,6 ms	4,0 ms
D4 (garden)	$(220 \times 240 \times 3)$	243,8 ms	0,5 ms	165,6 ms	0,3 ms
D5 (waterday)	$(640 \times 480 \times 3)$	1144,6 ms	7,6 ms	916,6 ms	6,1 ms
D6 (waternight)	$(640 \times 480 \times 1)$	845,0 ms	6,1 ms	852,3 ms	6,3 ms
D7 (Mobotix)	$(820 \times 820 \times 3)$	1556,4 ms	12,2 ms	1383,4 ms	10, 2ms
D8 (Vivothek)	$(1280 \times 960 \times 3)$	2612,9 ms	24,0 ms	2551,4 ms	25,1 ms
Durchschnitt	-	1165,2 ms	9,1 ms	1066,7 ms	8,7 ms

[29] Intel Core i7-7500U CPU @ 2.70GHz | 16,0 GB RAM | Intel HD Graphics 620.
[30] AMD Ryzen 5 2500U CPU @ 2.00 GHz | 8,0 GB RAM | AMD Radeon Vega 8.

Nach den Ergebnissen aus Tabelle 7.12 ist die Laufzeit abhängig von der Eingangsgröße der zu verarbeitenden Bilder. Dies gilt sowohl für die semantische Segmentierung, als auch für die Berechnung des Bedeckungsgrades. Das beste Ergebnis bietet mit durchschnittlich 165,9 Millisekunden Gesamtverarbeitungszeit für Bilder des Datensatzes *garden* das zweite System. Die beste durchschnittliche Gesamtverarbeitungszeit über alle Datensätze erreicht mit 1075,4 Millisekunden ebenfalls das zweite System.

7.7 Auswertung der Anforderungen

Dieses Kapitel fasst den Status der Erfüllung der Anforderungen an die vorliegende Arbeit zusammen. Die Hauptanforderungen des Projektes aus Tabelle 7.13 sind vollständig erfüllt. Es wurden lauffähige Softwarelösungen für die Netzimplementierung und Datenaufbereitung sowie eine Anwendung für Endnutzer*innen entwickelt und realisiert. Diese ermöglichen die Implementierung neuronaler Netze für die Schnee- und Wolkenerkennung sowie die geforderte Einordnung in die Referenzklassen.

Tabelle 7.13: Auswertung der Hauptanforderungen des Projektes

Nr.	Prio.	Anforderung	Status	Referenz
A1	erf.	Schneeerkennung	erfüllt	Kapitel: 7.2
A2	erf.	Wolkenerkennung	erfüllt	Kapitel: 7.3
A3	erf.	Bedeckungsgrad	erfüllt	Kapitel: 7.2.3 Kapitel: 7.3.3
A4	erf.	Klassifizierung (Schnee, 5 Klassen)	erfüllt	Kapitel: 7.2.3
A5	erf.	Klassifizierung (Schnee, 11 Klassen)	erfüllt	Kapitel: 7.2.3
A6	erf.	Klassifizierung (Wolken, 5 Klassen)	erfüllt	Kapitel: 7.3.3
A7	erf.	Netzimplementierung	erfüllt	Kapitel: 6.2
A8	erf.	Datenaufbereitung	erfüllt	Kapitel: 6.1
A9	erf.	Anwendung für Endnutzer*innen	erfüllt	Kapitel: 6.4

Die Qualitätsanforderungen aus Tabelle 7.14 werden teilweise erfüllt. Die Verarbeitung von Bildern variabler Größe zur Minimierung von Informationsverlusten ist sowohl durch die entwickelten Segmentierungsnetzwerke, als auch durch den Domain-Klassifikator möglich. Eine Berücksichtigung perspektivischer Verzerrungen existiert lediglich für die Schneeerkennung. Bei der Wolkenerkennung wird aufgrund der dargelegten Erkenntnisse auf Entzerrungsmethoden verzichtet. Die Anforderungen an die Accuracy und den MAE bei der Einordnung in die Referenzklassen werden bei der Schneeerkennung für einzelne Domains sowohl in fünf, als auch in elf Referenzklassen bezogen auf die Ground Truth erfüllt. Für den Domain Transfer erfüllt die Schneeerkennung diese Anforderungen bei fünf Referenzklassen ebenfalls überwiegend. Lediglich für die unbekannte Domain *garden* wird die vorgegebene Accuracy bei einem Domain Transfer nicht eingehalten. Bei der Wolkenerkennung wird der geforderte MAE bei Nutzung einzelner Domains, sowie durchschnittlich bei den Domain Transfer Methoden erreicht. Die Anforderung an die Accuracy wird unter den gegebenen Umständen bei der Wolkenbedeckungsmessung jedoch nicht erfüllt. Die Laufzeitanforderung wird für die reine Berechnung des Bedeckungsgrades durchweg erfüllt. Wird die semantische Segmentierung in die Laufzeit mit einbezogen, kann die vorgegebene Laufzeit von 200 Millisekunden bezogen auf eine VGA-Auflösung mit den genutzten Systemen nicht eingehalten werden. Nachtaufnahmen in Form von IR-Bildern werden hier aufgrund der Verfügbarkeit ausschließlich bei der Domain *waternight* ausgewertet. Bei dieser Domain wird die Anforderung erfüllt.

Tabelle 7.14: Auswertung der Qualitätsanforderungen

Nr.	**Prio.**	**Anforderung**	**Status**	**Referenz**
A10	erf.	variable Bildgröße	erfüllt	Kaptiel: 6.2
A11	opt.	Bildverzerrung	teilweise erfüllt	Kaptiel: 7.2.3
A12	opt.	Genauigkeit	teilweise erfüllt	Kapitel: 7.2.3 Kapitel: 7.3.3
A13	erf.	Geschwindigkeit	nicht erfüllt	Kaptiel: 7.6
A14	erf.	Tag/Nacht	erfüllt	Kaptiel: 7.2.1

Die in Tabelle 7.15 aufgelisteten Anforderungen an die Software-Implementierungen werden ebenfalls vollständig erfüllt. Die implementierte Software ist vollständig in der Programmiersprache Python verfasst und kompatibel zum Betriebssystem Linux. Darüber

hinaus werden ausschließlich frei verfügbare und kostenlos nutzbare Bibliotheken[31] verwendet. Die Parametrierungen der Programme erfolgen ausschließlich über json-Dateien außerhalb des Programmcodes.

