HAW HAMBURG

Bachelorarbeit

Niels Gandraß

Merkmalsextraktion durch Methoden des unüberwachten maschinellen Lernens zur Klassifikation von Aerosol-Rückstreuprofilen aus LIDAR-Ceilometern

Fakultät Technik und Informatik
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

Niels Gandraß

Merkmalsextraktion durch Methoden des unüberwachten maschinellen Lernens zur Klassifikation von Aerosol-Rückstreuprofilen aus LIDAR-Ceilometern

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Bachelor of Science Technische Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Michael Neitzke
Zweitgutachter: Prof. Dr.-Ing. Andreas Meisel

Eingereicht am: 29. April 2019

Niels Gandraß

Thema der Arbeit

Merkmalsextraktion durch Methoden des unüberwachten maschinellen Lernens zur Klassifikation von Aerosol-Rückstreuprofilen aus LIDAR-Ceilometern

Stichworte

Autoencoder, Clustering, Cluster-Optimierung, CNN, Dimensionsreduktion, LIDAR-Ceilometer, maschinelles Lernen, mehrschrittiges Lernen, Merkmalsextraktion, Rückstreuprofilanalyse, Signalfilterung, Sliding-Window, Soft-Clustering, unüberwachtes Lernen

Kurzzusammenfassung

In dieser Bachelorarbeit wird ein Verfahren zur unüberwachten Merkmalsextraktion aus Aerosol-Rückstreuprofilen vorgestellt. Hierbei werden LIDAR-Ceilometer Messdatensätze zuerst einer Vorverarbeitung unterzogen, um dann mithilfe eines Convolutional-Autoencoders in komprimierte Merkmalsvektoren überführt zu werden. Auf Basis dieser Merkmalsvektoren findet ein initiales Clustering statt, welches durch Minimierung der KL-Divergenz zu einer gewählten Zielverteilung weiter optimiert wird. Zwei Experimente, zunächst auf dem MNIST-Zifferndatensatz und im Anschluss mit realen Ceilometerdaten, haben die Funktionsfähigkeit des Ansatzes bestätigt. Erfolgreich konnten so einige meteorologische Phänomene in Clustern gruppiert und halbautomatisch mit Labels versehen werden. Neben den Implementationsdetails des Verfahrens werden außerdem einige Eigenschaften der vorliegenden Messdaten besprochen sowie ein umfangreicher Ausblick auf weitere Forschung in diesem Gebiet gegeben.

Niels Gandraß

Title of Thesis

Unsupervised Machine Learning: Feature Extraction for Classification of LIDAR-Ceilometer Aerosol-Backscatter Profiles

Keywords

autoencoder, backscatter analysis, clustering, cluster optimization, CNN, dimensionality reduction, feature extraction, LIDAR-ceilometer, machine learning, multi-stage learning, signal filtering, sliding-window, soft-clustering, unsupervised learning

Abstract

In this bachelor thesis a method for unsupervised feature extraction from aerosol backscatter profiles is proposed. After a first data preprocessing step the LIDAR-ceilometer datasets are converted into an compressed representation using an convolutional autoencoder. These encoded representations are then used for the computation of an initial cluster assignment which gets further optimized, trough minimizing the KL-divergence to an arbitrary target distribution, during consecutive training. Two experiments, the first based on the MNIST database of handwritten digits and the second based on real ceilometer aerosol backscatter data, were successfully executed. The proposed method has been successfully utilized, in order to group some meteorological phenomena into clusters which then labels were assigned to. In addition to the description of implementation details, multiple aspects of the ceilometer datasets are analyzed and a comprehensive outlook to further possible research in this field is given.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

Abkürzungen

CAE Convolutional Autoencoder.

CLI Kommandozeile (engl. *command-line interface*).

CNN Faltungsnetz (engl. *Convolutional Neural Network*).

DWD Deutscher Wetterdienst.

E-PROFILE EUMETNET Profiling Programme.

EUMETNET European Meteorological Network.

KL-Divergenz Kullback-Leibler-Divergenz.

LIDAR Light detection and ranging.

m ü. NHN Meter über Normalhöhennull (ugs. *Höhe über dem Meeresspiegel*).

MSE mean squared error (dt. *Mittlerer quadratischer Fehler*).

NetCDF Network Common Data Format.

PCA Hauptkomponentenanalyse (engl. *Principal Component Analysis*).

SNR Signal-Rausch-Verhältnis (engl. *signal-to-noise ratio*).

UTC Coordinated Universal Time.

UUID Universally Unique Identifier.

Symbolverzeichnis

$\tilde{\beta}$ / `attn_bsct` Menge der gemessenen Rückstreuung nach Kalibrierung in $m^{-1}sr^{-1}$.

$\tilde{\beta}_K$ / Komprimierte kalibrierte Rückstreuung in $m^{-1}sr^{-1}$.

β / `beta_raw` Menge der gemessenen Rückstreuung vor Kalibrierung in $m^{-1}sr^{-1}$.

C_L / `cal_value` Durch Rayleigh-Kalibrierung bestimmte LIDAR-Konstante.

chb / `--combine-height-bins` Kompressionsparameter.

cs / `--combine-samples` Kompressionsparameter.

$c(\tilde{x}) \in C(\tilde{X})$ / Wahrscheinlichkeitsverteilung der Clusterzugehörigkeit von $\tilde{x}$.

K / Kompressionsrate.

L / Verlustfunktion (engl. *loss function*).

L_c / Verlustfunktion des Clustering-Layers.

L_r / Verlustfunktion der Rekonstruktion durch den Autoencoder.

λ / Gewichtung der Verlustfunktion des Clustering-Layers L_c.

$m_i^{(t)}$ / Clustermittelpunkt (auch Clusterschwerpunkt) des i-ten Clusters zum Zeitpunkt (Iteration) t.

$\max_p$ / Maximalwert eines einzelnen Datenpunkts/Pixels über mehrere Messdatenschnitte in $m^{-1}sr^{-1}$.

$\min_p$ / Minimalwert eines einzelnen Datenpunkts/Pixels über mehrere Messdatenschnitte in $m^{-1}sr^{-1}$.

μ_p / Mittelwert eines einzelnen Datenpunkts/Pixels über mehrere Messdatenschnitte in $m^{-1}sr^{-1}$.

P / Gewählte Zielverteilung der Clusteroptimierung.

p_{ij} / Maß für die Ähnlichkeit zwischen $\tilde{x}_j$ und einem Punkt p_i aus P.

Q / Wahrscheinlichkeitsverteilung für die Clusterzugehörigkeit.

q_{ij} / Maß für die Ähnlichkeit zwischen $\tilde{x}_j$ und $m_i^{(t)}$.

S_i / i-tes Cluster eines Clusterings.

σ_p / Standardabweichung eines einzelnen Datenpunkts/Pixels über mehrere Messdatenschnitte in $m^{-1}sr^{-1}$.

σ_p^2 / Varianz eines einzelnen Datenpunkts/Pixels über mehrere Messdatenschnitte in $(m^{-1}sr^{-1})^2$.

ss / `--slice-steps` Verschub des Fensterverfahrens.

sw / `--slice-width` Fensterbreite des Fensterverfahrens.

t_{ovl} / Untere Detektionsgrenze relativ zum Ceilometer in m.

$x \in X$ / Eingangsdatensatz für Autoencoder.

$x' \in X'$ / Rekonstruktion von x durch Autoencoder.

$\tilde{x} \in \tilde{X}$ / Enkodierter Merkmalsvektor auf Basis von x.

$y \in Y$ / Zugewiesenes Label in verschlagwortetem Datensatz.

1 Einleitung

1.1 Motivation

Seit 2008 betreibt der Deutsche Wetterdienst ein deutschlandweites Ceilometer-Messnetz, welches mittlerweile über 150 vollautomatische Messstationen umfasst. Die hieraus stammenden Datensätze werden wiederum in globale Messnetzverbände, wie zum Beispiel dem E-PROFILE Programm des European Meteorological Networks, eingespeist. So ergibt sich eine Sammlung weltweiter Messdatensätze, die kontinuierlich ausgebaut wird.

Hauptbestandteil der aggregierten Daten sind sogenannte Aerosol-Rückstreuprofile (engl. *attenuated backscatter profiles*). Sie lassen Rückschlüsse über die Eigenschaften unterschiedlicher Luftschichten innerhalb der Troposphäre zu. Somit eignen sie sich zur Detektion verschiedener meteorologischer Phänomene. Dies umfasst beispielsweise die kontinuierlich laufende Überwachung auftretender Saharastaubereignisse sowie die Analyse von Aschewolken, welche unter anderem in Folge von Vulkanausbrüchen oder großen Flächenbränden entstehen können. Ferner erlauben die gesammelten Messdaten die Erkennung von mehreren Wolkentypen, von unterschiedlichen Niederschlagsarten sowie von einer Vielzahl weiterer meteorologisch relevanter Phänomene. Die Auswertung der gemessenen Aerosol-Rückstreuprofile erfolgt zur Zeit überwiegend manuell. Dies erfordert das tägliche Sichten und Bewerten einer großen Menge an Messdatensätzen über einen langen Zeitraum hinweg.

Die Motivation hinter dieser Arbeit besteht darin, den bisher manuell durchgeführten Klassifikationsprozess zu vereinfachen sowie zu vereinheitlichen. Da den gemessenen Rückstreuprofilen keinerlei maschinenlesbare Beschreibungen (engl. *labels*) beiliegen, ist eine Anwendung von Methoden des überwachten maschinellen Lernens ausgeschlossen. Demnach beschäftigt sich diese Arbeit mit der Merkmalsextraktion (engl. *feature extraction*) mithilfe von Methoden des unüberwachten maschinellen Lernens auf Basis der Ceilometerdaten.

1.2 Zielsetzung

Ziel dieser Arbeit ist die Entwicklung eines prototypischen Verfahrens zur unüberwachten Merkmalsextraktion aus den vorliegenden Ceilometer-Messdatensätzen. Das vorgestellte Verfahren soll mithilfe neuronaler Netze, genauer mit sogenannten Faltungsnetzen (engl. *convolutional neural networks*), realisiert werden. Hierbei soll sowohl das eigentliche Verfahren entwickelt und beschrieben, als auch eine Referenzimplementation inklusive Testrahmen erstellt und getestet werden.

Ein weiteres Ziel ist, dass diese Bachelorarbeit als Grundlage für weiterführende Forschungsprojekte in diesem Bereich dient. Daher ist ein umfangreicher Grundlagenteil sowie eine Analyse der zugrundeliegenden Eigenschaften der Ceilometer-Messdaten ebenso Bestandteil dieser Arbeit. Ferner ist die Implementation so aufzubauen, dass sie leicht für weitere Experimente in diesem Rahmen genutzt werden kann. Dies soll durch ein einfaches Austauschen von Netzarchitekturen sowie vorgefertigte Methoden zur Auswertung und Verifikation der Ergebnisse ermöglicht werden. Außerdem sind die einzelnen Teilschritte modular zu gestalten, um Zwischenergebnisse wiederverwenden zu können und somit reproduzierbare sowie vergleichbare Ergebnisse zu erhalten.

1.3 Aufbau der Arbeit

Ein einleitender Grundlagenteil erläutert insbesondere die der Informatik ferneren Zusammenhänge. Dies beinhaltet eine Beschreibung des zugrundeliegenden physikalischen Messprinzips und die hieraus resultierenden Eigenschaften, welche im Rahmen des vorgestellten Verfahrens zu berücksichtigen sind. Ferner wird der Aufbau der Messdatensätze besprochen und es wird auf die Grenzen der jeweiligen Messverfahren eingegangen.

Der dem Grundlagenteil angestellte Analyseteil beschäftigt sich zum einen mit der Beschreibung der zu detektierenden meteorologischen Phänomene als auch zum anderen mit den statistischen Eigenschaften der vorliegenden Messdaten. Außerdem werden für das vorgestellte Verfahren ausschlaggebende Parameter bestimmt.

In den darauf folgenden zwei Kapiteln werden zunächst die theoretischen Grundlagen des vorgestellten Verfahrens und später die Implementationsdetails der entwickelten Softwarelösung besprochen. Hierbei wird jeweils detailliert auf alle, für eine vollständige

Durchführung des Verfahrens benötigten Teilschritte eingegangen. Dies beinhaltet sämtliche Arbeitsschritte die vonnöten sind, um von einem rohen Messdatensatz zu sowohl einem komprimierten Merkmalsvektor als auch zu einer optimierten Clusterzuordnung zu gelangen.

Die Ergebnisse aller durchgeführten Experimente finden sich im Kapitel 6. Dies umfasst jeweils eine Versuchsbeschreibung, die Details der verwendeten neuronalen Netze sowie eine Auswertung inklusive Ergebnisbesprechung.

Abschließend wird im letzten Kapitel dieser Arbeit das Ergebnis der durchgeführten Experimente zusammenfassend wiedergegeben. Ferner wird ein Ausblick auf Optimierungsmöglichkeiten des vorgestellten Verfahrens als auch auf mögliche weiterführende Forschung in diesem Themengebiet gegeben.

2 Grundlagen

In diesem Kapitel werden die zugrunde liegenden Prinzipien des verwendeten Messverfahrens vorgestellt und erläutert. Hierbei wird ein besonderer Fokus auf die der Informatik ferneren Grundlagen, welche für die Verarbeitung und Analyse der Daten benötigt werden, gelegt.

Dies beinhaltet unter anderem die Vorstellung von verschiedenen optischen Verfahren zur Bestimmung atmosphärischer Parameter, von physikalischen Grundlagen des hier verwendeten Messprinzips sowie von den Arten der detektierbaren meteorologischen Ereignisse. Außerdem sind Details zum Messnetz, aus welchem die in dieser Arbeit genutzten Daten stammen, Teil dieses Kapitels. Ferner wird kurz auf die verschiedenen Datenformate, in welchen die Messdaten vorliegen, eingegangen.

2.1 Optische Bestimmung der Wolkenuntergrenze

Eine zuverlässige Ermittlung der Wolkenuntergrenze (auch Wolkenbasis) ist für die Meteorologie als auch u.A. für die Luftfahrt von großer Bedeutung. Sie ist definiert als die Höhe, in welcher die Lufttemperatur dem Taupunkt gleicht, die Luft somit vollständig mit Wasserdampf gesättigt ist und dadurch Wolken entstehen. Angegeben wird die Wolkenuntergrenze als Meter über Normalhöhennull (ugs. *Höhe über dem Meeresspiegel*) (m ü. NHN).

Gemessen werden kann die Wolkenuntergrenze mit so genannten Ceilometern (auch Ceilograph). Hierbei handelt es sich um Messgeräte, welche optische Verfahren nutzen, um die gesuchten Parameter zu bestimmen. Die relevanten Typen sind zum einen das triangulationsbasierte sowie zum anderen das moderne LIDAR-Ceilometer.

2.1.1 Triangulationsbasierte Ceilometer

Herkömmliche Ceilometer bestehen aus einem starken Scheinwerfer und einer zugehörigen Bodenstation. Die Lichtquelle strahlt von einem wohl definierten Punkt aus senkrecht nach oben. Ist eine Bewölkung vorhanden, so erzeugt der Scheinwerfer einen Lichtfleck auf der Wolkenunterseite. Von der Bodenstation aus kann die Höhe der Wolkenuntergrenze nun mittels Triangulation bestimmt werden (siehe Abbildung 2.1).

Ferner ist anzumerken, dass dieses Verfahren nur bei starker Bewölkung oder bei Nacht zuverlässig einsetzbar ist.

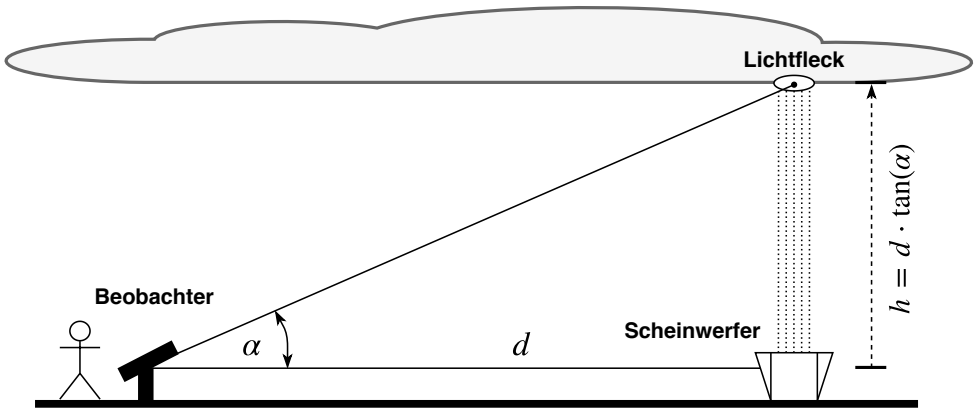


Abbildung 2.1: Schematische Darstellung des Messprinzips eines triangulationsbasierten Ceilometers

2.2 Optische Messung vertikaler Aerosol-Rückstreuprofile

Triangulationsbasierte Ceilometer, wie in Abschnitt 2.1.1 beschrieben, erlauben lediglich die Bestimmung der Wolkenuntergrenze. Neuere LIDAR-Ceilometer hingegen ermöglichen zusätzlich das Aufzeichnen vertikaler Rückstreuprofile, aus welchen sich Rückschlüsse auf die Eigenschaften der vorhandenen Luftschichten ziehen lassen.

2.2.1 LIDAR-Ceilometer

LIDAR-Ceilometer (auch Laser-Ceilometer) bestehen aus einem Laser sowie einer Detektor-Einheit. Der Laser emittiert Lichtpulse senkrecht in die Atmosphäre. Die in den ver-

schiedenen Luftschichten vorhandenen Teilchen streuen das Laserlicht je nach Dichte und Zusammensetzung des vorgefundenen Aerosols. Aus den auf den Detektor treffenden Anteilen des gestreuten Lichts ergibt sich die Intensität der Rückstreuung. Hierbei kann aus der gemessenen Zeit zwischen Aussendung des Pulses und Ankunft am Detektor die Höhe der momentan betrachteten Luftschicht bestimmt werden. Dies ermöglicht es, ein Rückstreuprofil verschiedener, übereinander liegender Schichten gleichzeitig mit nur einer Messung zu bestimmen.

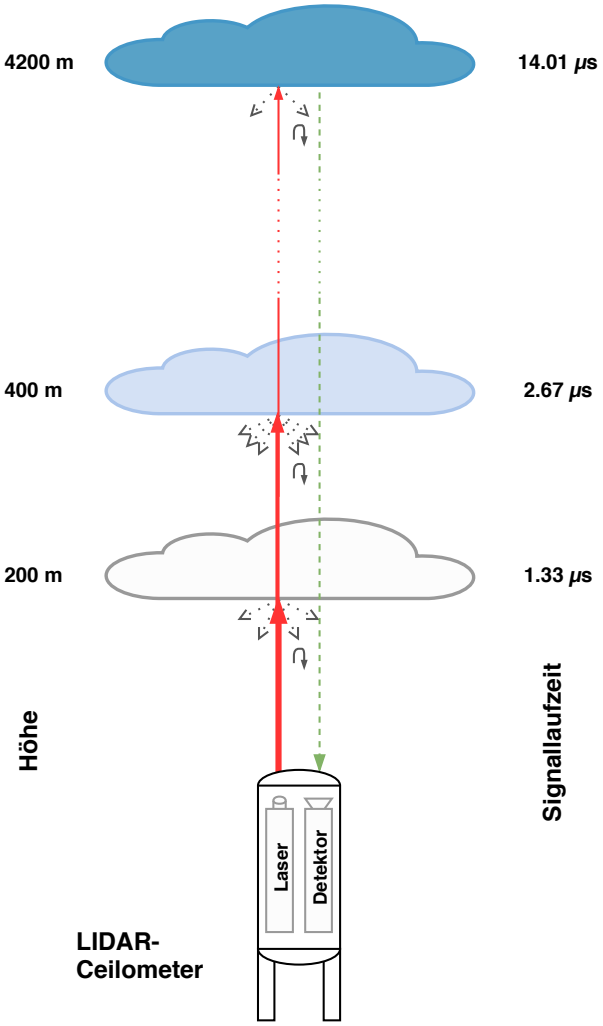


Abbildung 2.2: Schematische Darstellung des Messprinzips eines LIDAR-Ceilometers

Im Vergleich zum triangulationsbasierten Ceilometer erlaubt ein laserbasiertes Ceilometer also nicht nur die Höhe der Wolkenuntergrenze zu bestimmen. Zusätzlich können Aerosol-Rückstreuprofile verschiedener Höhenschichten im Rahmen einer einzelnen

Messung gebildet werden. Dies erlaubt die Charakterisierung verschiedenster meteorologischer Ereignisse. Ferner können beschriebene Gerätetypen unabhängig von aktueller Bewölkung und Tageszeit betrieben werden.