Tabelle 7.15: Auswertung der Anforderungen an die Software-Implementierungen

Nr.	Prio.	Anforderung	Status	Referenz
A15	erf.	Kompatibilität	erfüllt	Kapitel: 6
A16	erf.	Bibliotheken	erfüllt	Kapitel: E
A17	opt.	Parametrierung	erfüllt	Kapitel: 6

Die Anwendung für Endnutzer*innen enthält ebenfalls alle geforderten Funktionen an das Ausgabeformat, die Datenspeicherung und -verarbeitung, einfache Portierbarkeit und die Nutzung sowie Erstellung verschiedener Bildmaskierungen (siehe Tabelle 7.16).

Tabelle 7.16: Auswertung der Anforderungen an die Anwendung für Endnutzer*innen aus A9

Nr.	Prio.	Anforderung	Status	Referenz
A18	erf.	Ausgabeformat	erfüllt	Kapitel: 6.4
A19	erf.	Portierung	erfüllt	Kapitel: 6.4
A20	opt.	Datenspeicherung	erfüllt	Kapitel: 6.4
A21	opt.	Datenverarbeitung	erfüllt	Kapitel: 6.4
A22	opt.	Maskierung	erfüllt	Kapitel: 6.4

[31] Die genutzten Bibliotheken sind in Tabelle E.1 im Anhang aufgelistet.

8 Fazit und Ausblick

Ein Ziel dieser Arbeit war die semantische Segmentierung von Schnee- und Wolkenaufnahmen verschiedener Bodenkameras und Standorte. Mit Hilfe der semantisch segmentierten Aufnahmen galt es, die Schnee- beziehungsweise Wolkenbedeckungsgrade möglichst genau zu ermitteln. Unter der Voraussetzung, dass ein Deep Learning-Verfahren für die Lösung dieser Aufgabe zu verwenden war, waren zudem geeignete Trainingsdaten für die Realisierung der dafür nötigen künstlichen neuronalen Netze zu erstellen.
Für die Umsetzung dieser Ziele wurden im Rahmen der vorliegenden Arbeit als Infrastruktur drei Programme entwickelt und implementiert. Dabei entstand ein Programm für die effiziente Aufbereitung von Trainingsdaten, eines zum Trainieren und Testen der künstlichen neuronalen Netze, sowie eine Anwendung, die es den Endnutzer*innen ermöglicht, die trainierten neuronalen Netze für die Ermittlung der Bedeckungsgrade anzuwenden. Alle Programme wurden ausschließlich unter Verwendung von Bibliotheken ohne kostenpflichtige Lizenzen geschaffen.
Unter Verwendung des Programms zur Datenaufbereitung wurden insgesamt mehr als 2500 Bilder auf Pixelebene für das Training der künstlichen neuronalen Netze annotiert. Da die Qualität der semantischen Segmentierung durch Deep Leaning-Verfahren eng mit der Qualität der Trainingsdaten zusammenhängt (vgl. [Mullen u. a., 2019]), trägt das im Rahmen dieser Arbeit für die Datenaufbereitung entwickelte Programm maßgeblich zum Erfolg des Projektes bei.
Durch die Implementierung verschiedener Netz-Architekturen sowie die Entwicklung einer eigenen Architektur konnten mehrere mögliche Ansätze für die Lösung der Problemstellung gegenübergestellt werden. Mit der im Rahmen dieser Arbeit entwickelten KMTX-Architektur wurde der durchschnittliche F1-Score bei Tests verschiedener Domains im Vergleich zum zweitplatzierten UX-Net um $2,3\,\%$ verbessert. Dabei konnten bei einzelnen Domains F1-Scores von mehr als $92\,\%$ erreicht werden. Auch bei der Wolkenerkennung erzielt die semantische Segmentierung mit dem KMTX-Net für einzelne Domains F1-Scores von über $89\,\%$ bei deutlich geringerer Anzahl aufbereiteter Trainingsbilder.
Für die Klassifizierung der Bedeckungsgrade erreicht die Schneebedeckungsmessung bei

fünf Referenzklassen durchweg Accuracies von über 90 % sowie teilweise von 100 %. Bei der Nutzung von elf Referenzklassen liegen die Ergebnisse der Accuracies im Durchschnitt ebenfalls bei über 90 %. Die Accuracy der Klassifizierung des Wolkenbedeckungsgrades erreicht Werte bis 86 %. Darüber hinaus werden sowohl bei der Schnee-, als auch bei der Wolkenerkennung durchschnittliche MAE von unter 10 % erreicht.
Für den Domain Transfer wurden die Netzarchitekturen so implementiert, dass sie unabhängig von der Größe der Eingangsbilder sind. Darüber hinaus wurde das merkmalsbasierte Domain Adversarial Transfer Learning (DATL) implementiert, getestet und dessen Einfluss auf die Ergebnisse beim Domain Transfer analysiert. So konnten merkliche Verbesserungen bei der semantischen Segmentierung von Aufnahmen unbekannter Domains durch das DATL nachgewiesen werden. Die geschaffene Infrastruktur sowie die entwickelte Netz-Architektur und die Methoden für die semantische Segmentierung lassen sich nach den Erkenntnissen der vorliegenden Arbeit sowohl für die Schnee- als auch für die Wolkenerkennung nutzen.
Zusätzlich zu den Errungenschaften bei der semantischen Segmentierung wurde ein Klassifikator realisiert, der die Identifizierung und Aussortierung von Aufnahmen variabler Größe ermöglicht, welche für die semantische Segmentierung nicht nutzbar sind.
Alles in allem wurde mit dieser Arbeit ein weitestgehend domainunabhängiges Verfahren für die Ermittlung des Schnee- und Wolkenbedeckungsgrades über semantische Segmentierung erarbeitet. Im Hinblick auf die Ergebnisse der Validierung und der empirischen Analyse stellt das hier entwickelte Verfahren eine gute Alternative zu der manuellen Bedeckungsmessung dar.
Während dieser Ausarbeitung wurden zudem potentielle Verbesserungsmöglichkeiten für die hier umgesetzten Methoden und die erzielten Ergebnisse identifiziert. Die Auswertung des Einflusses der Trainingsdatenanzahl bei der Wolkenerkennung hat ergeben, dass in diesem Bereich Verbesserungen durch weitere aufbereitete Trainingsbilder möglich sind. Weiterhin kann die verwendete KMTX-Architektur, wie von [Tong u. a., 2021] beschrieben, zu einem Evidential Fully Convolutional Network (EFCN) erweitert werden. Dadurch wird das Training mit Softlabel ermöglicht. Diese erlauben die Verwendung mehrerer Klassen pro Pixel in Bildbereichen, in denen die Pixel nicht mit absoluter Sicherheit einer Klasse zugeordnet werden können. Dieses Verfahren bietet besonders bei der Wolkenerkennung für die Klassifizierung von Wolkenausläufern Vorteile. In diesen Bereichen treten die größten Abweichungen zwischen Ground Truth und Prediction auf, da eine exakte Klassenzuordnung für diese Bereiche sehr schwierig ist. Auch bei der Schneeerkennung finden sich mögliche Anwendungsfälle für diese Verbesserung, beispielsweise bei sehr dünner Schneedecke oder Raureif.

Eine weitere potentielle Verbesserung bietet die Nutzung von Generative Adversarial Networks für den Domain Transfer und insbesondere für das DATL. Damit können neben dem in dieser Arbeit umgesetzten merkmalsbasierten DATL die von [Brion u. a., 2021] beschriebenen Transfer-Verfahren auf Bild- und Label-Ebene umgesetzt werden. Nach [Khan u. a., 2020] kann durch die Augmentations der GAN-Architektur die Accuracy der neuronalen Netze weiter verbessert werden: „The accuracy can be further improved with more augmented data by injecting a Generative Adversarial Network (GAN) to the deep models“.
Ein weiteres Gebiet, welches ebenfalls eine Art von Domain Transfer darstellt, ist die Umwandlung von IR-Aufnahmen zu Farbaufnahmen. Durch diese Vorverarbeitung von Nachtaufnahmen von IR-Kameras können diese als Tagaufnahmen verarbeitet werden, wodurch aufbereitete Trainingsdaten für die Nachtaufnahmen eingespart werden. Darüber hinaus bieten Farbaufnahmen mehr Features für die Unterscheidung der Klassen bei der semantischen Segmentierung und es müssen keine zusätzlichen Features für die Verarbeitung von einkanaligen IR-Aufnahmen extrahiert werden. Verfahren für eine solche Bildverarbeitung werden von [Limmer u. Lensch, 2016] und [Browne u. a., 2022] eingeführt. Wie von [Browne u. a., 2022] beschrieben, lassen sich die Ergebnisse der Umwandlung von IR- zu Farbaufnahmen ebenfalls durch die Verwendung von GAN optimieren.
Auf der anderen Seite können IR-Aufnahmen auch für die Verbesserung der Segmentierungsergebnisse von Farbaufnahmen verwendet werden. [Hongcai u. a., 2019] erläutern dafür ein Verfahren zur Feature-Fusion, welches sowohl Farbaufnahmen, als auch IR-Aufnahmen der selben Szenerie für das Training der neuronalen Netze verwendet, um zusätzliche Features für die semantische Segmentierung zu generieren.

Literaturverzeichnis

[Aggarwal 2018] Aggarwal, Charu C.: *Neural Networks and Deep Learning*. Springer International Publishing, 2018 https://doi.org/10.1007/978-3-319-94463-0

[Arel u. a. 2010] Arel, Itamar ; Rose, Derek C. ; Karnowski, Thomas P.: Deep Machine Learning - A New Frontier in Artificial Intelligence Research [Research Frontier]. In: *IEEE Computational Intelligence Magazine* 5 (2010), Nr. 4, 13-18. http://dx.doi.org/10.1109/MCI.2010.938364

[Baierl 2008] Baierl, Leif: *Aufbau und Test katadioptrischer Systeme zur Rekonstruktion von 3D*, Universität Koblenz Landau, Diss., September 2008

[Brandenbusch 2018] Brandenbusch, Kai: *Semantische Segmentierung mit DeepConvolutional Neural Networks*, Technische Universität Dortmund, Masterthesis, November 2018

[Brion u. a. 2021] Brion, Eliott ; Léger, Jean ; Barragán-Montero, A.M. ; Meert, Nicolas ; Lee, John A. ; Macq, Benoit: Domain adversarial networks and intensity-based data augmentation for male pelvic organ segmentation in cone beam CT. In: *Computers in Biology and Medicine* 131 (2021). https://doi.org/10.1016/j.compbiomed.2021.104269. – ISSN 0010–4825

[Browne u. a. 2022] Browne, Andrew W. ; Deyneka, Ekaterina ; Ceccarelli, Francesco ; To, Josiah K. ; Chen, Siwei ; Tang, Jianing ; Vu, Anderson N. ; Baldi, Pierre F.: Deep learning to enable color vision in the dark. In: *PLOS ONE* 17 (2022), April, Nr. 4, 1-15. https://doi.org/10.1371/journal.pone.0265185

[Chen u. a. 2017] Chen, Liang-Chieh ; Papandreou, George ; Schroff, Florian ; Adam, Hartwig: Rethinking Atrous Convolution for Semantic Image Segmentation. In: *CoRR* abs/1706.05587 (2017). http://arxiv.org/abs/1706.05587

[Chen u. a. 2018] Chen, Liang-Chieh ; Zhu, Yukun ; Papandreou, George ; Schroff, Florian ; Adam, Hartwig: Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation. In: *CoRR* abs/1802.02611 (2018). http://arxiv.org/abs/1802.02611

[Cho u. a. 2015] Cho, Junghwan ; Lee, Kyewook ; Shin, Ellie ; Choy, Garry ; Do, Synho: Medical Image Deep Learning with Hospital PACS Dataset. In: *CoRR* abs/1511.06348 (2015). http://arxiv.org/abs/1511.06348

[Chollet 2017] Chollet, Francois: Xception: Deep Learning With Depthwise Separable Convolutions. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017

[Cover u. Hart 1967] Cover, T. ; Hart, P.: Nearest neighbor pattern classification. In: *IEEE Transactions on Information Theory* 13 (1967), Nr. 1, 21-27. http://dx.doi.org/10.1109/TIT.1967.1053964

[Dozat 2016] Dozat, Timothy: Incorporating Nesterov Momentum into Adam. In: *International Conference on Learning Representations* (2016), Februar

[Dumoulin u. Visin 2018] Dumoulin, Vincent ; Visin, Francesco: *A guide to convolution arithmetic for deep learning*. https://arxiv.org/abs/1603.07285. Version: 2018