2.2.2 Eigenschaften des Messprinzips

Durch das in Abschnitt 2.2.1 beschriebene Verfahren ergeben sich einige Eigenschaften, die bei der Auswertung der Daten berücksichtigt werden müssen.

2.2.2.1 Abnahme der Intensität / „Schattenbildung“

Der Laser emittiert Lichtpulse mit einer festgelegten Intensität. Trifft ein Lichtpuls auf Partikel, so wird ein Teil der Photonen gestreut. Der Puls nimmt also mit zunehmender Höhe in seiner Intensität ab. Dies ist bei der Auswertung der auf den Detektor treffenden Teile des Lichtpulses zu berücksichtigen.

Zieht beispielsweise eine dichte Wasserwolke über das Ceilometer, so wird ein Großteil der Photonen von dieser gestreut und nur ein kleiner Teil des ausgesendeten Lichts dringt durch die Wolke. Je nach Dichte der Wolke kann es somit vorkommen, dass ein Großteil des Lichts gestreut wird und es oberhalb der Wolke zu einer so genannten Schattenbildung kommt. In solch einer Situation kann keine zuverlässige Aussage über das Rückstreuprofil oberhalb der Wolke gemacht werden, da hierdurch das Signal-Rausch-Verhältnis (engl. *signal-to-noise ratio*) (SNR) zu gering wird.

2.2.2.2 Instabilität einzelner Messungen

Die gemessene Rückstreuung kann sich zwischen einzelnen Messungen, je nach Ausrichtung der getroffenen Teilchen sowie der Dichte der Aerosolschichten, stark unterscheiden. Ferner kann es vorkommen, dass Licht von Störpartikeln/-objekten gestreut wird, welche für das Messergebnis irrelevant sind (z.B. Vögel). Um die Auswirkung auf das Messergebnis zu reduzieren, wird häufig über mehrere unmittelbar auf einander folgende Messungen gemittelt. Dies hat den Vorteil, dass hierdurch das SNR deutlich verbessert werden kann.

Nichtsdestotrotz kann es vorkommen, dass einzelne Rückstreuprofile Artefakte von Störquellen aufweisen. Mit diesen ist bei der Interpretation der Daten geeignet umzugehen.

2.2.2.3 Overlap-Artefakte in geringer Höhe

Da Laser und Detektor bauformbedingt einen gewissen Abstand zu einander haben, kann die Detektor-Einheit erst ab einer gewissen Höhe die gesamte Rückstreuung des Lasers detektieren (siehe Abbildung 2.3). Dies wird als sogenanntes Overlap-Problem bezeichnet. Für den Übergangsbereich ($ovl =]0, 1[$) ist eine Korrektur des Signals zwar möglich, jedoch stark von unterschiedlichen Geräteparametern abhängig und daher sehr aufwendig durchzuführen. Eine zuverlässige Signalnutzung ist demnach erst ab einer vollständig möglichen Detektion der Laserrückstreuung ($ovl = 1$) gewährleistet.

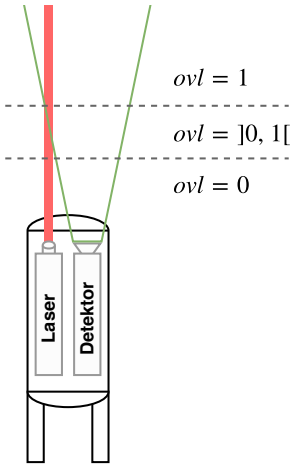


Abbildung 2.3: Overlap-Problem eines LIDAR-Ceilometers

Durch das Overlap-Problem entstandene Artefakte können mithilfe eines aufwendigen Verfahrens [1] korrigiert werden. Für diese Arbeit reicht es jedoch aus, eine Mindestdetektionshöhe einzuführen, unterhalb welcher die aufgenommenen Messwerte verworfen werden. Die Größe dieses Bereichs hängt stark von dem verwendeten Gerät ab.

2.2.3 Grenzen optischer Messverfahren

Die hier verwendeten Daten stammen von LIDAR-Ceilometern, welche mit einem monochromatischen Laser ausgestattet sind. Dadurch können zwar Rückschlüsse auf die Dichte und das allgemeine Vorhandensein der verschiedenen Aerosolschichten gezogen werden, jedoch reichen diese Daten nicht zur genauen Bestimmung der Zusammensetzung einer solchen Schicht aus. Um weitere Aussagen treffen zu können ist es von Nöten, die Rückstreuprofile mit Daten aus weiteren Messgeräten zu kombinieren.

Nichtsdestotrotz ermöglichen LIDAR-Ceilometer eine kostengünstige Erkennung der in Abschnitt 3.1 aufgeführten meteorologischen Ereignisse.

2.3 Ceilometer Messnetze

Um Aussagen über die Einbringung sowie Ausbreitung von Aerosolschichten treffen zu können, werden Daten von mehreren geographisch verteilten Ceilometern benötigt. Hierzu betreiben verschiedene Organisationen Messnetze, welche die Daten einzelner Geräte zusammenführen. Ferner existieren Verbände, die Daten aus mehreren Messnetzen aggregieren.

2.3.1 Messnetz des Deutschen Wetterdienstes (DWD)

Mit mittlerweile mehr als 150 Stationen[1] betreibt der Deutsche Wetterdienst (DWD) seit 2008 ein umfangreiches deutschlandweites Messnetz aus LIDAR-Ceilometern. Die im Rahmen dieser Arbeit genutzten Datensätze stammen aus besagtem Messnetz.

2.3.1.1 Eingesetzte Messgeräte

Im homogenen Messnetz des DWD werden CHM 15k „NIMBUS“[2] LIDAR-Ceilometer der Firma Lufft (siehe Abbildung 2.4) eingesetzt. Diese sind mit einem 1064 nm Laser ausgestattet und erlauben die Messung von Aerosol-Rückstreuprofilen bis in eine Höhe von maximal 15 km. Die Datenerfassung erfolgt durch Zählen der zurückgestreuten und vom Detektor registrierten Photonen.

2.3.1.2 Messdaten

Die in Abschnitt 2.3.1.1 beschriebenen Ceilometer werden vollautomatisch betrieben und über die Netzanbindung des Geräts ferngewartet. Aufgenommene Messdatensätze werden im NetCDF-Format gespeichert und an die Server des DWD weitergeleitet. Details zu den erfassten Daten sowie dem Datenformat sind in Abschnitt 2.4 beschrieben.

[1] Stand Dezember 2018

[2] CHM15K Details: https://www.lufft.com/products/cloud-height-snow-depth-sensors-288/ceilometer-chm-15k-nimbus-2300/ (Abgerufen am 20.03.2019)

(a) Außenansicht (b) Laser und Detektor-Einheit

Abbildung 2.4: CHM 15K „NIMBUS“ LIDAR-Ceilometer der Firma Lufft [2]

2.3.2 E-PROFILE Messnetz des EUMETNET

Das EUMETNET Profiling Programme (E-PROFILE)[3] ist ein europaweites Messnetz, welches Daten aus Wind-Profilern sowie LIDAR-Ceilometern sammelt und zur Verfügung stellt. Betrieben wird es von dem European Meteorological Network (EUMETNET). Hier werden auch die Daten des in Abschnitt 2.3.1 beschriebenen DWD Messnetzes eingespeist.

Aktuelle sowie historische Datensätze können auf der Website des Projekts eingesehen werden: https://e-profile.eu/.

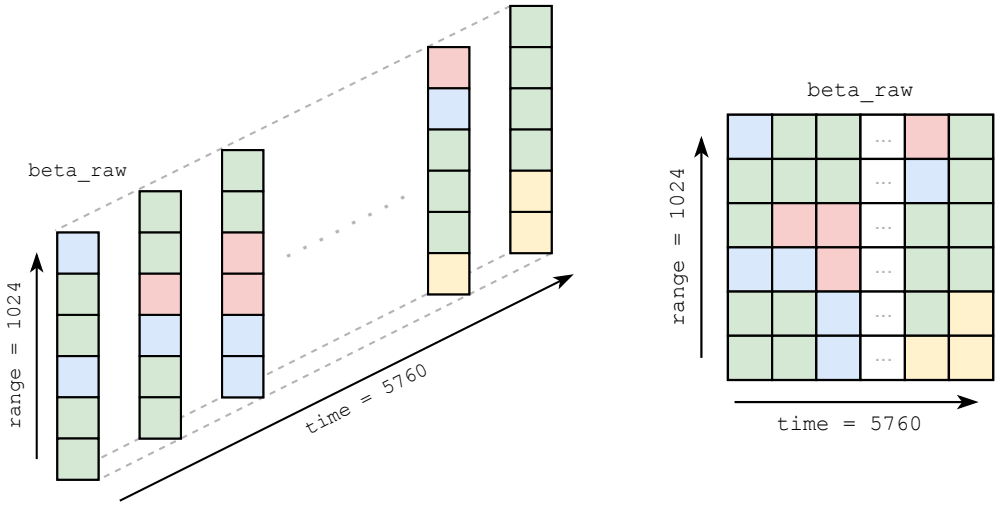
2.4 Format der LIDAR-Ceilometer Messdaten

Die von den verwendeten CHM 15k Ceilometern aufgezeichneten Rückstreuprofile werden im sogenannten NetCDF-Format abgelegt. Im folgenden Abschnitt wird sowohl auf das verwendete Format als auch auf die in den Datensätzen vorhandenen Variablen und Messwerte eingegangen.

[3] E-PROFILE Website: http://eumetnet.eu/activities/observations-programme/current-activities/e-profile/ (Abgerufen am 20.03.2019)

2.4.1 Network Common Data Format (NetCDF)

Das Network Common Data Format (NetCDF) [3] ist ein selbstbeschreibendes binäres Dateiformat. Es erlaubt das Ablegen von beliebig strukturierten, ein- sowie mehrdimensionalen, Array-basierten Daten (Beispiel siehe Abbildung 2.5). Dadurch ermöglicht es einen einfachen und flexiblen Austausch wissenschaftlicher Datensätze. Die hier verwendeten Daten liegen im `NETCDF3_CLASSIC` Format [4] vor.



(a) Darstellung als separate Messungen (b) Darstellung als 2D-Array

Abbildung 2.5: Aufbau der `beta_raw` NetCDF Variablen im CHM 15k Messdatensatz

Dadurch, dass der NetCDF Standard ein offener Standard ist, existieren eine Vielzahl von Programmbibliotheken, die das NetCDF Format unterstützen. Im Rahmen dieser Arbeit wurden folgende Werkzeuge genutzt:

- `netcdf4-python`[4] Eine Python Schnittstelle zur NetCDF C Bibliothek.
- `Panoply`[5] Ein vom NASA Goddard Institute for Space Studies in Java geschriebener NetCDF-Dateibetrachter.

[4] `netcdf4-python` Projekt-Website: https://github.com/Unidata/netcdf4-python (Abgerufen am 20.03.2019)
[5] `Panoply` Projekt-Website: https://www.giss.nasa.gov/tools/panoply/ (Abgerufen am 20.03.2019)

2.4.2 Relevante Variablen / Dimensionen

Im Folgenden wird auf den Teil der CHM 15k Messdatensätze genauer eingegangen, der für die spätere Weiterverarbeitung der Messdaten relevant ist.

2.4.2.1 Globale Attribute

Jede NetCDF Datei beinhaltet eine Menge globaler Attribute. Diese geben unter anderem Auskunft über den Namen des Gerätestandorts, beinhalten das Datum, an welchem die Datei angelegt wurde, sowie weitere Parameter (siehe Quellcode 2.1).

```
1  :comment = "SW-Update 26.07.2018 - MW";
2  :software_version = "12.12.1 2.13 0.754 1";
3  :title = "CHM15k Nimbus";
4  :wmo_id = 10471; // int
5  :month = 12; // int
6  :source = "CHM080079";
7  :overlap_file = "TUB080021 (2012-08-20 09:31:51)";
8  :serlom = "TUB080021";
9  :location = "Leipzig";
10 :year = 2018; // int
11 :device_name = "CHM080079";
12 :institution = "DWD";
13 :day = 2; // int
```

Quellcode 2.1: Globale Attribute in NetCDF Messdatensatz

2.4.2.2 Gemessene Rückstreuintensität (`beta_raw`)

Die vom Ceilometer gemessene Rückstreuintensität β wird in der Variablen `beta_raw` abgelegt. Hierbei handelt es sich um ein zweidimensionales Array mit der Form `(time=5760, range=1024)` (siehe Abbildung 2.5b).

Eine NetCDF-Datei beinhaltet hierbei alle 5760 Messwerte des jeweiligen Tages. Diese ergeben sich aus der Messfrequenz von 15^{-1}Hz. Je Messung werden die Rückstreuintensitäten aller 1024 unterschiedenen Höhenschichten aufgenommen. Somit ergibt sich eine Gesamtdatenmenge von 5.898.240 Messwerten des Datentyps `float` pro Tag.

```
float beta_raw(time=5760, range=1024);
  :units = "";
  :long_name = "normalized range corrected signal
               ((P_raw / lp) - b) / (cs * o(r) * p_calc) * r * r with
               P_raw = sum(P_raw_hr) * range_gate_hr / range_gate";
```

Quellcode 2.2: `beta_raw` Variable im NetCDF Messdatensatz

Anmerkung: Die in `beta_raw` abgelegte Rückstreuintensität ist unkalibriert. Details zur benötigten Kalibrierung der Messwerte finden sich in Abschnitt 2.4.3.

2.4.2.3 Höhenskala der Messungen (`range`)

Bei jeder Messung werden Rückstreuintensitäten für 1024 unterschiedliche Höhenschichten aufgezeichnet. Die Variable `range` besteht aus einem eindimensionalen Array, welches zu jedem der in `beta_raw` abgelegten Messwerte die zugehörige Höhe beinhaltet. Angegeben wird die Höhe relativ zum Ceilometer in Metern.

```
float range(range=1024);
  :units = "m";
  :long_name = "distance from lidar";
  :axis = "Z";
```

Quellcode 2.3: `range` Variable im NetCDF Messdatensatz

Anmerkung: Zur Berechnung der absoluten Höhe in m ü. NHN ist die Referenzhöhe des Ceilometers (`altitude`) (siehe Abschnitt 2.4.2.5) zu berücksichtigen.

2.4.2.4 Zeitpunkte der Messungen (`time`)

Analog zu `range` gibt `time` die Zeit jeder einzelnen in `beta_raw` beinhalteten Messung an. Diese liegt als Sekunden seit dem 01.01.1904 in Coordinated Universal Time (UTC) vor.

```
double time(time=5760);
  :units = "seconds since 1904-01-01 00:00:00.000 00:00";
  :long_name = "time UTC";
  :axis = "T";
```

Quellcode 2.4: `time` Variable im NetCDF Messdatensatz

Achtung: Der vorliegende Zeitstempel unterscheidet sich vom weit verbreiteten Linux-Zeitstempel (Sekunden seit dem 01.01.1970) und muss daher gegebenenfalls vor der Weiterverarbeitung konvertiert werden.

2.4.2.5 Referenzhöhe des Ceilometers (`altitude`)

Die Höhe des Ceilometers in m ü. NHN ist in der skalaren Variablen `altitude` abgelegt. Sie wird benötigt, um die in `range` hinterlegten relativen Höhen auf absolute Höhen in m ü. NHN abzubilden.

```
1 float altitude;
2   :units = "m";
3   :long_name = "altitude of ceilometer
4                 above mean sea level";
```

Quellcode 2.5: `altitude` Variable im NetCDF Messdatensatz

2.4.2.6 Detektierte Wolkenuntergrenze (`cbh`, `cbe`)

Die verwendeten Ceilometer detektieren automatisch, sofern vorhanden, die Höhe der Wolkenuntergrenze. Hierbei werden bis zu 3 übereinander liegende Wolkenschichten erkannt. Abgelegt werden die Messwerte in der Variablen `cbh` der Form `(time=5760, layer=3)`. Hierbei entspricht ein Wert von -1 keiner detektierten Wolkenschicht und positive Werte der zum Ceilometer relativen Höhe der Wolkenuntergrenze in Metern.

Die Varianz der gemessenen Höhen wird zusätzlich in der Variablen `cbe` derselben Form gespeichert.

```
1 short cbh(time=5760, layer=3);
2   :units = "m";
3   :long_name = "cloud base height";
4
5 short cbe(time=5760, layer=3);
6   :units = "m";
7   :long_name = "cloud base height variation"
```

Quellcode 2.6: `cbh` und `cbe` Variablen im NetCDF Messdatensatz

2.4.2.7 Maximale Detektionshöhe (`mxd`)

Durch die in Abschnitt 2.2.2.1 beschriebene Abnahme der Rückstreuintensität bzw. durch Schattenbildung ergibt sich eine maximale Detektionshöhe. Oberhalb dieser ist das SNR so gering, dass keine zuverlässigen Messwerte mehr aufgenommen werden können. Diese Höhe bestimmt das Ceilometer und legt sie für jede durchgeführte Messung in der Array-Variablen `mxd` ab. Auch hier wird die Höhe in Metern relativ zum Ceilometer angegeben.

```
1 short mxd(time=5760);
2   :units = "m";
3   :long_name = "maximum detection height";
```

Quellcode 2.7: `mxd` Variable im NetCDF Messdatensatz

2.4.3 Rayleigh-Kalibrierung der Rückstreuprofile

Da die gemessenen Rückstreuintensitäten β unter verschiedenen Bedingungen (z.B. durch die Temperatur des Detektors) schwanken, müssen sie mit Hilfe der sogenannten LIDAR-Konstanten C_L kalibriert werden [5]. Das Vorgehen zur Bestimmung dieser Konstanten mithilfe der Rayleigh-Kalibrierung soll nicht Bestandteil dieser Arbeit sein, jedoch ist es unabdingbar, die in den Messdatensätzen vorhandenen Werte zu kalibrieren.

Die stundenaktuellen Werte für C_L werden in einer separaten Datei je Station mitgeliefert. Aus diesen ergibt sich die kalibrierte Rückstreuintensität $\tilde{\beta} = \beta \ / \ C_L$.

Im Weiteren wird immer mit der kalibrierten Rückstreuintensität $\tilde{\beta}$ gearbeitet.

3 Analyse

Im folgenden Kapitel werden zuerst die im Rahmen dieser Arbeit relevanten verschiedenen meteorologischen Phänomene vorgestellt, die mithilfe der aufgenommenen Rückstreuprofile aus LIDAR-Ceilometer detektierbar sind. Weiter wird auf die Erkenntnisse aus der Analyse vorliegender Messdaten eingegangen. Hierbei werden zudem Parameter für Teilschritte des in Abschnitt 4 vorgestellten Verfahrens bestimmt.

3.1 Detektierbare Ereignisse

Die eingesetzten LIDAR-Ceilometer erlauben die Detektion einer Vielzahl verschiedener meteorologischer Phänomene. Die im Rahmen dieser Arbeit betrachteten Ereignisse lassen sich grob in drei Kategorien einteilen: Wolken, Aerosolschichten und Niederschlag. Ferner kann es ebenfalls vorkommen, dass kein Ereignis detektiert wurde. Dies ist beispielsweise bei einem klaren sonnigen Tag der Fall und zeichnet sich durch gleichmäßiges Hintergrundrauschen im Rückstreuprofil aus.

Darüber hinaus können beispielsweise Frontdurchgänge oder sogar, bei Betrachtung von Messungen mehrerer Geräte, ganze Trajektorien einzelner Aerosolschichten erkannt werden. Dies soll jedoch nicht Gegenstand dieser Arbeit sein.

3.1.1 Wolken

Wolken zeichnen sich im Allgemeinen durch eine sehr hohe Rückstreuintensität $\tilde{\beta}$ aus. Abbildung 3.1 zeigt eine so genannte Cirruswolke. Hierbei handelt es sich um eine reine Eiswolke, welche sich meist in größeren Höhen (typischerweise 8 - 12 km) finden lässt. Häufig weisen diese Wolken eine Dicke von 0.5 bis 2 km auf.