[DWD 1967] DWD, (Deutscher W. ; Schneider, Kurt (Hrsg.) ; Schnell, Arthur (Hrsg.): *Leitfäden für die Ausbildung im Deutschen Wetterdienst Nr. 4 Wetterbeobachtung*. Selbstverlag des Deutschen Wetterdienstes, 1967

[DWD 2021] DWD, (Deutscher W.: *Einheitliche Beobachteranleitung für nebenamtliche Stationen*. Selbstverlag des Deutschen Wetterdienstes, 2021

[Farabet u. a. 2013] Farabet, Clement ; Couprie, Camille ; Najman, Laurent ; LeCun, Yann: Learning Hierarchical Features for Scene Labeling. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2013

[Fawaz u. a. 2018] Fawaz, Hassan I. ; Forestier, Germain ; Weber, Jonathan ; Idoumghar, Lhassane ; Muller, Pierre-Alain: Deep learning for time series classification: a review. In: *CoRR* abs/1809.04356 (2018). http://arxiv.org/abs/1809.04356

[Goodfellow u. a. 2016] Goodfellow, Ian ; Bengio, Yoshua ; Courville, Aaron: *Deep Learning*. MIT Press, 2016. – ISBN 978–0262035613. – http://www.deeplearningbook.org (zul. aufgerufen am 02.04.2022)

[Grandini u. a. 2020] Grandini, Margherita ; Bagli, Enrico ; Visani, Giorgio: *Metrics for Multi-Class Classification: an Overview*. https://arxiv.org/abs/2008.05756. Version: 2020

[Géron 2020] Géron, Aurélien: *Praxiseinstieg Machine Learning mit Scikit-Learn, Keras und TensorFlow*. Bd. 2. O'REILLY, 2020

[He u. a. 2014] He, Kaiming ; Zhang, Xiangyu ; Ren, Shaoqing ; Sun, Jian: Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. In: *CoRR* abs/1406.4729 (2014). http://arxiv.org/abs/1406.4729

[He u. a. 2016] He, Kaiming ; Zhang, Xiangyu ; Ren, Shaoqing ; Sun, Jian: Deep Residual Learning for Image Recognition. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016

[Hongcai u. a. 2019] Hongcai, Du ; Li, Kun ; Guo, Jianhua ; Zhang, Jinsong ; Yang, Jingyu: Cloud and snow detection from remote sensing imagery based on convolutional neural network, 2019, S. 41

[Hu u. a. 2016] Hu, Li-Yu ; Huang, Min-Wei ; Ke, Shih-Wen ; Tsai, Chih-Fong: The distance function effect on k-nearest neighbor classification for medical datasets. In: *SpringerPlus* 5 (2016), August, Nr. 1. http://dx.doi.org/10.1186/s40064-016-2941-7

[von der Hude 2020] In: Hude, Marlis von d.: *k-nächste Nachbarn (k nearest neighbours)*. Wiesbaden : Springer Fachmedien Wiesbaden, 2020. – ISBN 978–3–658–30153–8, 99–106

[Jaccard 1912] Jaccard, Paul: THE DISTRIBUTION OF THE FLORA IN THE ALPINE ZONE.1. In: *New Phytologist* 11 (1912), Nr. 2, S. 37–50

[Jacob u. a. 2021] Jacob, Joseph ; Ciccarelli, Olga ; Barkhof, Frederik ; Alexander, Daniel C.: Disentangling Human Error from the Ground Truth in Segmentation of Medical Images ACL, 2021

[Jadon 2020] Jadon, Shruti: A survey of loss functions for semantic segmentation. (2020), Oktober, 1-7. http://dx.doi.org/10.1109/CIBCB48159.2020.9277638

[Kandel u. Castelli 2020] Kandel, Ibrahem ; Castelli, Mauro: The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset. In: *ICT Express* 6 (2020), Nr. 4, 312-315. https://doi.org/10.1016/j.icte.2020.04.010. – ISSN 2405–9595

[Khan u. a. 2020] Khan, Zia ; Yahya, Norashikin ; Alsaih, Khaled ; Ali, Syed S. ; Meriaudeau, Fabrice: Evaluation of Deep Neural Networks for Semantic Segmentation of Prostate in T2W MRI. In: *Sensors* 20 (2020), Juni. http://dx.doi.org/10.3390/s20113183

[Kim u. a. 2017] Kim, Heehoon ; Nam, Hyoungwook ; Jung, Wookeun ; Lee, Jaejin: Performance analysis of CNN frameworks for GPUs. In: *2017 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2017, S. 55–64

[Kingma u. Ba 2014] Kingma, Diederik P. ; Ba, Jimmy: *Adam: A Method for Stochastic Optimization.* 2014

[Koeppe u. a. 2019] Koeppe, Arnd ; Hesser, Daniel F. ; Mundt, Marion ; Bamer, Franz ; Markert, Bernd: Mechanik 4.0. Künstliche Intelligenz zur Analyse mechanischer Systeme. (2019), Dezember, S. 553–567

[Komiyama u. a. 2018] Komiyama, Tomoya ; Hotta, Kazuhiro ; Oda, Kazuo ; Kakuta, Satomi ; Sano, Mikako: Semantic Segmentation in Red Relief Image Map by UX-Net, 2018, S. 597–602

[Kraus 2004] Kraus, Karl: *Photogrammetrie.* 7. Berlin, Germany : De Gruyter, 2004 (De Gruyter Lehrbuch)

[Kuner 2015] Kuner, R.: *Deutscher Wetterdienst: Vorschriften und Betriebsunterlagen Nr. 3 Beobachterhandbuch für Wettermeldestellen des synoptisch-klimatologischen Mess- und Beobachtungsnetzes.* 2015

[Li u. a. 2018] Li, Zhiwei ; Shen, Huanfeng ; Cheng, Qing ; Liu, Yuhao ; You, Shucheng ; He, Zongyi: Deep learning based cloud detection for remote sensing images by the fusion of multi-scale convolutional features. In: *CoRR* abs/1810.05801 (2018). http://arxiv.org/abs/1810.05801

[Limmer u. Lensch 2016] Limmer, Matthias ; Lensch, Hendrik P. A.: Infrared Colorization Using Deep Convolutional Neural Networks. In: *CoRR* abs/1604.02245 (2016). http://arxiv.org/abs/1604.02245

[Long u. a. 2015] Long, Jonathan ; Shelhamer, Evan ; Darrell, Trevor: Fully convolutional networks for semantic segmentation. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, S. 3431–3440

[Lozán u. a. 2015] Lozán, José L. ; Escher-Vetter, Heidi ; Grassl, Hartmut ; Kasang, Dieter ; Notz, Dirk: Der Klimawandel und das Eis der Erde: Ein Überblick. In: *Warnsignal Klima: Das Eis der Erde* (2015). http://dx.doi.org/10.2312/WARNSIGNAL.KLIMA.EIS-DER-ERDE.02

[Marquardt u. Reigber 2002] Marquardt, Dr. rer. nat. C. ; Reigber, Prof. Dr.-Ing. Dr. Ing. E.h. C.: Messung an verborgenen Orten. In: *Helmholtz-Jahresheft* (2002), S. 18–20

[Müller u. Behnke 2014] Müller, Andreas C. ; Behnke, Sven: Learning depth-sensitive conditional random fields for semantic segmentation of RGB-D images. In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2014, S. 6232–6237

[Mossig 2012] Mossig, Ivo: Stichproben, Stichprobenauswahlverfahren und Berechnung des minimal erforderlichen Stichprobenumfangs. Bremen : Universität Bremen, Institut für Geographie, 2012 (1-2012). – Beiträge zur Wirtschaftsgeographie und Regionalentwicklung. – http://hdl.handle.net/10419/90425 (zul. aufgerufen am 02.04.2022)

[Mullen u. a. 2019] Mullen, James F. J. ; Tanner, Franklin R. ; Sallee, Phil A.: Comparing the Effects of Annotation Type on Machine Learning Detection Performance. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, 2019

[Opitz u. Burst 2021] Opitz, Juri ; Burst, Sebastian: *Macro F1 and Macro F1*. https://arxiv.org/abs/1911.03347. Version: 2021

[Paaß u. Hecker 2020] Paass, Gerhard ; Hecker, Dirk: *Künstliche Intelligenz*. Springer Fachmedien Wiesbaden, 2020 https://doi.org/10.1007/978-3-658-30211-5. – ISBN 978–3–658–30211–5

[Padberg 2021] Padberg, Kirstin: *Bildverarbeitungsgestutzte Ermittlung des Schneebedeckungsgrades*, Hamburg University of Applied Sciences, Masterthesis, November 2021

[Perkovic 2022] Perkovic, Mike: *Methodenvergleich zum Einsatz von Deep Learning zur kamerabasierten Ermittlung des Schneebedeckungsgrades*, Hamburg University of Applied Sciences, Masterthesis, Januar 2022

[Ronneberger u. a. 2015] Ronneberger, Olaf ; Fischer, Philipp ; Brox, Thomas: U-Net: Convolutional Networks for Biomedical Image Segmentation. (2015), Mai

[Rosenblatt 1958] Rosenblatt, F.: The perceptron: A probabilistic model for information storage and organization in the brain. In: *Psychological Review* 65 (1958)

[Scannell u. a. 2020] Scannell, Cian M. ; Chiribiri, Amedeo ; Veta, Mitko: *Domain-Adversarial Learning for Multi-Centre, Multi-Vendor, and Multi-Disease Cardiac MR Image Segmentation.* 2020

[Scaramuzza u. a. 2006] Scaramuzza, Davide ; Martinelli, Agostino ; Siegwart, Roland: A Toolbox for Easily Calibrating Omnidirectional Cameras. In: *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2006, S. 5695–5701

[Schreer 2005] Schreer, Oliver: *Stereoanalyse und Bildsynthese.* Springer-Verlag, 2005 http://dx.doi.org/10.1007/3-540-27473-1

[Shi u. a. 2016] Shi, Wenzhe ; Caballero, Jose ; Theis, Lucas ; Huszar, Ferenc ; Aitken, Andrew P. ; Ledig, Christian ; Wang, Zehan: Is the deconvolution layer the same as a convolutional layer? In: *CoRR* abs/1609.07009 (2016). http://arxiv.org/abs/1609.07009

[Srivastava u. a. 2014] Srivastava, Nitish ; Hinton, Geoffrey ; Krizhevsky, Alex ; Sutskever, Ilya ; Salakhutdinov, Ruslan: Dropout: a simple way to prevent neural networks from overfitting. In: *The journal of machine learning research* 15 (2014), Nr. 1, S. 1929–1958

[Strauss 2007] Strauss, Franziska: *Unsicherheiten in der Bewölungsevaluierung mit ISCCP-Daten*, Universität Wien, Diss., Januar 2007

[Tong u. a. 2021] Tong, Zheng ; Xu, Philippe ; Denœux, Thierry: Evidential fully convolutional network for semantic segmentation. In: *Applied Intelligence* 51 (2021), April, Nr. 9, 6376–6399. https://doi.org/10.1007/s10489-021-02327-0

[Traeger u. a. 2003] Traeger, M. ; Eberhart, A. ; Geldner, G. ; Morin, A. M. ; Putzke, C. ; Wulf, H. ; Eberhart, L. H. J.: Künstliche neuronale Netze. In: *Der Anaesthesist* 52 (2003), November, Nr. 11, 1055–1061. https://doi.org/10.1007/s00101-003-0576-x

[Uras 2015] Uras, Faruk: *Automatische Wolkenanalyse und Wolkendetektion unter Verwendung einer Fisheye-Kamera*, Hamburg University of Applied Sciences, Diplomarbeit, Juli 2015

[Verma u. a. 2017] Verma, Amit ; Tripathi, Madanmohan ; Upadhyay, Kaushal: A Review Article on Green Energy Forecasting. In: *Asia-Pacific Journal of Advanced Research in Electrical and Electronics Engineering* 1 (2017), Februar, 1-8. http://dx.doi.org/10.21742/ajaeee.2017.1.1.01

[Winkler u. a. 2019] Winkler, J. K. ; Sies, K. ; Fink, C. ; Toberer, F. ; Enk, A. ; Haenssle, H. A.: Digitalisierte Bildverarbeitung: künstliche Intelligenz im diagnostischen Einsatz. In: *Forum* 35 (2019), Dezember, Nr. 2, 109–116. https://doi.org/10.1007/s12312-019-00729-3

[Xue u. a. 2019] Xue, Zhucun ; Xue, Nan ; Xia, Gui-Song ; Shen, Weiming: Learning to Calibrate Straight Lines for Fisheye Image Rectification. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019