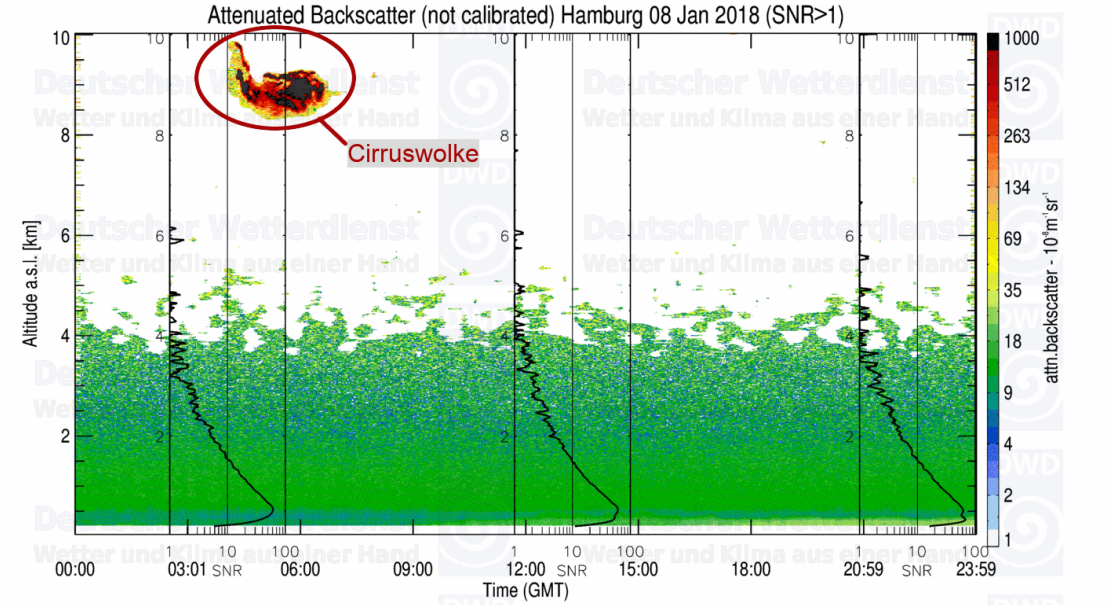


Abbildung 3.1: Ceilometer-Rückstreuprofil mit detektierter Cirruswolke

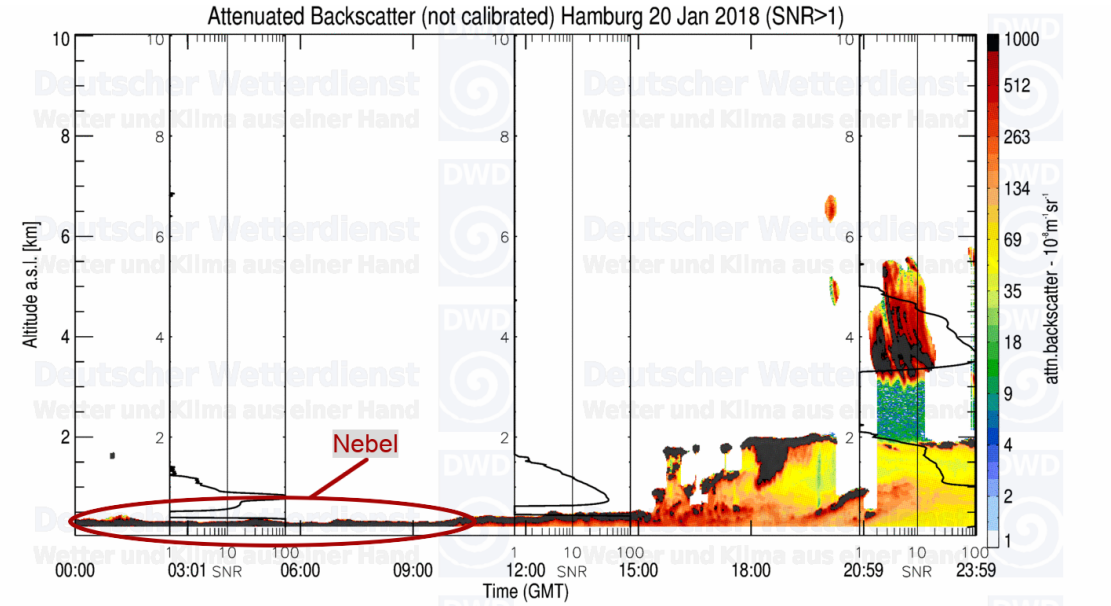


Abbildung 3.2: Ceilometer-Rückstreuprofil mit detektiertem Nebel

Im Vergleich zu Cirruswolken weisen Wasserwolken (siehe Abbildung 3.3) meist noch eine weitaus höhere Rückstreuintensität auf. Hierbei lässt sich gut die Reduktion der Signalintensität und die damit verbundene Abnahme des SNR (siehe Kapitel 2.2.2.1) oberhalb der Wolke feststellen. Wasserwolken sind meist nur einige hundert Meter dick und als dünne Streifen auf dem Rückstreuprofil zu erkennen. Sie zeichnen sich außerdem dadurch aus, dass es direkt oberhalb der Wolke zuerst zu einem Unterschwinger[1] (*engl. undershoot*), gefolgt von einem Überschwinger[1] (*engl. overshoot*) in der gemessenen Rückstreuintensität kommt.

Andere gut detektierbare Typen sind Gewitterwolken sowie Nebel (siehe Abbildung 3.2). Erstere sind mehrere Kilometer dick und im Rückstreuprofil als V-förmige, kegelartige Strukturen erkennbar. Nebel hingegen besitzt Bodenkontakt und zeichnet sich analog zu Wasserwolken durch eine geringe Dicke und hohe Rückstreuintensität aus.

3.1.2 Aerosolschichten

Aerosole sind heterogene Gemische aus festen oder flüssigen Schwebeteilchen in einem Gas. Diese Gemische lassen sich als flächige Schichten mit meist mittlerer Rückstreuintensität erkennen. Klassische Beispiele in der Meteorologie sind Saharastaub sowie Aschewolken nach Ausbruch eines Vulkans. Abbildung 3.3 zeigt Aerosolschichten in mehreren Höhenlagen sowie deren Durchmischung.

Es wird zwischen Aerosolen in der so genannten planetaren Grenzschicht, dem untersten Teil der Erdatmosphäre, und Schichten in größeren Höhen unterschieden. Die genaue Bestimmung der in einem Aerosol enthaltenen Partikel ist mit dem hier verwendeten Messverfahren nicht möglich. Hierzu sind Daten von weiteren Messgeräten erforderlich. Jedoch kann über die Menge des zurück gestreuten Laserlichts eine grobe Aussage über die Dichte der jeweiligen Schichten getätigt werden und somit z.B. Vulkanasche von Kraftfahrzeug-Emissionen unterschieden werden.

[1] Über- bzw. Unterschwingen tritt nach Änderung eines Signals auf, wenn das gemessene Signal den neuen Realwert nicht sofort erreicht, sondern zuerst über bzw. unter den eigentlichen Wert hinausschießt, bevor es sich auf den realen Signalwert einstellt.

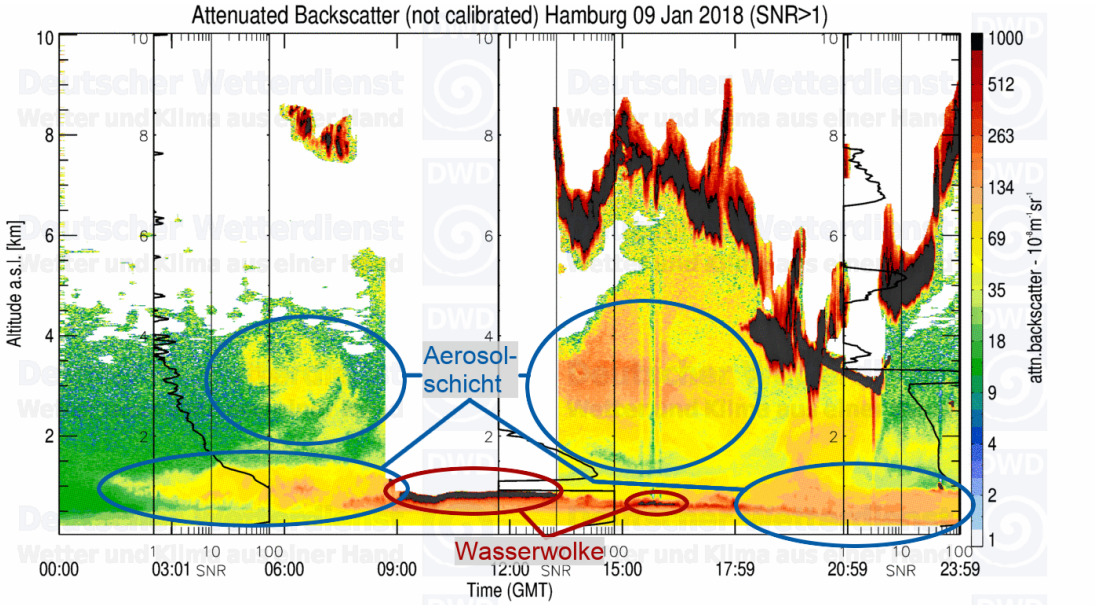


Abbildung 3.3: Ceilometer-Rückstreuprofil mit Aerosolschichten sowie Wasserwolken

3.1.3 Niederschlag

Jede Form von Kondensation atmosphärischen Wassers, welches infolge der Schwerkraft in Richtung Erde fällt, wird als Niederschlag bezeichnet. Hierbei wird im Rahmen dieser Arbeit zwischen drei Niederschlagstypen unterschieden.

Die geläufigste hier behandelte Form ist der klassische Niederschlag, welcher den Boden erreicht. Dieser zeichnet sich durch lange Streifen im Rückstreuprofil aus, welche sich unterhalb einer Wolke bilden und den Erdboden erreichen. Dieses Phänomen ist in Abbildung 3.4 zu erkennen. Aufgrund der großen Abstände zwischen einzelnen Regentropfen ist die Reduktion der Signalintensität durch die fallenden Tropfen nicht so stark, dass es wie bei Wasserwolken zu einer Schattenbildung kommt. Dies hat somit die beschriebenen Streifen im Rückstreuprofil zur Folge.

Erreicht der Niederschlag hingegen nicht den Erdboden, so spricht man von so genannten Fallstreifen. Hierbei verdunstet das kondensierte Wasser wieder, bevor es auf dem Erdboden auftreffen kann. Je nach Temperatur der Wolke kann der Niederschlag auch in Form von Eiskristallen aus der Wolke fallen. Schmilzt das Eis auf dem Weg zum Erdboden, so bildet sich an dieser Stelle eine so genannte Schmelzschicht. Diese zeichnet sich durch einen dünnen Strich mit geringerer Rückstreuintensität als der Fallstreifen aus.

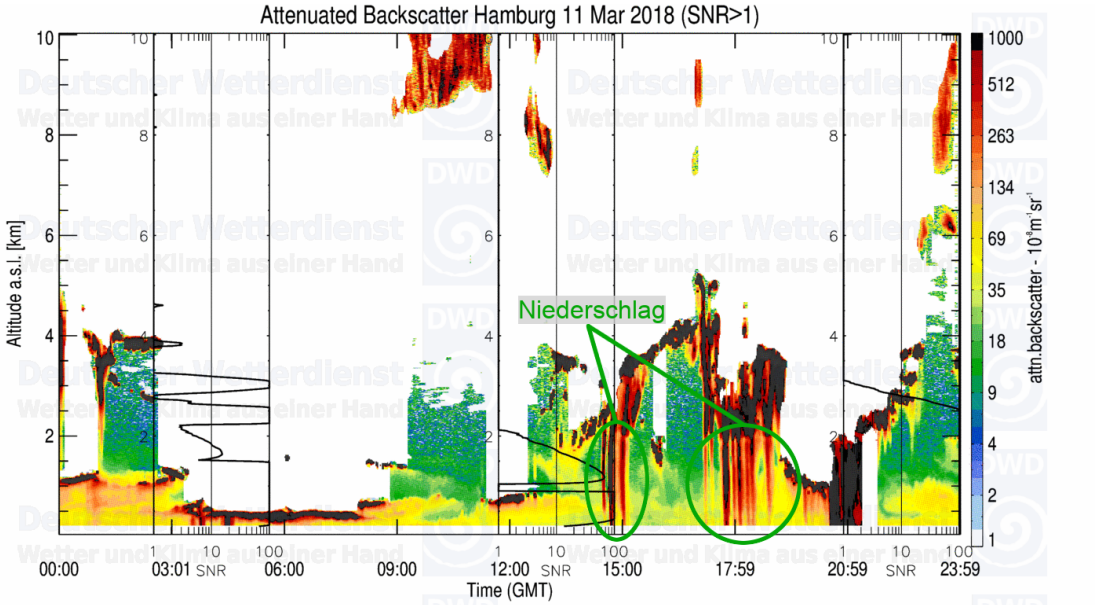


Abbildung 3.4: Ceilometer-Rückstreuprofil mit Niederschlag

3.2 Statistische Betrachtung der Datensätze

Um einen ersten Eindruck der Messdaten zu gewinnen, wurde eine Histogrammanalyse aller Datensätze, welche jeweils über ein Jahr angefallen sind, für ausgewählte Ceilometer durchgeführt. Die Verteilung der kalibrierten Rückstreuintensitäten $\tilde{\beta}$ ist am Beispiel des Ceilometers in Leipzig im Jahre 2018 in Abbildung 3.5 grafisch dargestellt. Ferner wurde der Wertebereich von $\tilde{\beta}$ (siehe Tabelle 3.1) sowie Mittelwert und Standardabweichung bestimmt.

Die aus der statistischen Analyse gewonnenen Daten werden im Rahmen einer späteren Normalisierung (siehe Abschnitt 4.4.2) genutzt. Hierfür ergibt sich, dass eine simple Min/Max-Skalierung der Eingangsdaten aufgrund der kompakten Werteverteilung gut möglich ist.

Maximale Rückstreuintensität $\tilde{\beta}_{max}$	3.25 e−03 $m^{-1}sr^{-1}$
Minimale Rückstreuintensität $\tilde{\beta}_{min}$	−2.38 e−05 $m^{-1}sr^{-1}$

Tabelle 3.1: Wertebereich der Rückstreuintensität $\tilde{\beta}$ aller Messdatensätze aus dem Jahr 2018 des Ceilometers in Leipzig

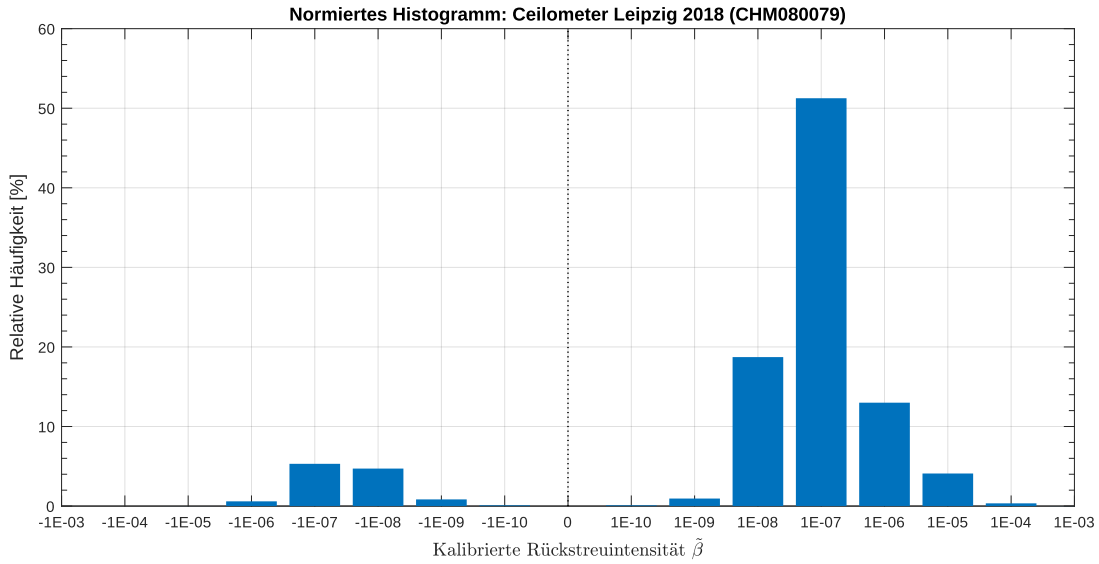


Abbildung 3.5: Normiertes Histogramm aller Messdatensätze aus dem Jahr 2018 des Ceilometers in Leipzig

3.3 Komprimierbarkeit von Rückstreuprofilen

Im Rahmen der Analyse wurde ebenfalls die Komprimierbarkeit der Messdaten geprüft. Ziel ist hierbei das Erreichen einer trivialen Dimensionsreduktion der Eingangsdaten, die für das in dieser Arbeit vorgestellte Verfahren benötigt werden.

Hierzu wurden die Auswirkungen einer Mittelwertbildung durch Kombination mehrerer benachbarter Höhenbereiche einer einzelnen Messung betrachtet. Außerdem wurde die Kompression durch das Zusammenfassen mehrerer Messungen bewertet. Zur Beurteilung der aus der Komprimierung entstehenden Verluste wurden Messdaten verschiedener meteorologischer Phänomene jeweils in unterschiedlichen Stärken komprimiert und die Resultate ausgewertet.

Abbildung 3.6 zeigt einen Ausschnitt aus einem Rückstreuprofil, welcher stark ausgeprägte Strukturen verschiedenster Intensitäten enthält. Hierbei entspricht der Parameter `--combine-height-bins` (*chb*) einer Mittelwertbildung über mehrere Höhenbereiche innerhalb einer einzelnen Messung und der Parameter `--combine-samples` (*cs*) einer Kombination mehrerer Messungen. Es zeigt sich, dass für stark ausgeprägte Strukturen problemlos eine Kombination von jeweils bis zu 4 Höhenbereichen sowie Messungen möglich ist, ohne zu große Informationsverluste zu erzeugen. Ferner ist festzuhalten, dass

selbst eine Kombination von bis zu 8 Einzelmessungen in diesem Szenario noch möglich ist.

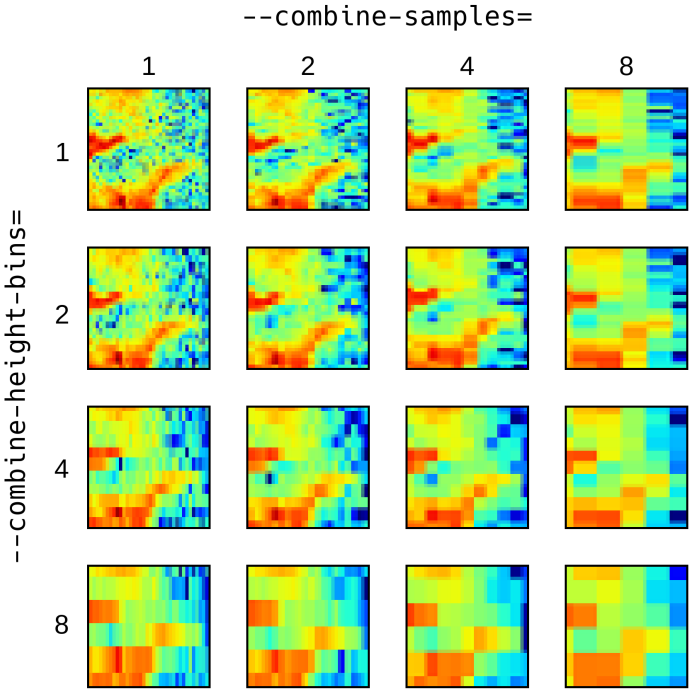


Abbildung 3.6: Vergleich verschiedener Komprimierungsintensitäten anhand stark ausgeprägter Strukturen

Im Unterschied zur vorherigen, zeigt Abbildung 3.7 nur sehr schwach ausgeprägte Strukturen mit geringerer Varianz in der Rückstreuintensität. Wenden wir hier, wie für Abbildung 3.6 vorgeschlagen, eine Mittelwertbildung über 4 Höhenbereiche sowie 8 Einzelmessungen an, so kommt es zum Verschwimmen der Strukturgrenzen. Für Ausschnitte dieser Art bietet sich eine Kombination von 2 Höhenbereichen sowie 8 Einzelmessungen an.

Die Auswertung weiterer, in der Anzahl über die beiden vorgestellten Beispiele hinausgehenden, Strukturen hat abschließend ergeben, dass die Wahl einer Kombination von jeweils 4 Höhenbereichen sowie Messungen ein gutes Mittelmaß darstellt. Hierdurch ergibt sich, gemäß Gleichung 3.1, eine Reduktion der Dimensionalität der Eingangsdaten mit einer Kompressionsrate K von 93.75%.

$$K = 1 - (chb^{-1}cs^{-1}) \tag{3.1}$$

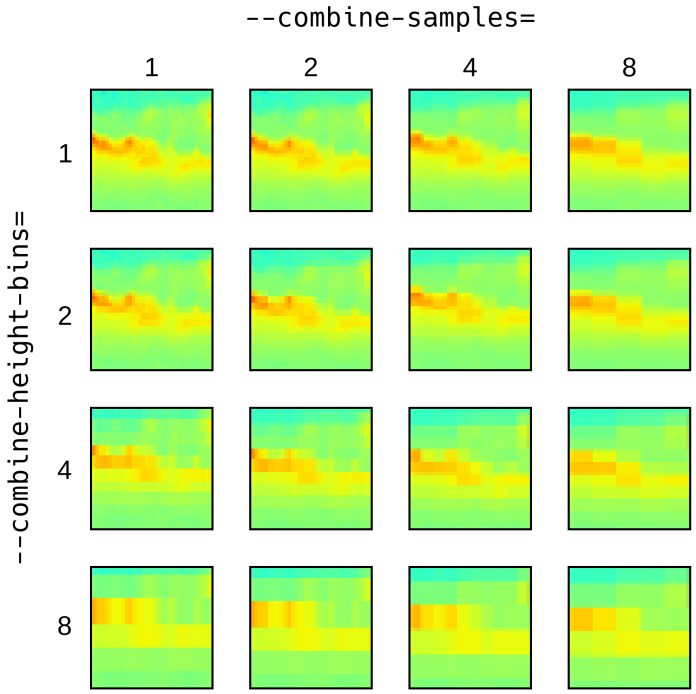
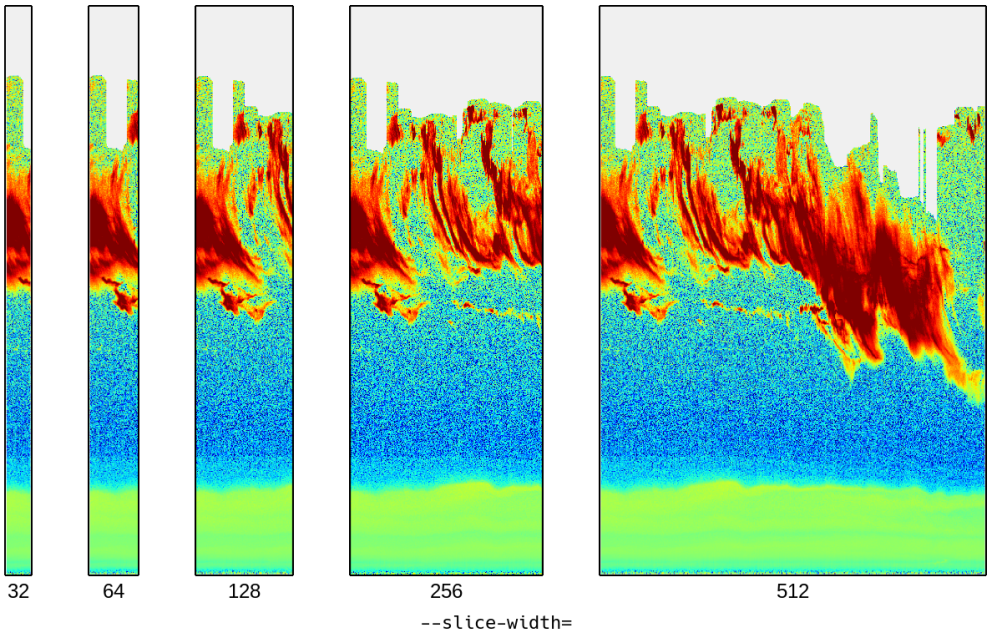


Abbildung 3.7: Vergleich verschiedener Komprimierungsintensitäten anhand weicher Strukturen

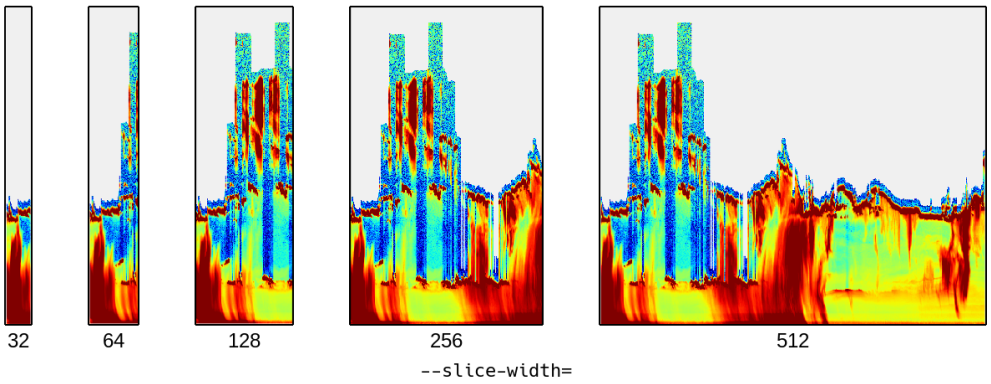
3.4 Aufteilbarkeit der Messdaten

Die gemessenen Aerosol-Rückstreuprofile liegen als kontinuierliche Zeitreihe von Einzelmessungen vor. Eine weitere untersuchte Eigenschaft ist, die mindestens je betrachtetem Ausschnitt benötigte Breite. Diese soll so gewählt werden, dass das beobachtete meteorologische Phänomen weiterhin identifiziert werden kann, die Breite der einzelnen Ausschnitte hierbei aber minimal bleibt. Dies ist für eine spätere Wahl der Eingangsdatengröße entscheidend.

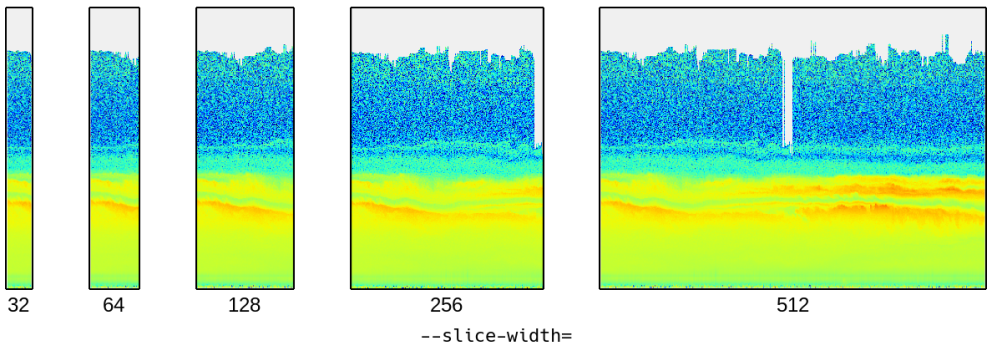
Betrachtet wurden Ausschnitte der Messdaten mit variierender Breite, wobei die Breite der Anzahl der Einzelmessungen je Ausschnitt entspricht. Analog zur Komprimierbarkeit (siehe Abschnitt 3.3) wurden die Auswirkungen unterschiedlicher Fensterbreiten (`--slice-width`) für mehrere Szenarien betrachtet. Abbildung 3.8 zeigt die resultierenden Messdatenschnitte für drei beispielhafte Rückstreuprofile, welche zusammen genommen alle der zu detektierenden Phänomene beinhalten.



(a) Rückstreuprofil vom 08. September 2018 aus Leipzig



(b) Rückstreuprofil vom 16. Mai 2018 aus Leipzig



(c) Rückstreuprofil vom 05. September 2018 aus Leipzig

Abbildung 3.8: Vergleich verschiedener Fensterbreiten für die Aufteilung von Messdaten

Es hat sich gezeigt, dass Fenstergrößen über 64 Messungen für Ausschnitte mit einer hohen Ereignisdichte ungeeignet sind, da hierdurch eine ungewollt hohe Anzahl verschiedener Phänomene innerhalb eines Fensters auftreten. Ferner lässt sich feststellen, dass alle Ereignisse, welche mit einer Fensterbreite von 64 Messungen erkennbar sind sich ebenfalls mit Ausschnitten der halben Größe identifizieren lassen. Daher wird im Folgenden, zugunsten der Größe der Eingangsdaten für das Verfahren, mit einer Fensterbreite von 32 Messungen pro Messdatenschnitt gearbeitet.

4 Verfahren

In diesem Kapitel wird das im Rahmen dieser Arbeit implementierte Verfahren zum unüberwachten Clustering mithilfe von Autoencodern beschrieben. Es gliedert sich in mehrere Teilschritte, welche im Folgenden detailliert beschrieben werden.

Das zugrundeliegende Konzept beruht auf dem Training eines Autoencoder-Netzes, welches lernt, die Messdaten in eine komprimierte Repräsentation zu überführen. Hat die gelernte Kodierung eine gewünschte Güte erreicht, so wird ein initiales Clustering der aus den Messdaten generierten Merkmalsvektoren durchgeführt. Im weiteren Trainingsverlauf wird dieses Clustering parallel zur Rekonstruktion der Messdaten optimiert.

Die Aufteilung des Verfahrens in ein vorangestelltes Training (Pre-Training) und insbesondere die darauf folgende Optimierung ist an das, in (van der Maaten, 2009) [6] vorgestellte Verfahren zur unüberwachten Dimensionsreduktion angelehnt. Dieses nutzt eingeschränkte Boltzmann-Maschinen (engl. Restricted Boltzmann Machines) (RBMs) [7] zur Abbildung der hochdimensionalen Eingangsdaten in einen komprimierten Merkmalsraum. Nach durchgeführtem Pre-Training dieser Netze wird das Ergebnis durch das beschriebene Verfahren weiter optimiert.

4.1 Überblick

Das vorgestellte Verfahren gliedert sich in fünf Teilschritte, welche jeweils in einem eigenen Abschnitt in der Tiefe erörtert werden. Um diese in den Gesamtkontext des Verfahrens einordnen zu können, zeigt Abbildung 4.1 die einzelnen Abschnitte sowie die benötigten Eingangsdaten inklusive der aus dem jeweiligen Arbeitsschritt resultierenden Artefakte. Die Aufteilung wurde so gewählt, dass Zwischenergebnisse aus einzelnen Abschnitten wiederverwendet werden können und somit Resultate für eine Evaluation des Verfahrens zu jedem Zeitpunkt zur Verfügung stehen.

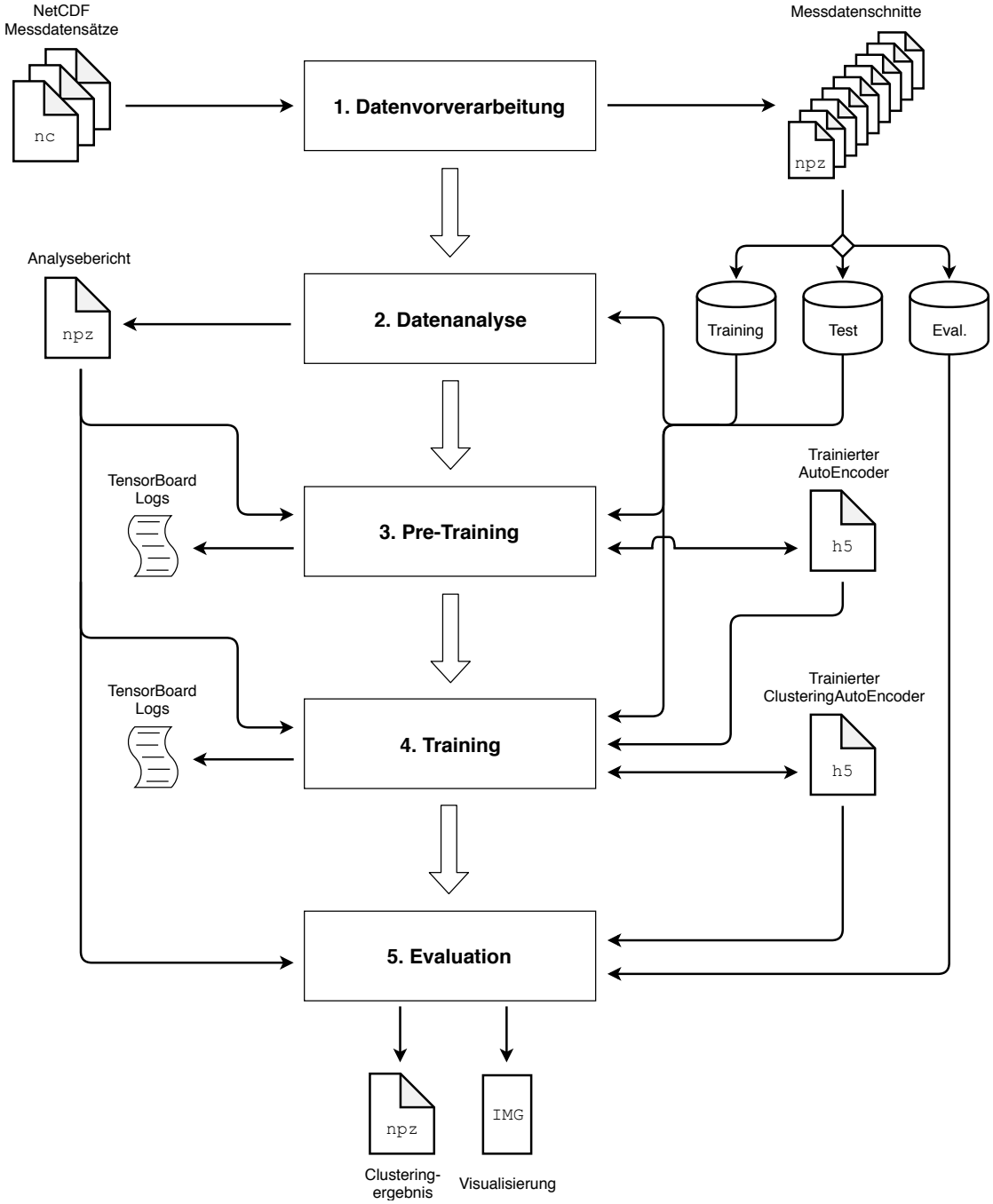


Abbildung 4.1: Teilschritte, Datenfluss sowie entstehende Artefakte des vorgestellten Verfahrens

Während der **Datenvorverarbeitung** (Abschnitt 4.2) werden die aus dem Messnetz des deutschen Wetterdienstes (DWD) stammenden NetCDF-Messdatensätze für die folgenden Schritte aufbereitet. Hierbei entstehende Messdatenschnitte werden in Trainings-, Test- sowie Evaluations-Daten aufgeteilt. Die darauf folgende **Datenanalyse** (Abschnitt 4.3) führt statistische Auswertungen durch, welche beispielsweise für die spätere Normalisierung der Datensätze benötigt werden. Hierbei wird ein maschinenlesbarer Analysebericht erzeugt. Im so genannten **Pre-Training** (Abschnitt 4.4) wird ein Autoencoder-Netz trainiert, die Messdatenschnitte in komprimierte Merkmalsvektoren zu überführen und aus diesen zu rekonstruieren. Das hierbei entstehende trainierte Netz wird im **Training** (Abschnitt 4.5) um Clustering-Methoden erweitert. Ferner erzeugen alle Schritte, in denen neuronale Netze trainiert werden, Logs, mit welchen der Trainingsverlauf einzelner Netze ausgewertet und verglichen werden kann. Liegt ein fertig trainierter Clustering-Autoencoder vor, so kann dieser genutzt werden, um ein Clustering von Messdatenschnitten im Rahmen der **Evaluation** (Abschnitt 4.6) durchzuführen. Hierbei entsteht eine maschinenlesbare Cluster-Zuweisung sowie optional mehrere Visualisierungen.

4.2 Datenvorverarbeitung

Ziel der Datenvorverarbeitung ist es, die Messdatensätze der Ceilometer, welche im NetCDF-Format vorliegen, in ein geeignetes Eingabeformat für das zu trainierende neuronale Netz zu transformieren. Im Rahmen dieser Arbeit werden hierzu serialisierte Numpy[1]-Arrays genutzt. Ferner sind die Rückstreuintensitäten β zu kalibrieren, Artefakte zu entfernen sowie gegebenenfalls eine simple Kompression der Messdaten durchzuführen.

4.2.1 Verwendete Datenstrukturen

Die aus den Ceilometern stammenden Datensätze besitzen, wie bereits im Abschnitt 2.4.2 beschrieben, eine Vielzahl an Variablen. Hiervon werden jedoch, wie in Tabelle 4.1 dargestellt, nicht alle benötigt.

[1] Siehe: https://www.numpy.org/ (Abgerufen am 02.04.2019)

Variable	Nutzungsgrund
`attn_bsct`	Hauptmessgröße
`time`	Bestimmung der Messzeitpunkte in `attn_bstc`
`range_gate`	Bestimmung der relativen Höhenbereiche in `attn_bstc`
`altitude`	Berechnung der absoluten Höhenbereiche in `range_gate`
`cbh`	Entfernen von Artefakten
`mxd`	Entfernen von Artefakten

Tabelle 4.1: Genutzte Variablen aus NetCDF-Messdatensätzen

4.2.2 Kalibrierung der Rückstreuprofile

Die Kalibrierung der Rückstreuintensitäten β erfolgt mithilfe der sogenannten Rayleigh-Methode [5]. Hierzu ist die LIDAR-Konstante C_L für den entsprechenden Messzeitpunkt aus den mitgelieferten Kalibrierdatensätzen zu entnehmen. Die Berechnung der kalibrierten Rückstreuung $\tilde{\beta}$ erfolgt dann gemäß Gleichung 4.1.

$$\tilde{\beta} = \beta \cdot {C_L}^{-1} \tag{4.1}$$

Die Messzeitpunkte `time` liegen als Zeitstempel in Sekunden seit dem 01.01.1904 (HFS+ Format) vor. Um Probleme mit genutzten Programmbibliotheken zu vermeiden, werden die Zeitstempel in Sekunden seit dem 01.01.1970 (UNIX Format) umgerechnet. Außerdem sind die relativen Höhenangaben `range_gate`, unter Berücksichtigung der Höhe des Ceilometers `altitude`, in absolute Höhen in m ü. NHN umzurechnen. Dies ermöglicht den problemlosen Vergleich der Messdaten mehrerer Ceilometer untereinander.

4.2.3 Entfernen unbrauchbarer Signalanteile

Durch die in Abschnitt 2.2.2 beschriebenen Eigenschaften des Messprinzips kommt es zur Bildung von Detektionsgrenzen im oberen Messbereich sowie zur Artefaktbildung in Ceilometernähe. Zur Verbesserung der Signalqualität können dadurch entstehende unbrauchbare Signalanteile vor der Weiterverarbeitung entfernt werden.

Um Artefakte, die aus dem Overlap-Problem entstehen, zu entfernen, wird eine untere Verwertbarkeitsgrenze t_{ovl} eingeführt. Alle Datenpunkte $\tilde{\beta}$ unterhalb dieser Grenze werden abgeschnitten.

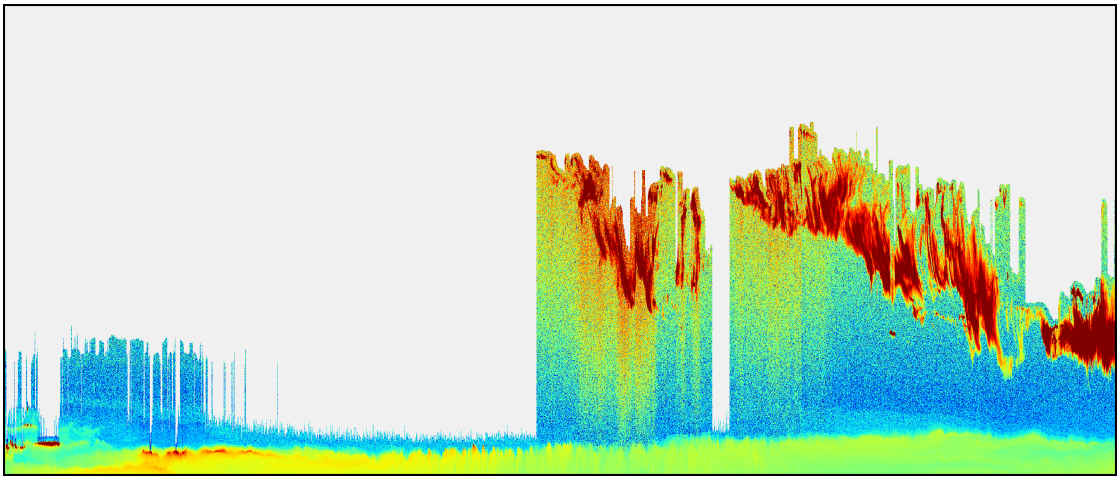
Datenpunkte, deren SNR zu gering für eine spätere Verwertung ist, sollten ebenfalls nicht in den bereinigten Datensätzen auftauchen. Hierzu bieten sich mehrere Grenzen an. Zum einen ist es möglich, oberhalb einer der bis zu drei detektierten Wolkenhöhen `cbh[0-2]` einen Schnitt zu setzen. Dies kann je nach betrachteten Phänomenen sinnvoll sein. Im allgemeinen Fall ist jedoch die Nutzung der maximalen Detektionshöhe `mxd` zu bevorzugen. Hierbei bestimmt das Ceilometer eigenständig die maximale Höhe, bis zu welcher die Rückstreuungen der jeweiligen Messung noch ein nutzbares SNR aufweisen. Messpunkte oberhalb der gewählten Grenze können analog zu den Artefakten entfernt werden.