[Yaqub u. a. 2020] Yaqub, Muhammad ; Feng, Jinchao ; Zia, M. S. ; Arshid, Kaleem ; Jia, Kebin ; Rehman, Zaka U. ; Mehmood, Atif: State-of-the-Art CNN Optimizer for Brain Tumor Segmentation in Magnetic Resonance Images. In: *Brain Sciences* 10 (2020), Nr. 7. http://dx.doi.org/10.3390/brainsci10070427. – ISSN 2076–3425

[Yu u. Koltun 2016] Yu, Fisher ; Koltun, Vladlen: *Multi-Scale Context Aggregation by Dilated Convolutions*. 2016

[Zhang 2000] Zhang, Z.: A flexible new technique for camera calibration. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22 (2000), Nr. 11, 1330-1334. http://dx.doi.org/10.1109/34.888718

A Verwendete Datensätze

In diesem Abschnitt sind die verwendeten Trainingsdatensätze dargestellt. Diese setzen sich aus den pixelweise annotierten Bilddaten für die Schnee- und Wolkenerkennung sowie aus Bilddaten ohne Annotierung für die Klassifikatoren zusammen (siehe Tab. A.1).

Tabelle A.1: Zusammensetzung der Datensätze

Referenz	**Datensatz**	**Projekt**	**Bildanzahl**	**Aufteilung**[32]		
D1	waterday	snow	478	70	25	5
D2	garden	snow	796	70	25	5
D3	muccam01	snow	771	75	20	5
D4	garden	snow	846	75	20	5
D5	waterday	snow	517	75	20	5
D6	waternight	snow	563	75	20	5
D7	Mobotix	cloud	60	50	15	35
D8	Vivothek	cloud	60	50	15	35
D9	usability muccam01	snow	91	65	20	15
D10	usability cloud	cloud	879	75	20	5
D11	usability cloud cellphone	cloud	18	0	0	100

[32] Trainingsbilder | Validierungsbilder | Testbilder – Aufteilung in % .

B Codeausschnitte

Dieser Anhang enthält ausgewählte, für die Nachvollziehbarkeit der Arbeit wichtige Codeausschnitte.

Codeausschnitt B.1: Labelmizer: Klasse ColorGrabber

```
# get color channels
(b, g, r) = self.image[y, x, :]
# calculate color ratio
b_g = b / g
b_r = b / r
# processing all pixels
for i in range(self.H):
   for j in range(self.W):
      if (
            (self.image[i, j, 0] / self.image[i, j, 1] <= b_g * self.project[
               ↪ 'distance_factor'] and self.image[i, j, 0] / self.image[i,
               ↪ j, 1] >= b_g / self.project['distance_factor'])
         and (self.image[i, j, 0] / self.image[i, j, 2] <= b_r * self.project
            ↪ ['distance_factor'] and self.image[i, j, 0] / self.image[i, j,
            ↪  2] >= b_r / self.project['distance_factor'])
        ): self.label_map[i, j] = self.class_to_label * self.project['
           ↪ visualization_factor']
```

Codeausschnitt B.2: Netzimplementierung: Beispielhafter Netzaufbau

```
import tensorflow as tf
import numpy as np
import keras.backend as K
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, GlobalAveragePooling2D,
    ↪ GlobalMaxPooling2D, Reshape, Dense, Conv2D, BatchNormalization,
    ↪ Activation, Concatenate
# define input layer
model_input = Input(defined_model_input_shape)
# define all layers between input and output
```

```
model_output = define_all_layers(inp, architecture)
#generate model
model = Model(model_input, model_output)
```

Codeausschnitt B.3: Netzimplementierung: Channel Attention Module

```
def CAM(cam_input, c_unit_factor):
    """
  Function to build the channel attention module (CAM)
  Args:
    cam_input: tf.keras.layers.Layer, input layer
    c_unit_factor: float, ratio factor between neurons in hidden layer and
        ↪ input layer channels
  Return:
    channel_attention: tf.keras.layers.Layer, with channel weighted input
    """
    shape = cam_input.shape
    hidden_size = int(shape[-1] * c_unit_factor)
    # global pooling
    c_shape = (1, 1, shape[-1])
    cam_aver = GlobalAveragePooling2D()(cam_input)
    cam_aver = Reshape(c_shape)(cam_aver)
    cam_max = GlobalMaxPooling2D()(cam_input)
    cam_max = Reshape(c_shape)(cam_max)
    # init shared weights
    Dense_hidden = Dense(hidden_size)
    Dense_out = Dense(shape[-1])
    # build mlp
    c_aver = Dense_hidden(cam_aver)
    c_aver = BatchNormalization()(c_aver)
    c_aver = Dense_out(c_aver)
    c_aver = BatchNormalization()(c_aver)
    c_max = Dense_hidden(cam_max)
    c_max = BatchNormalization()(c_max)
    c_max = Dense_out(c_max)
    c_max = BatchNormalization()(c_max)
    # generate output
    c_comb = c_aver + c_max
    c_comb = Activation('sigmoid')(c_comb)
    channel_attention = cam_input * c_comb
    return channel_attention
```

Codeausschnitt B.4: Netzimplementierung: Spatial Attention Module

```
def SAM(sam_input, s_kernel):
    """
  Function to build the spatial attention module (SAM)
  Args:
    sam_input: tf.keras.layers.Layer, input layer
    s_kernel: int, size of filterkernel
  Return:
    spatial_attention: tf.keras.layers.Layer, with spatial weighted input
    """
    # channel pooling
    sam_aver = tf.expand_dims(K.mean(sam_input, axis=-1), axis=-1)
    sam_max = tf.expand_dims(K.max(sam_input, axis=-1), axis=-1)
    # convolution
    s_comb = Concatenate()([sam_aver, sam_max])
    s_comb = Conv2D(1, s_kernel, padding='same')(s_comb)
    s_comb = BatchNormalization()(s_comb)
    s_comb = Activation('sigmoid')(s_comb)
    # generate output
    spatial_attention = sam_input * s_comb
    return spatial_attention
```