4.2.4 Kompression der Messdaten

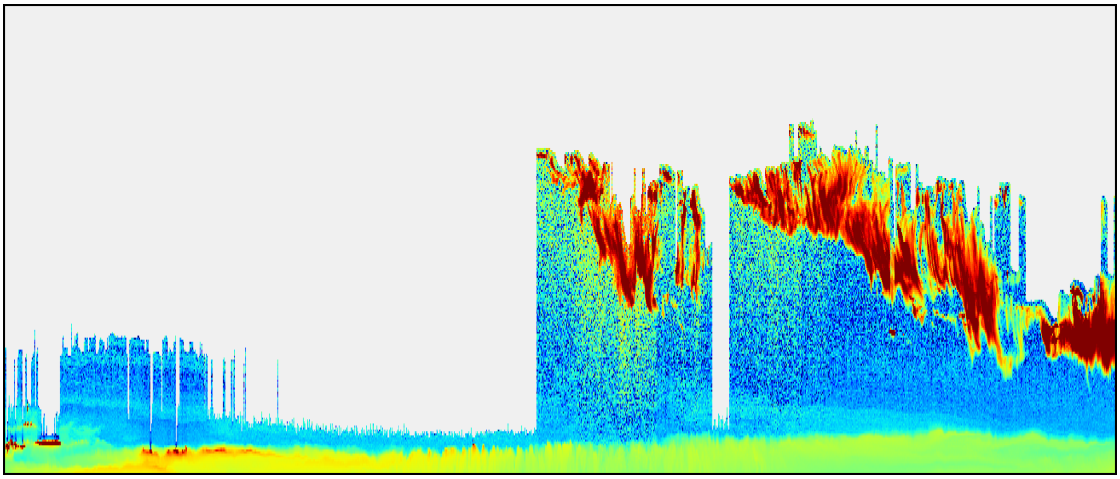
Zur Reduktion der Eingangsdatengröße des Autoencoders können die Messdaten komprimiert werden. Die in Abschnitt 3.3 beschriebene Analyse zur Komprimierbarkeit hat ergeben, dass eine Kombination von jeweils $chb = 4$ Höhenbereichen sowie $cs = 4$ Messungen bei den vorliegenden Messdaten möglich ist. Demnach ergibt sich die komprimierte Rückstreuintensität $\tilde{\beta}_K$ aus dem Mittelwert von jeweils $n = 16$ Datenpunkten gemäß Gleichung 4.2. Messpunkte, die als unbrauchbare Signalanteile identifiziert und daher entfernt wurden, gehen nicht, beziehungsweise nur mittelwertneutral, mit in die Berechnung von $\tilde{\beta}_K$ ein.

$$\tilde{\beta}_K = \frac{\sum_{i=0}^{chb} \sum_{j=0}^{cs} \tilde{\beta}_{i,j}}{n} \tag{4.2}$$

In Abbildung 4.2 ist der Vergleich eines Rückstreuprofils vor sowie nach der beschriebenen Komprimierung zu sehen. Es ist zu erkennen, dass die Strukturen für das Verfahren in ausreichender Güte erhalten bleiben. Die grauen Flächen entsprechen den gemäß Abschnitt 4.2.3 entfernten Signalanteilen.



(a) Unkomprimiertes Rückstreuprofil vom 08.09.2018 aus Leipzig



(b) Komprimiertes Rückstreuprofil vom 08.09.2018 aus Leipzig

Abbildung 4.2: Vergleich (un-)komprimierter Rückstreuprofile

4.2.5 Generierung von Messdatenschnitten

Ziel dieses Teilschritts der Datenvorverarbeitung ist die Überführung der nun kalibrierten, bereinigten sowie komprimierten Messdaten in ein für das zu trainierende Faltungsnetz (engl. *Convolutional Neural Network*) (CNN) geeignete Eingangsformat. Hierzu werden sogenannte Messdatenschnitte mithilfe eines Fensterverfahrens erzeugt. Es werden jeweils Ausschnitte voller Höhe mit einer beliebig aber fest gewählten Breite *sw* (`--slice-width`) aus den Messdatensätzen erzeugt. Nach jedem generierten Messdatenschnitt wird das Fenster um eine ebenfalls beliebige aber fest zu wählende Anzahl an

Messungen ss (`--slice-steps`) weiter geschoben. Dieser Vorgang wird wiederholt, bis das Ende des jeweiligen Messdatensatzes erreicht ist.

Die Wahl von $sw = 32$ erfolgt gemäß des Analyseergebnisses aus Abschnitt 3.4. Mit dem Parameter ss lässt sich sowohl die resultierende Anzahl der Messdatenschnitte als auch die Überlappung einzelner Fenster beeinflussen. Je geringer dieser Wert gewählt wird, desto mehr Messdatenschnitte werden erzeugt und desto höher ist der Überlappungsgrad der einzelnen Fenster. Im Rahmen dieser Arbeit wurde $ss = \frac{1}{2} \cdot sw$ gewählt. Dies begründet sich darin, dass einzelne Ereignisse somit nicht durch eine unvorteilhafte Wahl der Schnittkanten nur in solch kleinen Teilen vorhanden sind, als dass sie auf keinem der Messdatenschnitte mehr sauber identifiziert werden können. Ferner erhöht ein Verschub um die halbe Fensterbreite die Anzahl der zur Verfügung stehenden Trainings- sowie Testdatensätze. Somit ergeben sich, gemäß Gleichung 4.3, für jeden betrachteten Tag, mit $n_{samples} = 5760$ Messungen pro Tag, insgesamt 360 Messdatenschnitte.

$$n_{slices} = \lfloor n_{samples} \cdot ss^{-1} \rfloor \tag{4.3}$$

Die generierten Datensätze werden zum Schluss jeweils mit einem eigenen Universally Unique Identifier (UUID) der Version 4 versehen und in Form von serialisierten Numpy-Arrays gespeichert. Jede Datei enthält die in Tabelle 4.2 aufgeführten Variablen. Abschließend erfolgt eine Aufteilung aller erzeugten Messdatenschnitte in die Bereiche Trainings-, Test- sowie Evaluationsdaten.

Variable	**Beschreibung**
`attn_bsct`	Kalibrierte, bereinigte und komprimierte Rückstreuintensität
`times`	Messzeitpunkte als UNIX-Timestamp
`height_bins`	Absolute Höhenbereiche
`bin_height`	Höhe eines einzelnen Höhenbereichs
`max_cbh`	Höchste detektierte Wolkenuntergrenze je Messung
`mxd`	Maximale Detektionshöhe je Messung
`altitude`	Absolute Höhe des Ceilometers
`location`	Bezeichnung des Ceilometer-Standorts
`dataset_filename`	Dateiname des Ursprünglichen Messdatensatzes

Tabelle 4.2: Inhalt eines Messdatenschnitts

4.3 Datenanalyse

Ziel des Datenanalyse-Schritts ist die Bestimmung verschiedener statistischer Eigenschaften der generierten Messdatenschnitte. Die Ergebnisse werden als maschinenlesbarer Analysebericht für später folgende Schritte abgelegt. Ferner können automatisiert Diagramme zur Auswertung der Eigenschaften der Eingangsdaten erzeugt werden.

4.3.1 Normalisierungsparameter

Zur Optimierung des Trainings erfolgt eine Normalisierung der Eingangsdaten. Dies hat sowohl den Vorteil einer schnelleren Konvergenz des Lernalgorithmus [8], [9] als auch einer Abmilderung anderer Konvergenzprobleme. Die hierzu benötigten Parameter werden im Rahmen der Datenanalyse bestimmt.

Für die Normalisierung relevant sind je Datenpunkt beziehungsweise Pixel $\tilde{\beta}_{i,j}$, wobei i und j die Position des jeweiligen Pixels angeben, über alle n Messdatenschnitte folgende Werte: Maximum $\max_p$, Minimum $\min_p$, Mittelwert μ_p sowie die Standardabweichung σ_p. Diese werden gemäß Gleichung 4.4 berechnet. Zusätzlich wird die Varianz je Pixel σ_p^2 sowie das globale Maximum und Minimum bestimmt.

$$\max_p(i,j) = \max(\tilde{\beta}_{i,j}) \tag{4.4a}$$

$$\min_p(i,j) = \min(\tilde{\beta}_{i,j}) \tag{4.4b}$$

$$\mu_p(i,j) = \frac{1}{n}\sum_{k=1}^{n} \tilde{\beta}_{i,j}(k) \tag{4.4c}$$

$$\sigma_p(i,j) = \sqrt{\frac{\sum_{k=1}^{n}\left[\tilde{\beta}_{i,j}(k) - \mu_p(i,j)\right]^2}{n}} \tag{4.4d}$$

Nach erfolgreicher Berechnung aller Werte können diese als Diagramm ausgegeben werden. Somit kann beispielsweise der durchschnittliche Eingangsdatensatz μ_p betrachtet werden, um weitere Rückschlüsse auf die Eigenschaften der Daten zu ziehen.

4.3.2 Histogrammanalyse

Im Rahmen der Architektur- sowie Parameterwahl für das neuronale Netz kann es interessant sein, analog zu dem in Abschnitt 3.2 für gesamte Messzeiträume berechneten Histogramm, die Werteverteilung innerhalb der generierten Messdatenschnitte zu betrachten. Hierzu wird ebenfalls ein Histogramm berechnet und dem Analysebericht angehängt.

4.3.3 Generierung eines Analyseberichts

Um die Resultate der Datenanalyse für die folgenden Schritte verfügbar zu machen, wird ein maschinenlesbarer Analysebericht gespeichert. Dieser beinhaltet alle in Tabelle 4.3 aufgelisteten Werte und wird wie die Messdatenschnitte ebenfalls in Form von serialisierten Numpy-Arrays abgelegt.

Variable	**Beschreibung**
`max`	Globale maximale Rückstreuintensität
`min`	Globale minimale Rückstreuintensität
`pp_max`	Array der Größe (i, j) mit $\max_p$ je Datenpunkt
`pp_min`	Array der Größe (i, j) mit $\min_p$ je Datenpunkt
`pp_mean`	Array der Größe (i, j) mit μ_p je Datenpunkt
`pp_std`	Array der Größe (i, j) mit σ_p je Datenpunkt
`pp_var`	Array der Größe (i, j) mit σ_p^2 je Datenpunkt
`histogram`	Absolut gezählte Werte pro Wertebereich
`histogram_bins`	Wertebereiche der in `histogram` abgelegten Zähler

Tabelle 4.3: Inhalt eines Analyseberichts

4.4 Pre-Training

Im Rahmen des Pre-Trainings wird ein sogenannter Autoencoder trainiert, um die erzeugten Messdatenschnitte in eine komprimierte Repräsentation zu überführen. Der Fortschritt sowie Erfolg des Trainings wird hierbei zur späteren Auswertung maschinenlesbar

protokolliert. Als Artefakt dieses Teilschritts wird der trainierte Autoencoder in serialisierter Form abgelegt. Dieser kann dann entweder im Folgeschritt genutzt oder auch weiter im Rahmen dieses Schrittes als Zwischenergebnis geladen und weiter trainiert werden.

4.4.1 Autoencoder

Ein Autoencoder ist ein künstliches neuronales Netz welches lernt, den Eingangsdatenvektor x in eine komprimierte Repräsentation $\tilde{x}$ zu überführen [10]. Hierbei ist die Dimension von $\tilde{x}$ deutlich geringer als die der Eingangsdaten x. Es handelt sich also um ein Verfahren zur Dimensionsreduktion beziehungsweise Enkodierung. Erst durch diese Komprimierung lässt sich im späten Verlauf ein effektives Clustering durchführen. Ohne die vorherige Dimensionsreduktion würde hierbei der sogenannte *Fluch der Dimensionalität* [11] zum Problem werden.

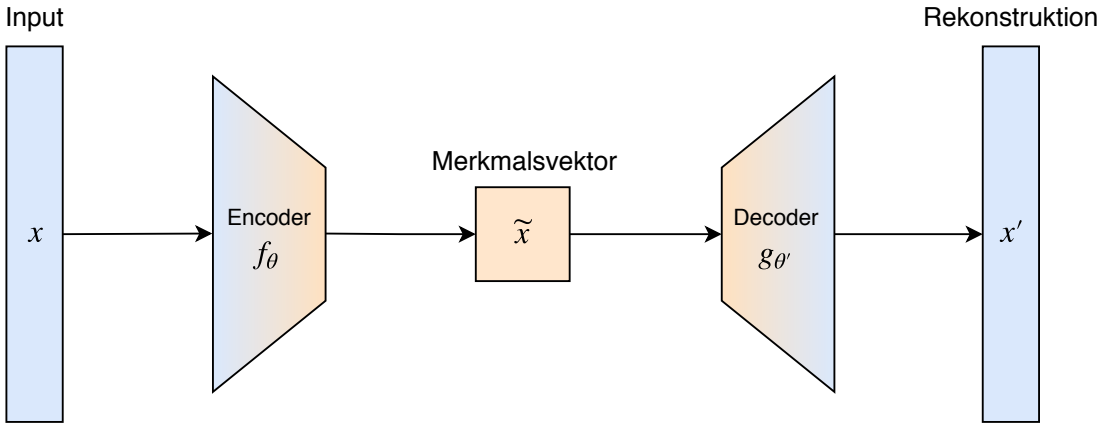


Abbildung 4.3: Schematischer Aufbau eines Autoencoders

Autoencoder bestehen im Allgemeinen aus einem Encoder f_θ und einem Decoder $g_{\theta'}$ wie in Abbildung 4.3 dargestellt. Hierbei überführt der Encoder die Eingangsdaten $x \in X$ in komprimierte Repräsentationen $\tilde{x} \in \tilde{X}$, welche dann vom Decoder zur Erzeugung möglichst guter Rekonstruktion $x' \in X'$ von x genutzt werden (siehe Gleichung 4.5). Sowohl f als auch g verfügen über unabhängige Mengen θ beziehungsweise θ' an Gewichten sowie Bias-Werten. Dieser Aufbau ermöglicht das unüberwachte Lernen einer Abbildung der Eingangswerte in den Merkmalsvektorraum. Hierzu wird als Verlustfunktion L_r der mean squared error (dt. *Mittlerer quadratischer Fehler*) (MSE) zwischen einem Eingangsdatum x und der Rekonstruktion x' minimiert. Der Vorteil gegenüber der klassischen

Hauptkomponentenanalyse (engl. *Principal Component Analysis*) (PCA) [12] liegt in der Nichtlinearität der gelernten Abbildungen.

$$f : X \longmapsto \tilde{X} \tag{4.5a}$$

$$g : \tilde{X} \longmapsto X' \tag{4.5b}$$

Es existieren verschiedene Formen von Autoencodern. Im Rahmen dieser Arbeit kommen Faltungsnetz (engl. *Convolutional Neural Network*) (CNN) basierte Autoencoder, auch Convolutional Autoencoder (CAE) genannt, zum Einsatz. Diese Wahl begründet sich in der konsequenten Überlegenheit von CNNs bei der Verarbeitung von bildähnlichen Daten [13] gegenüber klassischen Netzen ohne Faltungsebenen (engl. *convolutional layer*). Sie haben den Vorteil, dass Nachbarschaftsbeziehungen zwischen den einzelnen Messdatenpunkten erhalten bleiben und als zusätzliche Information einfließen. Somit wird bereits durch die zugrunde liegenden Operationen ein Verhältnis zwischen benachbarten Höhenschichten sowie einzelnen Messungen hergestellt. Außerdem lernen die genutzten Faltungsebenen gezielt die Extraktion der repräsentativsten Merkmale (engl. *features*), aus den vorhandenen Eingangsdaten [14]. Dies macht sie besonders geeignet für die Erzeugung komprimierter Merkmalsvektoren.

4.4.2 Normalisierung

Zur Skalierung sowie Normalisierung der während des Pre-Trainings eingelesenen Messdatenschnitte kann ein zuvor erzeugter Analysebericht (siehe Abschnitt 4.3.3) eingelesen werden. Dieser beinhaltet Parameter für zwei im Folgenden beschriebene Verfahren.

Die einfachste und bei der Nutzung von bildähnlichen Daten verbreitetste Variante ist die Skalierung der Eingangswerte auf das Intervall $[0, 1]$. Sie wird auch als Min/Max Skalierung bezeichnet und erfolgt gemäß Gleichung 4.6.

$$x_{norm} = \frac{x - \min(x)}{\max(x) - \min(x)} \tag{4.6}$$

Eine weitere Möglichkeit ist die Mittelwertbefreiung sowie Normierung der Messdatenschnitte [15]. Sie ist einer einfachen Skalierung insofern überlegen, als dass die Wertebereiche unterschiedlicher Dimensionen vereinheitlicht werden. Somit wird ausgeschlossen,

dass einzelne Dimensionen mehr Relevanz als andere erhalten. Normierte Eingangswerte ergeben sich bei dieser Variante gemäß Gleichung 4.7.

$$x_{norm} = \frac{x - \mu_p}{\sigma_p} \tag{4.7}$$

Da die Messdaten bereits einen einheitlichen Wertebereich aufweisen, wird im Rahmen dieser Arbeit auf die Min/Max Skalierung zurück gegriffen.

4.4.3 Überwachung des Lernfortschritts

Um die Funktionsfähigkeit sowie die Güte des trainierten Autoencoders beurteilen zu können, wird am Ende jeder Trainingsepoche ein Log mit Diagnosedaten geschrieben. Dieser beinhaltet den Lernverlauf des Netzes sowie optional weitere Einzelheiten, wie die Entwicklung der Gewichte oder auch Ausgaben einzelner Netzebenen zur späteren Datenanalyse. Die Erstellung erfolgt in einem zur Visualisierungssoftware TensorBoard[2] kompatiblen Format.

Außerdem ist es möglich, das aktuelle Netz inklusive aller Gewichte am Ende jeder Epoche in serialisierter Form abzuspeichern. Somit können einzelne Zwischenstände zu einem späteren Zeitpunkt geladen und genau evaluiert werden.

4.5 Training

Der im Pre-Training (Abschnitt 4.4) vorbereitete Autoencoder wird zu Beginn dieses Schrittes um einen sogenannten Clustering-Layer erweitert. Der so entstandene Clustering Autoencoder lernt, auf Basis der komprimierten Merkmalsvektoren eine Clusterzugehörigkeit zu dem jeweiligen Messdatenschnitt auszugeben. Nach anfänglicher Initialisierung wird das Clustering mit Verlauf des Trainings weiter optimiert. Analog zum vorherigen Schritt wird ebenfalls sowohl der Lernfortschritt protokolliert als auch das trainierte Netz gespeichert.

[2] Siehe: https://github.com/tensorflow/tensorboard (Abgerufen am 02.04.2019)

4.5.1 Clustering Autoencoder

Die Basis für einen Autoencoder mit Clustering-Funktionalität ist der bereits vortrainierte Autoencoder aus dem Pre-Training. Dieser wird, wie in Abbildung 4.4 dargestellt, um einen Clustering-Layer h_ϕ erweitert. Er besitzt eine eigene Menge an Gewichten ϕ und berechnet auf Basis der Merkmalsvektoren $\tilde{x} \in \tilde{X}$ die Clusterzugehörigkeiten $c(\tilde{x}) \in C(\tilde{X})$ [16], [17]. Somit ergibt sich eine Abbildung gemäß Gleichung 4.8.

$$h : \tilde{X} \longmapsto C(\tilde{X}) \tag{4.8}$$

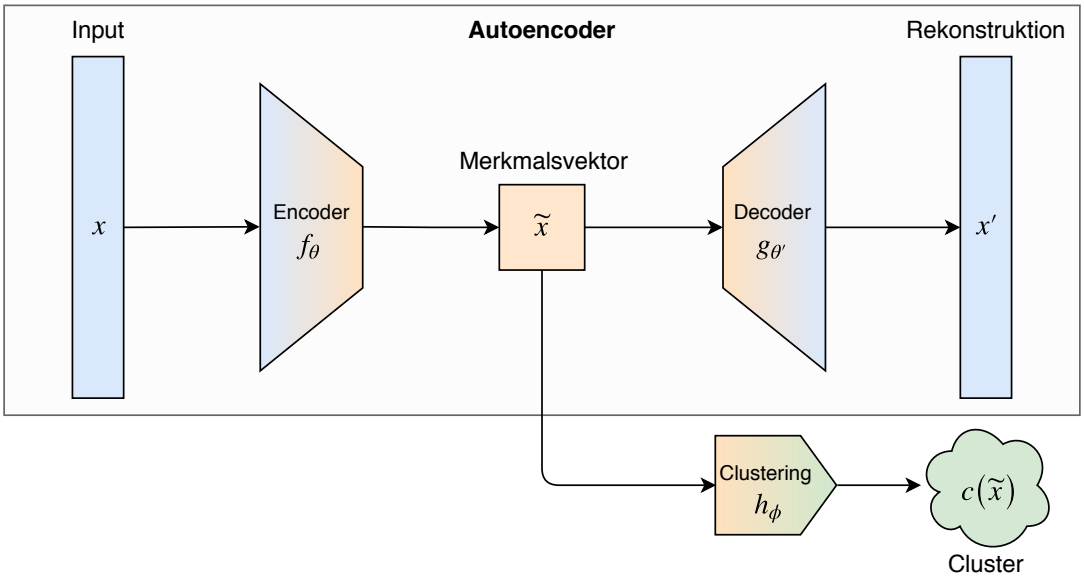


Abbildung 4.4: Schematischer Aufbau eines Clustering-Autoencoders

Die neu eingeführte Ebene besteht aus einem *fully connected Layer* mit k Neuronen, wobei k der Anzahl der Cluster entspricht, auf welche die Messdatenschnitte zu verteilen sind. Somit wird eine Entkopplung der Dimensionalität des Merkmalsvektors $\tilde{x}$ und der Anzahl der Cluster erreicht. Dies ermöglicht es, eine dem Problem angemessene Größe für $\tilde{x}$ zu wählen.

Die Clusterzugehörigkeit $c(\tilde{x})$ wird in Form einer Wahrscheinlichkeitsverteilung über die Zugehörigkeit eines Merkmalsvektors $\tilde{x}$ zu jedem der k Cluster angegeben (engl. *soft assignments*). Somit gilt also $\dim c(\tilde{x}) = k$ sowie $\sum c(\tilde{x}) = 1$. Auf die Berechnung der Clusterzugehörigkeit wird detailliert in Abschnitt 4.5.3 eingegangen.

4.5.2 Initialisierung

Vor dem ersten Training des beschriebenen Clustering-Layers wird eine Initialisierung seiner Gewichte ϕ vorgenommen. Diese entsprechen jeweils einem der k Clustermittelpunkte (engl. *centroids*) $\left\{m_i^{(t)}\right\}_{i=1\ldots k}$, die zu den Clustern S_i gehören. Sie werden als Ergebnis mehrerer Durchläufe von k-Means [18] beziehungsweise k-Means++ [19], über die gebildeten Merkmalsvektoren $\tilde{X}$ oder einer Untermenge dieser, initial gewählt. Hierbei wird jeder der n Durchläufe mit unterschiedlichen Startwerten durchgeführt. Nach Abschluss wird die Menge an Clustermittelpunkten $m_i^{(t)}$ gewählt, für welche die Summe der quadrierten Abweichungen von den Clustermittelpunkten E, gemäß Gleichung 4.9, minimal ist.

$$E = \sum_{i=1}^{k} \sum_{\tilde{x}_j \in S_i} \|\tilde{x}_j - m_i\|^2 \tag{4.9a}$$

$$\phi = \underset{\left\{m_i^{(t)}\right\}_n}{\arg\min} \; E(n) \tag{4.9b}$$

Die Wahl von k-Means als Clusteringverfahren begründet sich unter anderem in der Art, in der die einzelnen Cluster beschrieben werden. Eine Clusterzuordnung erfolgt basierend auf der Distanz des jeweiligen Datums $\tilde{x}$ zu den Clustermittelpunkten $m_i^{(t)}$ im Merkmalsraum $\tilde{X}$. Ein Vorteil dieser Darstellung ist, dass sie eine simple Aussage über die Zugehörigkeit eines Datums zu jedem einzelnen der Cluster zulässt (siehe Abschnitt 4.5.3). Außerdem erlaubt sie, dass für Datenpunkte, welche nicht zur Berechnung der Clustermittelpunkte genutzt wurden, trotzdem eine eindeutige Aussage bezüglich ihrer Clusterzugehörigkeit getroffen werden kann. Dies ist besonders nützlich, wenn das initiale Clustering nur auf Basis einer Untermenge der komprimierten Merkmalsvektoren durchgeführt werden kann[3]. Darüber hinaus ist das Ziel der Initialisierung nicht das Erreichen eines bestmöglichen finalen Clusterings, sondern vielmehr soll die initiale Clusterzuteilung lediglich als Basis für weitere Optimierungen dienen (siehe Abschnitt 4.5.4). Ferner besitzt k-Means alleinig die Clusteranzahl k als Parameter. Dies erleichtert die Hyperparametersuche im Vergleich zu komplexeren Verfahren, wie beispielsweise dem *Density-Based Spatial Clustering of Applications with Noise* (DBSCAN) [20].

[3] Da es vorkommen kann, dass die Gesamtheit aller Trainingsdaten nicht zur selben Zeit in den Hauptspeicher geladen werden kann, wird die Möglichkeit des Clusterings auf Basis von Untermengen benötigt.

Nach erfolgreicher Initialisierung der Gewichte kann der Clustering-Layer bereits zum unüberwachten Clustering von Messdatenschnitten genutzt werden. Es bietet sich jedoch an, das Clusteringergebnis im weiteren Trainingsverlauf zu optimieren. Das hierzu nötige Vorgehen wird in Abschnitt 4.5.4 detailliert beschrieben.

4.5.3 Soft Clustering

Wird einem Datensatz durch ein Clusteringverfahren nicht ein explizites Cluster zugewiesen, sondern eine Zugehörigkeitswahrscheinlichkeit zu jedem der möglichen Cluster (engl. *soft assignment*) angegeben, so spricht man von einem sogenannten *soft clustering*. Die Ausgabe des eingeführten Clustering-Layers entspricht einem solchen *soft assignment*. Hierbei wird zu einem komprimierten Merkmalsvektor $\tilde{x}$ die Wahrscheinlichkeit seiner Zugehörigkeit $c(\tilde{x})$ zu jedem der k Cluster berechnet. Somit ergibt sich die Ausgabedimension $\dim c(\tilde{x}) = k$ des Layers. Ferner gilt für die Summe der Wahrscheinlichkeiten: $\sum c(\tilde{x}) = 1$.

Die Zugehörigkeit zu einem Cluster S_i wird als Ähnlichkeit q_{ij} zwischen einem Eingangsvektor $\tilde{x}_j$ und dem jeweiligen Clustermittelpunkt $m_i^{(t)}$ definiert. Hierzu wird das in *parametric t-SNE* [6] eingeführte Ähnlichkeitsmaß gemäß Gleichung 4.10 genutzt.

$$q_{ij} = \frac{(1 + \|\tilde{x}_j - m_i\|^2 / \alpha)^{-\frac{\alpha+1}{2}}}{\sum_{i'} (1 + \|\tilde{x}_j - m_i\|^2 / \alpha)^{-\frac{\alpha+1}{2}}} \tag{4.10}$$

Dieses Maß gibt die gewünschte Auskunft bezüglich der Ähnlichkeit zweier Punkte i sowie j und basiert auf der Studentschen t-Verteilung [21]. Hierbei entspricht α der Anzahl der Freiheitsgrade der zugrunde liegenden t-Verteilung. Da das zusätzliche Lernen dieses Parameters in einem unüberwachten Szenario gegenstandslos ist [16], gilt folgend $\alpha = 1$. Im Vergleich zur Nutzung einer Gauß-Verteilung umgeht die eingesetzte t-Verteilung das sogenannte *crowding problem* [22] dadurch, dass es sich hierbei um eine endlastige Verteilung (engl. *heavy-tailed distribution*) handelt [6]. Gemäß (van der Maaten, 2009) [6] ist der Einsatz einer endlastigen Verteilung stets vorzuziehen.

Letztlich erhalten wir, gemäß Gleichung 4.11, für einen Merkmalsvektor $\tilde{x}_j \in \tilde{X}$ genau k Wahrscheinlichkeiten, welche den normierten Ähnlichkeitsmaßen q_{ij} für $i = 1 \dots k$ entsprechen.

$$c(\tilde{x}) = \frac{1}{\sum q_{ij}(\tilde{x})} \cdot q_{ij}(\tilde{x}) \tag{4.11}$$

4.5.4 Optimierung des Clusterings

Die in Abschnitt 4.5.2 berechneten initialen Gewichte können bereits für ein Clustering genutzt werden. Das Clusteringergebnis unterscheidet sich hierbei jedoch in seiner Güte noch nicht von der trivialen Anwendung eines klassischen Verfahrens. Zur weiteren Optimierung wird eine neue Zielverteilung P definiert. Als Maß der Unterschiedlichkeit der Zielverteilung P vom aktuellen Clustering Q wird die Kullback-Leibler-Divergenz (KL-Divergenz) [23], gegeben durch Gleichung 4.12, genutzt. Sie ist somit die Verlustfunktion L_c des Clusterings.

$$L_c = KL(P||Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}} \tag{4.12}$$

Die Wahl von P kann frei nach den zu optimieren gewünschten Eigenschaften erfolgen. Im Rahmen dieser Arbeit wird $p_{ij} \in P$, in Anlehnung an [16], gemäß Gleichung 4.13 definiert. Ziel dieser Verteilung ist die Erhöhung der Klassenreinheit sowie eine höhere Gewichtung von Datenpunkten, deren Distanz zum Clustermittelpunkt gering ist. Ferner wird der Beitrag eines jeden Clusters zum Trainingsverlust normiert. Hierdurch wird verhindert, dass große Cluster dominieren.

$$p_{ij} = \frac{{q_{ij}}^2 / \sum_j q_{ij}}{\sum_i ({q_{ij}}^2 / \sum_j q_{ij})} \tag{4.13}$$

Festzuhalten ist außerdem, dass P einerseits als Lernziel definiert ist, jedoch ebenfalls vom aktuellen Clustering Q abhängt. Hierdurch können sich Instabilitäten in der Konvergenz ergeben. Um dies zu verhindern, sollte die Zielverteilung P nicht nach jeder Mini-Batch neu berechnet werden, sondern über mehrere Batches hinüber konstant bleiben. Im Rahmen dieser Arbeit wird die Aktualisierung der Zielverteilung am Ende jeder Trainingsepoche durchgeführt.

Um die Optimierung des Clusteringergebnisses zeitgleich zum weiteren Training des Autoencoders zu ermöglichen, wird eine gewichtete Kombination aus den beiden Verlustfunktionen L_c sowie L_r als neue Verlustfunktion L des gesamten Netzes gemäß Glei-

chung 4.14 definiert. L_r entspricht hierbei dem Rekonstruktionsverlust des Autoencoders und λ der Gewichtung von L_c.

$$L = L_r + \lambda L_c \tag{4.14}$$

Im Rahmen des Trainings hat sich die empirische Wahl von $\lambda = 0.1$ als geeignet herausgestellt. Dies hat zur Folge, dass die Güte der Merkmalsvektoren $\tilde{X}$ nicht zulasten der Minimierung von L_c negativ beeinflusst wird. Dennoch erlaubt der Anteil von L_c eine Näherung an die gewünschte Zielverteilung P und sorgt damit für eine Optimierung des Clusteringergebnisses.

4.6 Evaluation

Ist ein trainierter Clustering-Autoencoder vorhanden, kann dieser zur Auswertung von Messdaten genutzt werden. Dies können zum einen Messdatenschnitte aus dem Evaluations-Datensatz, aber auch gänzlich neue Messdaten sein. Letztere müssen im Rahmen der in Abschnitt 4.2 beschriebenen Datenvorverarbeitung zuerst kalibriert und in Messdatenschnitte überführt werden. Für jeden der Eingangsdatensätze x wird sowohl die Clusterzuordnung $c(\tilde{x})$ als auch die Rekonstruktion der Eingangsdaten x' berechnet. Die weitere Verarbeitung dieser Daten wird im Folgenden beschrieben.

4.6.1 Clusterzuordnung

Für jeden zu evaluierenden Messdatenschnitt x wird die Wahrscheinlichkeitsverteilung über die Clusterzugehörigkeit $c(\tilde{x})$ berechnet. Diese wird in maschinenlesbarer Form als serialisiertes Numpy-Array gemäß Tabelle 4.4 abgelegt.

Variable	**Beschreibung**
-	Array der Form (`n_cluster`,) mit der Clusterzugehörigkeitswahrscheinlichkeit $c(\tilde{x})$ je Cluster S_i

Tabelle 4.4: Inhalt einer serialisierten Clusterzuordnung

Das vorgenommene *soft assignment* kann problemlos in ein sogenanntes *hard assignment* umgewandelt werden. Hierbei wird keine Zugehörigkeitswahrscheinlichkeit zu jedem der

Cluster angegeben, sonder der Messdatenschnitt wird einem einzelnen Cluster S_i der insgesamt k Cluster zugewiesen. Gewählt wird das Cluster mit der höchsten Wahrscheinlichkeit gemäß Gleichung 4.15.

$$S(c(\tilde{x})) = \arg\max\, c(\tilde{x}) \tag{4.15}$$

4.6.2 Rekonstruktion der Eingangsdaten

Gerade im Rahmen der Analyse des trainierten neuronalen Netzes ist es häufig gewünscht, zusätzlich zur Clusterzuordnung eine Auswertung der rekonstruierten Eingangsdaten zu erhalten. Hierzu können optional Diagramme der Rückstreuprofile sowohl der Eingangsdaten x als auch der vom Autoencoder produzierten Rekonstruktion x' generiert werden.

4.6.3 Auswertung getaggter Datensätze

Sind verschlagwortete Datensätze vorhanden, so können diese genutzt werden, um die Genauigkeit der Clusterzuordnung zu bestimmen. Hierzu sind die in Abschnitt 4.2 beschriebenen Messdatenschnitte um eine Variable `truth` zu erweitern. Diese beinhaltet das dem Datensatz zugeordnete Label.

Da das neuronale Netz weiterhin mithilfe von unüberwachtem Lernen trainiert wird, entspricht die gelernte Clusterzuordnung $c(\tilde{x})$ nicht zwingend den spezifizierten Labels. Es ist folglich eine Abbildung jedes der gefundenen Cluster S_i auf jeweils eins der Labels $y \in Y$ zu definieren. Hierbei wird einem Cluster S_i jenes Label zugewiesen, welches am häufigsten unter den Cluster-Elementen $\tilde{x}_i \in S_i$ vorkommt (siehe Gleichung 4.16). Für den Fall, dass mehrere Cluster als dominierende Menge Datensätze des selben Labels aufweisen, ist diese Zuordnung nicht mehr eindeutig. In solch einem Fall wird das entsprechende Label dem Cluster mit der größten Anzahl von Treffern zugewiesen.

$$y_i = \arg\max \sum_{y \in Y} [\text{label}(\tilde{x}_i) == y] \tag{4.16}$$

Mit den aus dieser Abbildung gewonnenen Daten kann die Erkennungsrate des trainierten Netzes angegeben werden. Diese entspricht dem prozentualen Anteil korrekt erkannter Datensätze innerhalb jeder Klasse. Zur besseren Auswertung kann optional eine Wahrheitsmatrix erzeugt und ausgegeben werden. Diese trägt sowohl den Anteil der korrekt zugeordneten Daten als auch die Verteilung der jeweils in einer Klasse vorhandenen Fehler auf.

5 Implementation

Das in Kapitel 4 vorgestellte Verfahren wurde im Rahmen dieser Arbeit als Python Programm implementiert. Hierzu wurde unter anderem das Keras[1] Framework mit TensorFlow[2]-Backend genutzt. Die entwickelte Softwarelösung stellt einen Testrahmen zur Modellierung, zum Training sowie zur Evaluation der neuronalen Netze dar. Ferner ist sie durch den modularen Aufbau der Teilschritte ebenfalls für eine laufende Evaluation von Messdatensätzen im Produktivbetrieb geeignet. Innerhalb dieses Kapitels wird auf den Aufbau sowie die technischen Details der entwickelten Softwarelösung eingegangen. Zusätzlich zu dieser Implementationsbeschreibung finden sich ausführliche pydoc-Kommentare an allen Schnittstellen und Konfigurationsvariablen des Programms.

5.1 Technologien und Werkzeuge

Das Umsetzung des vorgestellten Verfahrens erfolgte in der Programmiersprache Python der Version 3.6. Hierbei handelt es sich um eine interpretierte multiparadigmatische Sprache, welche dynamisch typisiert ist. Sie ist einerseits im Bereich des maschinellen Lernens weit verbreitet und findet andererseits bereits Anwendung beim DWD.

Für die Modellierung und das Training der neuronalen Netze kommt Keras als Schnittstelle zum bekannten Machine-Learning Framework TensorFlow zum Einsatz. Keras bietet eine einheitliche, vom unterliegenden Backend weitgehend unabhängige Schnittstelle zu verschiedenen Frameworks, darunter TensorFlow, CNTK sowie Theano. Eng mit der API von Keras verzahnt arbeitet Numpy als mathematische Bibliothek zur Handhabung von Vektoren, Matrizen sowie hochdimensionalen Arrays. Hierbei bietet diese Bibliothek nicht nur beschriebene Datentypen inklusive eines umfangreichen Indizierungskonzepts, sondern darüber hinaus auch effiziente Funktionen zur Durchführung numerischer Berechnung mit diesen.

[1] Siehe: https://keras.io/ (Abgerufen am 05.04.2019)
[2] Siehe: https://www.tensorflow.org/ (Abgerufen am 05.04.2019)

Das Einlesen der Messdatensätze im NetCDF Format erfolgt mithilfe des netcdf4[3] Moduls. Es bietet eine Schnittstelle zu nativen HDF5 sowie NetCDF-4 C-Bibliotheken. Zusätzlich kommt Pandas[4] zum Einlesen der Kalibrierdatensätze zum Einsatz. Die Erstellung von Diagrammen wird durch die Nutzung von matplotlib[5] unterstützt.

Eine vollständige Liste der benötigten Programmbibliotheken sowie die genauen Versionsnummern dieser findet sich im Abschnitt A.1. Ferner wird eine maschinenlesbare Abhängigkeitsdefinition, welche direkt vom Python-Paketmanager `pip` eingelesen werden kann, mitgeliefert. Hierbei ist die Verwendung eines *virtual environments* (venv) zu empfehlen. So können die genutzten Bibliotheken in der korrekten Version installiert werden, ohne dabei durch andere global installierte Pakete gestört zu werden beziehungsweise ohne diese zu stören.

5.2 Nutzerschnittstelle

Die Bedienung des Programms erfolgt über die Kommandozeile (engl. *command-line interface*) (CLI). Dies erlaubt den einfachen automatisierten Einsatz in einem Produktivumfeld. Die einzelnen, in Abschnitt 4.1 beschriebenen Teilschritte können über den ersten positionellen Parameter ausgewählt werden. Jeder Teilschritt verfügt über eine individuelle Schnittstellendefinition. Diese ist jeweils in den umfangreichen Hilfetexten dokumentiert.

Ausgaben über den aktuellen Stand des Programms beziehungsweise der laufenden Operation erfolgen auf der Konsole. Zur selektiven Ausgabe dieser sind mehrere Log-Level implementiert. Alle entstehenden Artefakte werden in dem jeweils spezifizierten Ordner abgelegt. Ferner ist es möglich, das Programm zu jedem Zeitpunkt anzuhalten, mithilfe einer interaktiven Python-Konsole auf Zwischenergebnisse zuzugreifen und diese gegebenenfalls zu exportieren.

[3] Siehe: https://github.com/Unidata/netcdf4-python (Abgerufen am 05.04.2019)
[4] Siehe https://pandas.pydata.org/ (Abgerufen am 05.04.2019)
[5] Siehe: https://matplotlib.org/ (Abgerufen am 05.04.2019)

5.3 Programmmodule und Packages

Zur Strukturierung der unterschiedlichen Programmteile gliedert sich die Implementation neben allgemeinen Codemodulen in mehrere Packages. Diese sind in Abbildung 5.1 grafisch dargestellt und entsprechen jeweils den in Abschnitt 4.1 eingeführten Teilschritten. Die einzige Ausnahme bildet hierbei das `train`-Package. Dieses umfasst sowohl das Pre-Training des Autoencoders als auch das Training des erweiterten Clustering-Netzes. In den folgenden Abschnitten wird jeweils auf die Implementationsdetails der einzelnen Arbeitsschritte eingegangen.