Codeausschnitt B.5: Netzimplementierung: Handhabung der Bildgröße

```
# get shape-factor
if model_architecture['shape_factor'] is not None:
    sf = model_architecture['shape_factor']
else:
    sf = (1, 1)
    for layer in model_architecture['pooling_layer']:
        sf = np.multiply(sf, tuple(layer['pooling_kernel']))

""" train phase """
# save shape-factor (sf) in model name
model._name = log_name + f"_Shapefactor_x_{sf[0]}x{sf[1]}_x_"

""" prediction phase """
# extract shape-factor (sf)
sf = str(model.name).split('_x_')[1].split('x')
sf = tuple((int(sf[0]), int(sf[1])))
# reshape image to max integer multiple
(H, W) = tuple(np.multiply(sf, tuple(map(int, np.divide((H, W), sf)))))
```

Codeausschnitt B.6: Netzimplementierung: Klassifizierungsteil des Domain-Klassifikators

```
# init list with bridge
pool_list = [GlobalAveragePooling2D()(bridge_out), GlobalMaxPooling2D()(
    ↪ bridge_out)]
# add all layer
for list_entry in pool_list:
   pool_list.append(GlobalAveragePooling2D()(list_entry))
   pool_list.append(GlobalMaxPooling2D()(list_entry))
# create flat layer
flat_layer = Concatenate()([list_entry for list_entry in pool_list])
# generate hidden layer
dense = Dense(architecture['bridge_parameter']['bridge_depth'], activation
    ↪ ='relu')(flat_layer)
# generate domain classifyer output layer
domain_pred = Dense(qty_classes, activation='sigmoid', name='domain_output
    ↪ ')(dense)
```

Codeausschnitt B.7: Anwendung für Endnutzer*innen: Zuordnung in Referenzklassen

```
def class_mapping(percent, class_mapping_list):
   """
  Function to map the percent value to its reference class

  Args:
    percent: int, percent value of snow/cloud cover
    class_mapping_list: list, list of reference classes

  Return:
    id: int, calcualted calss id
    name: str, calcualted calss name
    """

    # for not usable pictures
    if percent is None:
       return class_mapping_list[0]['id'], class_mapping_list[0]['name']

    # for usable pictures
    for c in class_mapping_list:
       if c['upper_value'] is not None and c['lower_value'] is not None:
          if percent < c['upper_value'] and percent >= c['lower_value']:
             return c['id'], c['name']
    return class_mapping_list[-1]['id'], class_mapping_list[-1]['name']
```

C Erweiterte Validierungstabellen

Im Folgenden sind erweiterte Validierungstabellen des Kapitels 7 dargestellt.

Tabelle C.1: Pixelwise Accuracy der standortunabhängigen Schneeerkennung

Trainingsmethode / Trainingsdaten	**D3 (muccam01)**	**D5 (waterday)**	**D4 (garden)**	**Durchschnitt**
D3 (muccam01)	97,8 %	80,6 %	82,5 %	87,0 %
D3 (muccam01), Trainingsfortsetzung mit D5 (waterday)	83,7 %	98,7 %	88,6 %	90,3 %
D3 (muccam01) und D5 (waterday)	98,0 %	98,4 %	88,4 %	94,9 %
DATL D3 (muccam01) und D5 (waterday)	97,9 %	98,3 %	86,8 %	94,3 %

Tabelle C.2: Vergleich von Pixelwise Accuracies und F1-Scores für den Datensatz D5 (*waterday*) bei direktem Training und der Trainingsfortsetzung

Trainingsmethode / Trainingsdaten	**Acc**	**F1-Score**
D5 (waterday)	98,6 %	92,8 %
D5 (waterday, halbe Trainingsdatenanzahl)	97,0 %	87,9 %
D3 (muccam01), Trainingsfortsetzung mit D5 (waterday)	98,7 %	91,7 %
D3 (muccam01), Trainingsfortsetzung mit D5 (waterday, halbe Trainingsdatenanzahl)	98,1 %	90,0 %

Tabelle C.3: Standortunabhängige Schneeerkennung bei Reduzierung auf das 2- Klassensystem für das DATL

(a) Vergleich der Pixelwise Accuracy

Trainingsmethode	**D3 (muccam01)**	**D5 (waterday)**	**D4 (garden)**	**Durchschnitt**
DATL	97,9 %	98,3 %	98,3 %	98,2 %
DATL mit Augmentations	97,9 %	98,3 %	94,5 %	96,9 %
DATL mit Augmentations und Bildzerstückelung	97,0 %	97,7 %	96,6 %	97,1 %

(b) Vergleich des F1-Scores

Trainingsmethode	**D3 (muccam01)**	**D5 (waterday)**	**D4 (garden)**	**Durchschnitt**
DATL	89,5 %	88,5 %	79,2 %	85,8 %
DATL mit Augmentations	85,6 %	92,3 %	69,5 %	82,5 %
DATL mit Augmentations und Bildzerstückelung	78,2 %	94,2 %	59,1 %	77,2 %

Tabelle C.4: MAE zwischen den Bedeckungsgraden der Ground Truth sowie der Predictions mit und ohne Pixelgewichtung

Datensatz	**Ground Truth**	**Prediction**
D3 (muccam01)	1,1 %	1,2 %
D4 (garden)	1,9 %	1,8 %
D5 (waterday)	1,5 %	1,4 %
D6 (waternight)	0,3 %	0,3 %

Tabelle C.5: Accuracy (G.T. und Ref.) sowie MAE des Klassifizierungsergebnisses bei fünf Referenzklassen unter Anwendung des Domain Adversarial Transfer Learnings für das 2- und 3-Klassensystem ohne Pixelgewichtung

Datensatz	**3-Klassensystem**			**2-Klassensystem**		
	G.T.	**Ref.**	**MAE**	**G.T.**	**Ref.**	**MAE**
D3 (muccam01)	94,9 %	–	1,6 %	94,9 %	–	1,5 %
D4 (garden)	74,4 %	–	7,7 %	74,4 %	–	5,1 %
D5 (waterday)	100,0 %	100,0 %	1,1 %	100,0 %	100,0 %	1,1 %

Tabelle C.6: Accuracy (G.T. und Ref.) sowie MAE des Klassifizierungsergebnisses bei fünf Referenzklassen unter Anwendung des Domain Adversarial Transfer Learnings mit und ohne Pixelgewichtung

Datensatz	**ohne Pixelgewichtung**			**mit Pixelgewichtung**		
	G.T.	**Ref.**	**MAE**	**G.T.**	**Ref.**	**MAE**
D3 (muccam01)	89,7 %	–	2,6 %	89,7 %	–	2,6 %
D4 (garden)	93,0 %	–	0,9 %	93,0 %	–	0,9 %
D5 (waterday)	96,2 %	96,2 %	1,3 %	96,2 %	96,2 %	1,4 %
D6 (waternight)	93,1 %	90,0 %	0,6 %	93,1 %	93,3 %	0,7 %