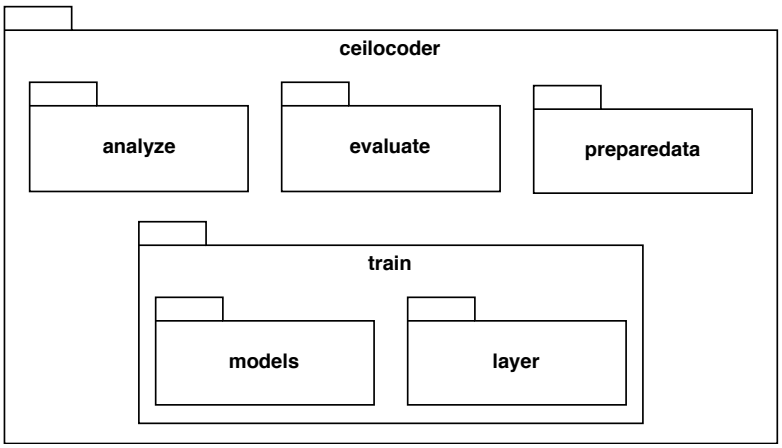


Abbildung 5.1: Übersicht aller Packages der Implementation

5.3.1 Allgemeines

Zusätzlich zu den beschriebenen Paketen existieren allgemeine Codeteile. Diese umfassen die Hauptroutine, ein Konfigurationsmodul, die Nutzerschnittstelle, ein Modul zur Erzeugung von Diagrammen sowie einige Hilfsfunktionen.

Der Einstiegspunkt des entwickelten Programms ist in der Datei `ceilocoder.py` definiert. An dieser Stelle wird lediglich der genutzte Logger konfiguriert und die für den ausgewählten Teilschritt zuständige Routine gestartet. Feste Konfigurationsparameter, also solche, die nicht dynamisch über die Nutzerschnittstelle gesetzt werden, befinden sich in der Datei `config.py`. Dies umfasst beispielsweise die genutzte Floatingpoint-Genauigkeit sowie die eingesetzten Datenformate. Die `Config`-Klasse wird zur Laufzeit

dynamisch um Parameter aus dem CLI erweitert, welches in der Datei `cli.py` definiert ist. Als Parser kommt die Python eigenene Bibliothek argparse[6] zum Einsatz.

Funktionen zur Erstellung von Diagrammen finden sich innerhalb des `plot` Moduls. Diese erlauben unter anderem das Darstellen von Rückstreuprofilen sowie Messdatenschnitten. Ferner werden die Erstellung von Wahrheitsmatrizen, Klassenreinheitsdiagrammen oder die Darstellung von rohen Bilddaten unterstützt. Weitere Hilfsfunktionen, wie beispielsweise die Konvertierung zwischen unterschiedlichen Datenformaten, sind in der Datei `util.py` definiert.

5.3.2 Datenvorverarbeitung (`preparedata`)

Das Paket `preparedate` beinhaltet alle für die Datenvorverarbeitung relevanten Funktionen. Der Zugriff auf die NetCDF-Messdatensätze sowie die dazugehörigen Kalibrierdaten wird mithilfe der Bibliotheken netcdf4 und pandas realisiert. Zur internen Handhabung wurde die `CeilometerDataset`-Klasse eingeführt. Sie kapselt eine Sammlung von Ceilometer-Messungen inklusive der zugehörigen Metadaten. Die im Folgenden beschriebenen Schritte werden unter der Nutzung eines Thread Pools parallel auf allen verfügbaren Prozessorkernen durchgeführt.

5.3.2.1 Kalibrierung und Filterung

Das Einlesen und Anwenden der Kalibrierparameter C_L sowie die optionale Signalfilterung sind im `preprocessing` Modul definiert. Ergebnis dieses Arbeitsschrittes ist ein kalibriertes sowie gefiltertes `CeilometerDataset`.

Da neue Werte für C_L nicht zwingend in gleichen Zeitabständen verfügbar sind, muss für jede Messung der passendste Wert identifiziert werden. Hierzu wird eine geordnete Liste der Kalibrierdaten-Zeitstempel erzeugt. Für einen Messdatensatz wird dann, mithilfe einer Binärsuche innerhalb dieser Liste, der zeitlich naheliegendste Parameter für die Kalibrierung der Rückstreuintensitäten β genutzt. Zusätzlich erfolgt die Konvertierung der Zeitstempel sowie die Berechnung der absoluten Höhenbereiche der nun kalibrierten Rückstreuintensitäten $\tilde{\beta}$.

[6] Siehe: https://docs.python.org/3/library/argparse.html (Abgerufen am 08.04.2019)

Ist eine Signalfilterung der Messdaten, wie in Abschnitt 4.2.3 beschrieben gewünscht, so wird diese ebenfalls in diesem Programmteil durchgeführt. Hierzu wird gemäß der gewünschten Filtermethode eine Maske erzeugt, oberhalb beziehungsweise unterhalb welcher die zu entfernenden Signalanteile liegen. Messpunkte, welche von dieser Filtermaske betroffen sind, werden durch `np.NaN` ersetzt.

5.3.2.2 Messdatenkomprimierung

Zur Reduktion der Eingangsdaten-Dimensionalität für Folgeschritte kann gemäß Abschnitt 4.2.4 eine Kompression der Rückstreuprofile erfolgen. Dies ist ebenfalls innerhalb der Datei `preprocessing.py` umgesetzt.

Die optionale Komprimierung erfolgt durch Bildung des Mittelwerts über mehrere Höhenbereiche und/oder Messungen. Hierzu werden die Dimensionen der Numpy-Arrays so angepasst, dass jeweils die zusammenzufassenden Messwerte auf einer neuen Achse liegen. Somit kann eine Mittelwertbildung einfach durch Reduktion der neu geschaffenen Achse durchgeführt werden. Die Anzahl der jeweils zusammenzufassenden Messpunkte ist über die Parameter `--combine-height-bins` beziehungsweise `--combine-samples` wählbar. Für den Fall, dass zur Mittelwertbildung ein Padding benötigt wird, werden der jeweiligen Achse mittelwertneurale Datenpunkte angehängt. Anschließend sind ebenfalls die Metadaten (Höhenbereiche, Zeitstempel, ...) an die neuen Wertebereiche anzupassen.

5.3.2.3 Slicing

Um die Messdatensätze in ein für den Autoencoder passendes Format zu überführen, werden diese, wie in Abschnitt 4.2.5 beschrieben, in mehrere Messdatenschnitte aufgeteilt. Dies geschieht innerhalb der Datei `slicing.py`. Hierbei können über die Kommandozeilenparameter `--slice-width` sowie `--slice-steps` die Breite sowie der zueinander relative Verschub der resultierenden Datensegmente spezifiziert werden.

Die generierten Daten werden in Form von serialisierten Numpy-Arrays im gewünschten Zielverzeichnis abgelegt und beinhalten jeweils alle in Tabelle 4.2 aufgelisteten Variablen. Der Dateiname eines Messdatenschnitts wird zufällig generiert und entspricht einer UUID (Version 4). Somit können Ausgaben eines Netzes sowie weitere Ergebnisse eindeutig den jeweiligen Eingangsdaten zugeordnet werden.

5.3.3 Datenanalyse (`analyze`)

Das `analyze` Package kapselt alle Vorgänge, die im Rahmen der Datenanalyse stattfinden. Hierbei übernimmt das Modul `analyze.py` die Validierung der Konfigurationsparameter und initiiert die Analyse inklusive der anschließenden Ergebnis-Visualisierung.

Die eigentliche Analyse der Messdatenschnitte erfolgt in der `SliceAnalysis` Klasse. Sie berechnet alle in Abschnitt 4.3 beschriebenen Parameter. Zu beachten ist, dass bei der Analyse großer Datenmengen Genauigkeitsverluste aufgrund von Rundungsfehlern der zugrunde liegenden Fließkomma-Arithmetik auftreten können. Ferner ist nicht garantiert, dass die Gesamtheit aller Datensätze gleichzeitig in den Hauptspeicher geladen werden kann. Daher findet die Verarbeitung schrittweise statt. Das Einlesen und die Bearbeitung der Messdaten erfolgt einzeln. Zwischenergebnisse mehrerer Messdatensätze werden in einer Menge von Arrays fester Größe[7] abgelegt. Die so gebildeten temporären Zwischengrößen werden am Ende der Analyse zusammengeführt, um die Analyseparameter zu bestimmen. Diese Vorgehensweise erlaubt es, die Analyse ohne Genauigkeitsverluste auch auf eine große Anzahl an Messdatenschnitten zu skalieren.

5.3.4 (Pre-) Training (`train`)

Alle Module, die im Rahmen des Trainings neuronaler Netze Anwendung finden, liegen im `train`-Paket. Es wird also sowohl für das Pre-Training eines Autoencoders als auch für das Training des Clustering-Autoencoders genutzt. Das Package gliedert sich, wie in Abbildung 5.2 dargestellt, in mehrere Module sowie zwei Unterpakete. Hierbei beinhaltet `models` alle entwickelten Netz-Architekturen und `layer` die zusätzlich zu den in Keras bereits vorhandenen, in dieser Arbeit eingeführten Layer. Im folgenden Abschnitt wird auf die Implementationsdetails der beschriebenen Punkte eingegangen.

5.3.4.1 Laden der Messdatenschnitte

Um die generierten Messdatenschnitte als Eingabe für das neuronale Netz nutzen zu können, müssen diese von der Festplatte geladen werden. Da das zeitgleiche Laden der gesamten Messdaten einen sehr hohen Speicherbedarf haben kann, wird stattdessen ein `SliceDataGenerator` genutzt. Dieser beerbt die Klasse `keras.utils.Sequence`[8] und

[7] Einstellbar über die Konfigurationsvariable `ANALYSIS_SUM_BIN_SIZE`
[8] Siehe: https://keras.io/utils/#sequence (Abgerufen am 10.04.2019)

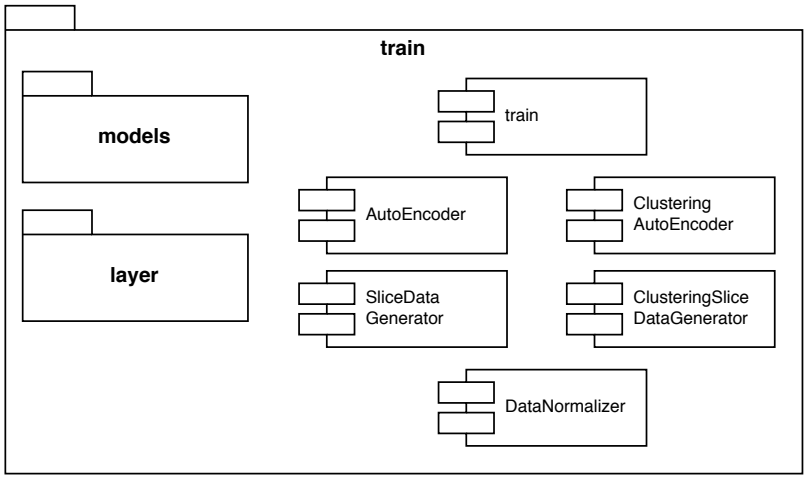


Abbildung 5.2: Modulübersicht des `train` Packages

stellt somit eine Möglichkeit dar, Daten-Batches während des Trainings dynamisch nachzuladen. Er wird innerhalb eines eigenen Threads parallel zum Training des Netzes ausgeführt und stellt stets eine feste Anzahl von Batches im Voraus zur Verfügung. Dies hat zur Folge, dass am Ende einer Batch nicht erst auf das, durch IO-Operationen langwierige Laden der nächsten Datensätze gewartet werden muss. Diese Entkopplung ist besonders bei der Ausführung auf einer GPU von Nutzen, da die Batch-Generierung parallel zum GPU-basierten Training auf der CPU stattfinden kann. Der `SliceDataGenerator` ist unabhängig von der Art der Messdatenschnitte implementiert und kann somit gleichermaßen für Trainings-, Test- sowie Evaluationsdaten genutzt werden.

Ferner können an dieser Stelle noch Modifikationen an den geladenen Messdaten vorgenommen werden. Hierzu kann bei der Initialisierung des Generators ein `DataNormalizer` übergeben werden. Dieser übernimmt die Normalisierung der Rückstreuprofile gemäß der Parameter aus einem Analysebericht und der gewünschten Normalisierungsform. Ebenso wird eine Denormalisierung für die vom Autoencoder ausgegebene Rekonstruktion unterstützt.

5.3.4.2 Autoencoder

Das Pre-Training eines Autoencoder-Netzes wird von der abstrakten `AutoEncoder`-Klasse gekapselt. Sie bietet einen allgemeinen Rahmen zum Training verschiedener Autoencoder-Architekturen. Dieser umfasst die Initialisierung der benötigten Datengeneratoren, das

Serialisieren und Laden von Netzen, die Durchführung des Trainings sowie die Auswertung einzelner Messdatenschnitte. Außerdem werden an dieser Stelle die spezifizierten Trainings-Callbacks konfiguriert. Während der Initialisierung können eine Reihe von Parametern, darunter sowohl der zu nutzenden Lernalgorithmus als auch die gewünschte Verlustfunktion, spezifiziert werden. Konkrete Architekturen sind im `models`-Paket abgelegt und beerben die Klasse. Hierbei wird jeweils die abstrakte Methode `__generate_model()` überschrieben. Sie liefert das zu nutzende `keras.Model`. Dieser Aufbau erlaubt die einfache Entwicklung verschiedener Netzarchitekturen sowie die Analyse des jeweiligen Trainingsfortschritts. Die zurzeit verwendete Architektur wird einmalig in der Datei `train.py` spezifiziert und kann somit leicht gewechselt werden. Sonderrollen nehmen hierbei folgende Architekturen ein:

- `models.AutoEncoderArchNone`: Leeres Model welches genutzt werden kann, wenn ein bereits vorhandenes und serialisiertes Model geladen werden soll.
- `models.AutoEncoderArchIdentity`: Spiegelt die Eingangswerte unmodifiziert direkt an den Ausgang wieder. Kann genutzt werden um die Funktionalität des Testrahmens zu überprüfen.

Bei der Definition neuer Architekturen ist darauf zu achten, dass sowohl Eingangs- und Ausgangslayer als auch die komprimierte Darstellung korrekt benannt sind. Hierzu stellt die Basisklasse Konstanten zur Benennung der Ebenen bereit. Es ist für spätere Arbeitsschritte zwingend notwendig diesen Konventionen zu folgen, um einen Zugriff auf alle benötigten Zwischenergebnisse zu erlauben.

5.3.4.3 Clustering-Autoencoder

Liegt ein trainierter Autoencoder vor, so kann dieser, wie in Abschnitt 4.5 beschrieben, um Clustering-Funktionalitäten erweitert werden. Hierzu ergänzt die Klasse `ClusteringAutoEncoder` einen bereits vorhandenen `AutoEncoder` um einen sogenannten `layer.ClusteringLayer`. Die Initialisierung der Layer-Gewichte erfolgt durch Zuweisung der mithilfe von k-Means bestimmten Clustermittelpunkte. Hierbei werden mehrere k-Means Durchläufe auf einer Untermenge von Messdaten durchgeführt. Der Durchlauf mit dem, gemäß Gleichung 4.9 besten Ergebnis wird zur Initialisierung der Gewichte genutzt. Das Serialisieren des initialisierten Netzes vor Trainingsbeginn kann durch den Kommandozeilenparameter `--init-checkpoint` ausgelöst werden.

Der Eingang des neuen Layers wird, wie in Abbildung 4.4 dargestellt, mit dem komprimierten Merkmalsvektor $\tilde{x}$ beschaltet. Das so entstandene Netz besitzt nun als zusätzlichen Ausgang die Ausgabe der neu eingeführten Ebene $c(\tilde{x})$. Durch das Hinzufügen des weiteren Ausgangs ändert sich das Format der zu generierenden Batches. Die Klasse `ClusteringSliceDataGenerator` erweitert einen `SliceDataGenerator` um die Angaben zur anzustrebenden Zielverteilung. Diese wird, wie in Abschnitt 4.5.4 detailliert beschrieben, am Ende jeder Trainingsepoche berechnet und den Batches jeweils in Teilen beigefügt.

Der genutzte Lernalgorithmus sowie die Verlustfunktion für die Rekonstruktion L_r werden, neben weiteren Parametern, vom spezifizierten `AutoEncoder` übernommen. Für den Ausgang des `ClusteringLayer` wird L_c, gegeben durch die KL-Divergenz, minimiert. Hierbei erfolgt die Kombination der Verlustfunktionen L gemäß Gleichung 4.14. Die genaue Gewichtung der Funktionen, sowie die gewünschte Cluster-Anzahl können bei der Initialisierung des Clustering-Autoencoders angegeben werden.

5.3.4.4 ResizeLayer / UpSampling2D

Innerhalb der Decoder-Komponente eines Autoencoders wird zum Erreichen der korrekten Ausgangs-Dimensionalität eine Dimensionserhöhung der Feature-Maps vorgenommen. Dies kann entweder mithilfe einer sogenannten *transposed convolution* [24], der inversen Operation einer Faltung, oder durch die Nutzung von *UpSampling*, dem Gegenstück zum *MaxPooling*, durchgeführt werden. Ersteres hat den Nachteil, dass es schnell zur Bildung von schachbrettmusterartigen Artefakten (engl. *checkerboard artifacts*) [25] kommt. Aus diesem Grund wird die Verwendung des rechenintensiveren UpSamplings bevorzugt.

Keras implementiert dieses Verfahren im `UpSampling2D`-Layer. Hierbei wird intern die TensorFlow-Funktion `tf.image.resize_images()` aufgerufen. Aufgrund von Rückwärtskompatibilität ist der Parameter `align_corners` standardmäßig auf `False` gesetzt[9]. Dies führt zu Problemen bei verschiedenen Interpolationsverfahren. Da dieser Parameter nicht über die Keras-API konfigurierbar ist[10], wurde der Layertyp `layer.ResizeLayer` eingeführt. Er unterstützt das Skalieren von zweidimensionalen Eingangsdaten mit verschiedenen Interpolationsverfahren. Hierbei wird direkt auf die beschriebene Funktion

[9] Siehe: https://github.com/tensorflow/tensorflow/issues/6720 (Abgerufen am 28.03.2019)
[10] Siehe: https://github.com/tensorflow/tensorflow/issues/23041 (Abgerufen am 28.03.2019)

zugegriffen, was das Setzen von `align_corners=True` erlaubt und somit die Interpolationsprobleme löst. Die Verwendung dieses Layers beim Entwurf von Autoencoder-Architekturen wird mit Nachdruck empfohlen.

5.3.5 Evaluation (`evaluate`)

Das `evaluate` Package besteht lediglich aus einem Modul. In diesem sind Funktionen zur Evaluation von Messdatenschnitten implementiert. Es werden alle spezifizierten Eingangsdatensätze mithilfe des gegebenen Clustering-Autoencoder Models evaluiert und hierbei je Datensatz eine, gemäß Tabelle 4.4 aufgebaute, serialisierte Clusterzuordnung ausgegeben. Aufgrund des maschinenlesbaren Ergebnisformats lässt sich ein trainiertes Netz somit auch zur automatisierten Auswertung im Produktivbetrieb nutzen. Zusätzlich können, durch Angabe des Kommandozeilenparameters `--plot`, sowohl die Eingangsdaten als auch die Autoencoder-Rekonstruktionen als Diagramm ausgegeben werden.

Liegen ferner getaggte Datensätze, wie in Abschnitt 4.6.3 beschrieben vor, so wird an dieser Stelle die Label-Zuweisung gemäß Gleichung 4.16 durchgeführt. Auf Basis dieser Zuweisung können sowohl eine Wahrheitsmatrix als auch ein Klassenreinheitsdiagramm erstellt werden.

6 Ergebnisse

Bevor das vorgestellte Verfahren für reale Messdaten Anwendung fand, wurde es mithilfe des MNIST Datensatzes handgeschriebener Ziffern [26] getestet. Dies hat den Vorteil, dass hierbei getaggte Daten vorliegen und somit eine objektive Verifikation des Ergebnisses ermöglicht wird. Nach Auswertung dieser Testreihe konnte ein neuronales Netz für LIDAR-Ceilometer Datensätze trainiert und erfolgreich für das Clustering genutzt werden. In diesem Kapitel wird sowohl auf den Versuchsaufbau als auch auf die erzielten Resultate der einzelnen Experimente eingegangen.