Tabelle C.7: Accuracy (G.T. und Ref.) des Klassifizierungsergebnisses bei elf Referenzklassen unter Anwendung des Domain Adversarial Transfer Learnings mit und ohne Pixelgewichtung

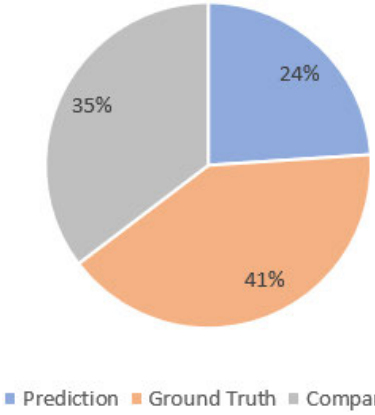
Datensatz	**ohne Pixelgewichtung**		**mit Pixelgewichtung**	
	G.T.	**Ref.**	**G.T.**	**Ref.**
D3 (muccam01)	71,8 %	53,8 %	71,8 %	46,2 %
D4 (garden)	86,0 %	58,1 %	86,0 %	48,8 %
D5 (waterday)	96,2 %	–	96,2 %	–
D6 (waternight)	93,1 %	–	93,1 %	–

Tabelle C.8: Einfluss der Trainingsdatenanzahl bei der Wolkenerkennung (Datensätze: D7 (*Mobotix*) und D8 (*Vivothek*), Augmentations, Bildzerstückelung)

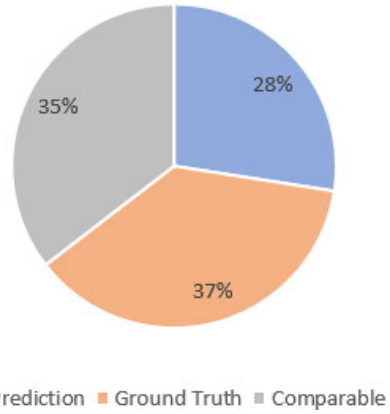
Anzahl Trainingsbilder	**Anzahl Trainingsdaten**	**Acc**	**F1-Score**
10	450	90,6 %	81,5 %
20	900	90,4 %	83,7 %
30	1350	91,4 %	83,6 %
40	1800	91,4 %	84,8 %
50	2250	92,1 %	85,1 %
60	2700	92,6 %	87,9 %

D Schneeerkennung – Empirische Analyse

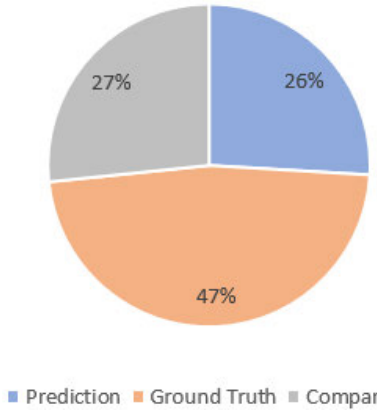
Im Folgenden werden die Ergebnisse der empirischen Analyse der Schneeerkennung aus Kapitel 7.4 aufgeteilt in die Szenerien *waterday* und *waternight* dargestellt.



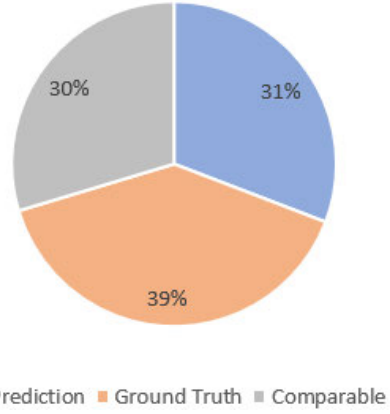
(a) Ergebnis aller Teilnehmer*innen für *waterday*



(b) Ergebnis der Teilnehmer*innen mit relevanter Vorbildung für *waterday*



(c) Ergebnis aller Teilnehmer*innen für *waternight*



(d) Ergebnis der Teilnehmer*innen mit relevanter Vorbildung für *waternight*

Abbildung D.1: Empirische Ergebnisse der Szenerien *waterday* und *waternight*

E Verwendete Software-Bibliotheken

In Tabelle E.1 sind die verwendeten Bibliotheken inklusive der entsprechenden Lizenzen aufgelistet.

Tabelle E.1: Verwendete Bibliotheken

Bibliothek	**Version**	**Lizenz**
albumentations	1.1.0	MIT License
OpenCV	4.5.4	MIT License
tqdm	4.31.1	MIT License
keras	1.0.8	MIT License
mdutils	1.3.1	MIT License
shutil	1.0.0	MIT License
pathlib	2.3.5	MIT License
pandas	1.3.5	BSD License
random	1.1.0	BSD License
scikit-learn	0.22.1	BSD License
time	3.10.4	BSD License
numpy	1.21.5	BSD License
warnings	3.10.4	BSD License
csv	1.0	GNU General Public License
json	0.9.1	Apache Software License
os	2.1.4	Apache Software License
argparse	1.4.0	Apache Software License
tensorflow	2.8.0	Apache Software License
watchdog	0.10.2	Apache Software License

F Digitaler Anhang

- CD Masterthesis
  - Dokumentation
    - MasterThesis_Koenig_Massmann.pdf
    - Kurzanleitung_Datenaufbereitung.pdf
    - Kurzanleitung_Endnutzeranwendung.pdf
  - Implementierungen
    - Datenaufbereitung
      - labelmizer.py
      - labelmizer_config.json
    - Netzimplemetierung
      - build_models.py
      - custom_functions.py
      - data_management.py
      - test.py
      - train.py
      - model_parameter.json
      - user_project_management.json
    - Anwendung für Endnutzer-innen
      - end_user_application.py
      - weather_stations.json
      - Unterordnerstruktur inklusive Modelle

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Hiermit versichern wir, dass wir die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt haben. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

In der vorliegenden Arbeit wurde die Konzeption und Validierung der Schneeerkennung eigenständig von Kai König und die Konzeption und Validierung der Wolkenerkennung eigenständig von Alexander Maßmann bearbeitet. Andere Teile der Arbeit wurden gemeinsam entwickelt.

Ort	Datum	Unterschrift im Original
Ort	Datum	Unterschrift im Original