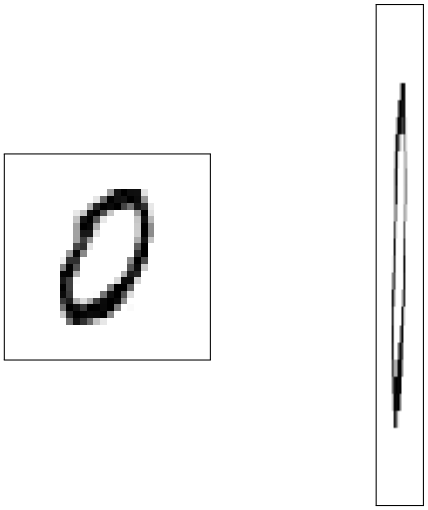
6.1 Klassifikation von MNIST Ziffern

Um die Funktionalität des vorgestellten Verfahrens zu zeigen, wurde es auf das klassische Problem der Klassifikation handgeschriebener Ziffern angewendet. Hierbei kam der bereits eingangs erwähnte MNIST Datensatz zum Einsatz. Dieser umfasst 60,000 Trainings- sowie 10,000 Testbilder, welchen jeweils die entsprechend dargestellte Ziffer zugeordnet ist. Da das vorgestellte Verfahren unüberwacht arbeitet, wurden die Labels ausschließlich für die spätere Evaluation der Trainingsergebnisse und nicht während des Trainings der Netze genutzt. Im folgenden Abschnitt werden die Vorgehensweise inklusive der Datenvorverarbeitung, die gewählte Netzarchitektur sowie die erzielten Ergebnisse besprochen.

6.1.1 Vorgehen

Damit ein möglichst repräsentatives Ergebnis entsteht, wurden die MNIST-Ziffern der Form der Ceilometer-Messdaten angeglichen. Hierzu wurde für jede der 28x28 Pixel großen Ziffern ein Messdatenschnitt, gemäß der in Kapitel 3 bestimmten Parameter, mit einer Größe von 8x256 Datenpunkten erzeugt. Die Skalierung der Eingangsdaten

erfolgt mithilfe einer Nearest-Neighbour Interpolation. Ein Beispiel des hieraus resultierenden Formats ist in Abbildung 6.1 zu sehen. Zur Verbesserung der Darstellung ist das Zielformat 6.1b im Maßstab 2:1 abgebildet.



(a) 28x28 (b) 8x256 2:1

Abbildung 6.1: Vergleich MNIST Ziffern Original- (a) und Zielformat (b)

Neben der Formatanpassung wurde ebenfalls eine Skalierung der Messwerte durchgeführt. Da die Ziffern als 8-Bit Grauwertbilder vorliegen, ergibt sich ein Wertebereich von 0 bis 255 für jeden der Pixel. Dieser wurde, gemäß Gleichung 4.6 auf das Intervall $[0, 1]$ skaliert, um einem normierten Ceilometer-Datensatz zu entsprechen. Neben den als Rückstreuprofil transformierten Bilddaten wurden ebenfalls die jeweils zugewiesenen Labels im generierten Messdatenschnitt abgelegt, um eine spätere Bewertung der Klassifikation vornehmen zu können. Die Aufteilung der Ziffern in Trainings- beziehungsweise Testdaten wurde exakt beibehalten.

Nachdem die Ziffern als Messdatenschnitte vorlagen, wurde zuerst ein Autoencoder trainiert und nach Beendigung des initialen Trainings um den eingeführten Clustering-Layer (siehe Abschnitt 4.5) erweitert. Anschließend wurde der erzeugte Clustering-Autoencoder weiter trainiert, bis ein zufriedenstellendes und optimiertes Clustering erzielt werden konnte. Den hieraus entstandenen Clustern wurde anschließend automatisiert, unter Zuhilfenahme der Labels, jeweils eine der zehn Klassen zugeordnet.

6.1.2 Architektur

Im Rahmen dieses Versuches wurden verschiedene Netzarchitekturen trainiert und evaluiert. Der in Abbildung 6.2 dargestellte Clustering-Autoencoder hat sich in diesen Tests als geeignete Architektur herausgestellt. Er verwendet drei `Conv2D`-Layer mit einer Filterkern-Größe (engl. *kernel size*) von 5x5 beziehungsweise 3x3 und einer Schrittweite (engl. *stride*) von 1. Die Reduktion der Feature-Map-Größe (engl. *subsampling*) erfolgt durch Verwendung von `MaxPooling2D` auf jeweils die halbe Eingangsdimensionalität. Die entstehenden Feature-Maps werden dann in einen `Dense`-Layer überführt, welcher folgend in die komprimierte Repräsentation der Daten übergeht. Hierfür wurde ein weiterer `Dense`-Layer mit 10 Neuronen gewählt. Die Decoder-Stufe ist analog zum Encoder-Teil aufgebaut, wobei alle Operationen jeweils durch ihre Inversen ersetzt wurden. So zum Beispiel `MaxPooling2D` durch die in Abschnitt 5.3.4.4 eingeführten `ResizeLayer` als auch `Flatten` durch `Reshape`. Abschließend wurde die Größe des `ClusteringLayer`, gemäß der Anzahl der zu findenden Cluster beziehungsweise der zuzuweisenden Labels, auf ebenfalls 10 Neuronen festgelegt. Die detaillierte Beschreibung der verwendeten Keras-Models ist den Quellcode-Auflistungen A.2 sowie A.3 zu entnehmen.

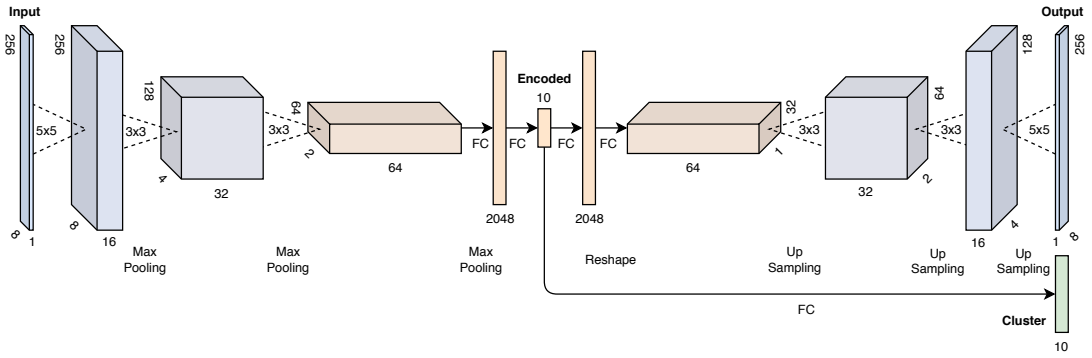


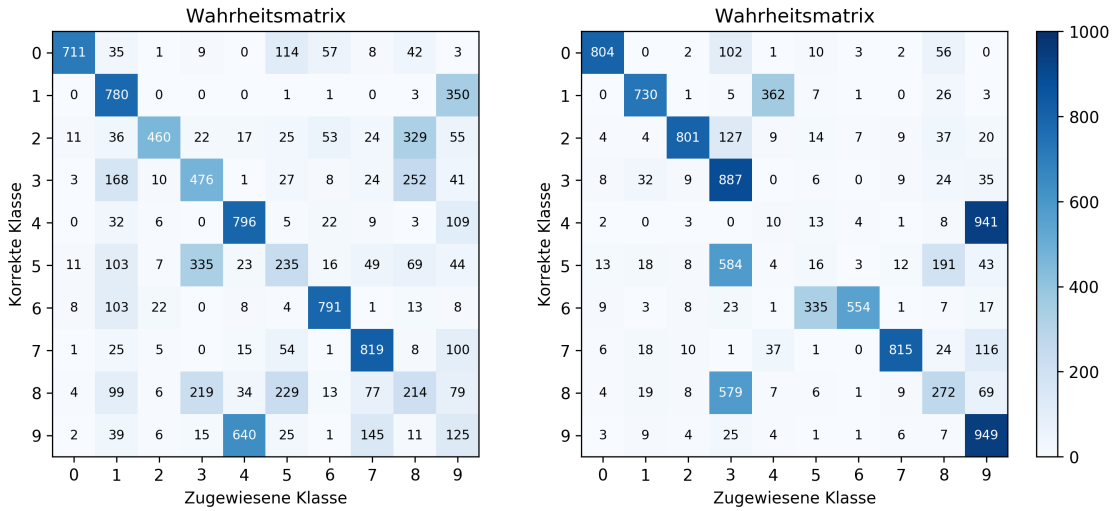
Abbildung 6.2: Architektur des MNIST Clustering-Autoencoders

Als Aktivierungsfunktion kommt einheitlich die *Rectified Linear Unit* (ReLU) Funktion (siehe Gleichung A.2) zum Einsatz. Lediglich der Ausgangs-Layer des Decoders nutzt abweichend die Sigmoid Funktion (siehe Gleichung A.1) um dem Wertebereich der Eingangsdaten zu gleichen.

6.1.3 Auswertung

Zur Bewertung des Trainingsergebnisses wurden mehrere Eigenschaften sowohl des Trainingsverlaufes als auch des resultierenden neuronalen Netzes betrachtet. Während des Trainings wurden die epochenweise vorhandenen Logdaten mithilfe von TensorBoard untersucht, um sicherzustellen, dass ein Lernfortschritt ohne Überanpassung (engl. *overfitting*) erzielt wird. Es konnte gezeigt werden, dass der im Rahmen des Pre-Trainings minimierte Rekonstruktionsverlust L_r während der Clusteroptimierung beibehalten werden konnte. Gleichzeitig ließ sich ebenfalls die KL-Divergenz zur gewählten Zielverteilung L_c minimieren. Es kommt also, trotz gleichzeitiger Optimierung beider Kriterien, durch die Kombination der Verlustfunktionen zu keiner nennenswerten Verschlechterung der Rekonstruktion bei zeitgleicher Verbesserung des Clustering-Ergebnisses.

Die Beurteilung der Güte des Clusterings erfolgt auf Basis des gesamten Testdatensatzes. Um die Effektivität der Optimierung nachzuweisen, wurde der lediglich initialisierte Clustering-Autoencoder mit einem trainierten Netz verglichen. Hierbei entspricht das lediglich initialisierte Model dem Clusteringergebnis des besten k-Means Durchlaufs, gemäß Abschnitt 4.5.2, und somit einer unoptimierten Anwendung des Verfahrens auf die komprimierten Merkmalsvektoren. Beide Netze haben jeweils die selben Datensätze zur Evaluation präsentiert bekommen. Hierbei wurde zum einen die korrekte Gruppierung gleicher Ziffern und zum anderen die Reinheit der resultierenden Klassen betrachtet.

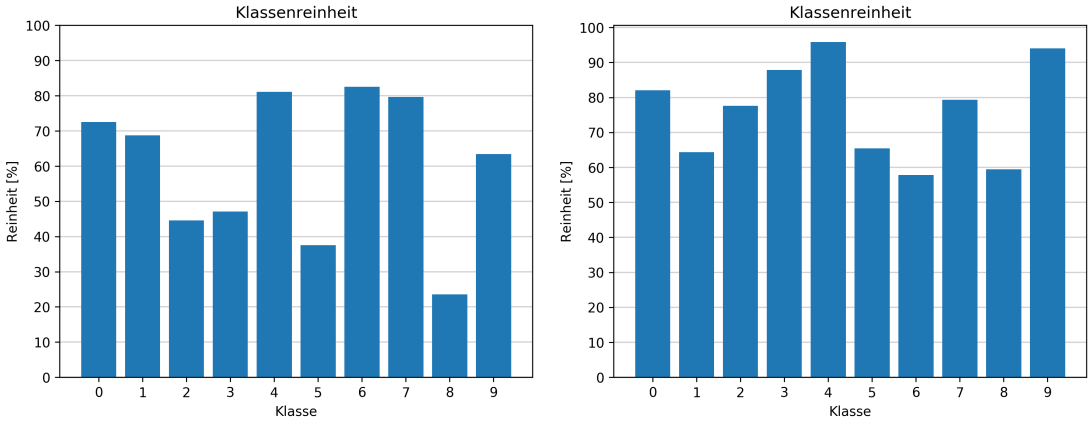


(a) Initiales k-Means Clustering (b) Trainierter Clustering-Autoencoder

Abbildung 6.3: Vergleich der MNIST Clustering-Ergebnisse

Eine Analyse der Gruppierung gleicher Ziffern wird mithilfe einer Wahrheitsmatrix (engl. *confusion matrix*) durchgeführt. Hierbei wird für jede vorhandenen Klasse ein Histogramm über die Mitglieder dieser gebildet und in Matrixform aufgetragen. Somit stehen in der Hauptdiagonalen die Anzahlen aller korrekt zugeordneten Datensätze, während alle anderen Elemente einer falschen Zuordnung entsprechen. Dies ermöglicht es, sowohl die Genauigkeit der Klassifikation als auch die Streuung innerhalb der einzelnen Klassen zu beurteilen. Die Zuordnung der Labels zu je einem der Cluster erfolgt wie in Abschnitt 4.6.3 beschrieben. Abbildung 6.3 zeigt einen Vergleich solcher Matrizen eines Clusterings direkt nach der Initialisierung gegenüber dem Ergebnis eines trainierten Clustering-Autoencoders.

Ein Ziel der angestrebten Verteilung (siehe Gleichung 4.13) ist unter anderem die Reduktion der Streuung innerhalb einer Klasse. Zur Verifikation dieser Eigenschaft wird die Klassenreinheit (engl. *cluster purity*) betrachtet. Sie entspricht dem prozentualen Anteil der dominierenden Klasse innerhalb eines Clusters. Je höher dieser Wert, desto weniger durchmischt ist ein einzelnes Cluster. Abbildung 6.4 zeigt, analog zum Vergleich der Wahrheitsmatrizen, den Vergleich der Klassenreinheiten für beide Netze.

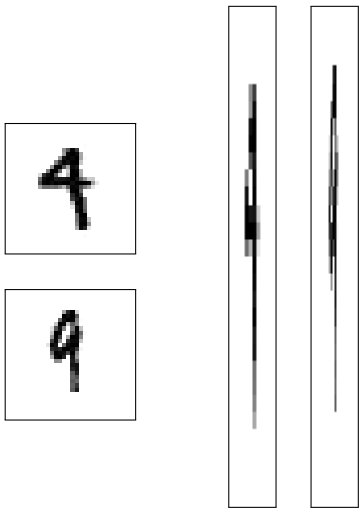


(a) Initiales k-Means Clustering (b) Trainierter Clustering-Autoencoder

Abbildung 6.4: Vergleich der MNIST Klassenreinheiten

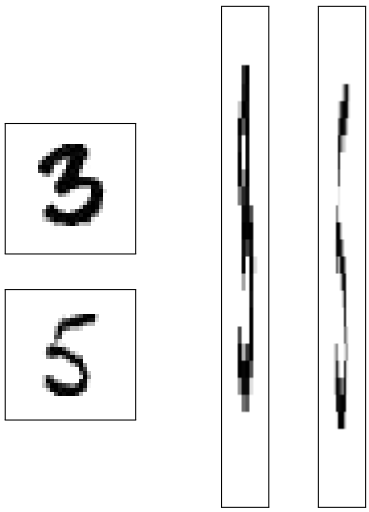
Betrachtet man die beschriebenen Diagramme, so lässt sich festhalten, dass ein Clustering mit k-Means auf Basis der komprimierten Merkmalsvektoren bereits ein nutzbares Ergebnis liefert. Hierbei fällt jedoch auf, dass sich häufig noch große Mengen falsch klassifizierter Ziffern innerhalb einer Klasse befinden. So beinhaltet beispielsweise die Klasse 3 auf 476 korrekt erkannte Zahlen weitere 252 Ziffern, denen das Label 8 sowie 168 Ziffern, denen das Label 1 zugewiesen wurde. Dies führt zu einer geringen

Klassenreinheit. Vergleicht man den frisch initialisierten mit dem trainierten Clusterig-Autoencoder, so ist eine deutliche Verbesserung der Clusterzuordnungen hinsichtlich der gewählten Zielverteilung ersichtlich. Sowohl die Erkennungsrate einzelner Ziffern, als auch die Reinheit der Klassen, konnte mithilfe des angeschlossenen Trainings signifikant verbessert werden. Einzelne Zahlenklassen haben jedoch auch eine starke Abnahme der Erkennungsrate erfahren. Dies ist bei den Klassen 4 sowie 5 der Fall. Hier wurden unterschiedliche Ziffern fälschlicherweise in einer anderen Klasse zusammengefasst. Dieses Verhalten lässt sich jedoch auf die Verzerrung der Eingangsdaten, genauer auf die horizontale Stauchung dieser, zurückführen. Abbildung 6.5 und Abbildung 6.6 zeigen für die problematischen Klassen die Eingangsdaten nach der Transformation in das Format der Ceilometer-Messdatenschnitte. Hierbei sind die gestauchten Ziffern zur besseren Erkennbarkeit bereits verbreitert dargestellt. Es ist zu sehen, dass sich sowohl Vieren und Neunen als auch Dreien und Fünfen durch die Reduktion der horizontalen Auflösung stark ähneln. Demnach liegt es nahe, diese jeweils innerhalb eines Clusters zu gruppieren, was die beobachteten Trainingsergebnisse erklärt.



(a) 28x28 (b) 8x256 2:1

Abbildung 6.5: Ähnlichkeit der transformierten Ziffern 4 und 9



(a) 28x28 (b) 8x256 2:1

Abbildung 6.6: Ähnlichkeit der transformierten Ziffern 3 und 5

Neben den in diesem Abschnitt bereits vorgestellten Auswertungen zeigen Abbildung A.1 sowie Abbildung A.2 einen Auszug der jeweils einem Cluster zugehörigen Ziffern direkt nach der Initialisierung sowie nach weiterführendem Training. Hierbei sind jeweils jene Messdatenschnitte dargestellt, welche sich am nächsten zu den jeweiligen Clustermittelpunkten befinden (engl. *high confidence assignments*).

6.1.4 Ergebnis

Es konnte gezeigt werden, dass das vorgestellte unüberwachte Lernverfahren in der Lage ist, die transformierten MNIST Ziffern erfolgreich in komprimierte Merkmalsvektoren zu überführen, auf Basis welcher ein Clustering durchgeführt wurde. Ferner konnte dieses initiale Clustering im Rahmen des anschließenden Trainings weiter verbessert werden. Die Analyse mithilfe des transformierten MNIST-Datensatzes ließ eine Auswertung über die Güte der Clusterzuweisungen für eine Klassifikation zu. Somit konnte erfolgreich gezeigt werden, dass sich sowohl das vorgestellte Verfahren, als auch die genutzte Netzarchitektur für die unüberwachte Merkmalsextraktion sowie ein anschließendes Clustering von Messdatenschnitten nutzen lassen. Die aus diesem Versuch gewonnenen Erkenntnisse fließen in die Anwendung des Verfahrens auf reale Ceilometer-Messdatensätze, welche ausführlich in Abschnitt 6.2 beschrieben wird, mit ein.

6.2 Clustering von Aerosol-Rückstreuprofilen

Nachdem die grundlegende Funktionalität des vorgestellten Verfahrens mithilfe des im vorherigen Abschnitt beschriebenen Experiments gezeigt werden konnte, wurde es auf die realen Ceilometer-Messdaten angewandt. Die dafür durchgeführten Änderungen der Netzarchitektur sowie die Auswertung der Resultate sind im folgenden Abschnitt beschrieben.

6.2.1 Vorgehen

Die Erzeugung der Messdatenschnitte wurde gemäß der Verfahrensbeschreibung (siehe Abschnitt 4.2) durchgeführt. Hierfür wurden alle im Jahr 2018 angefallenen Messdatensätze des Ceilometers in Leipzig genutzt und zu 90% in Trainings- sowie zu 10% in Testdaten aufgeteilt. Die generierten Ausschnitte der Rückstreuprofile wurden, wie im vorherigen Versuch ebenfalls geschehen, gemäß Gleichung 4.6 auf das Intervall $[0, 1]$ skaliert. Hierzu wurden die Werte aus dem von der Implementation erzeugten Analysebericht genutzt, welche im Detail der Tabelle A.1 zu entnehmen sind. Analog zur Voruntersuchung folgte auf das Pre-Training eines Autoencoders, welcher anschließend als Basis für den Clustering-Autoencoder diente, das eigentliche Training zur Verbesserung des Clusterings.

Da für die erzeugten Messdatenschnitte, im Gegensatz zum MNIST-Zifferndatensatz, keine Labels existieren, kann folglich nicht auf die im Vorversuch genutzten Auswertungsmethoden zurück gegriffen werden[1]. Daher wird im Rahmen dieses Versuches keine automatische Klassifikation, sondern eine manuelle Bewertung der einzelnen Cluster durchgeführt. Diese Methodik sowie die erzielten Ergebnisse sind detailliert in Abschnitt 6.2.3 beschrieben.

6.2.2 Architektur

Der Aufbau des verwendeten neuronalen Netzes unterscheidet sich nur wenig von der im Vorversuch genutzten Architektur, da sich diese bereits erfolgreich als funktionsfähig erwiesen hat. Abbildung 6.7 zeigt das verwendete Netz. Die detaillierte Zusammenfassung der Keras-Models kann den Quellcode-Auflistungen A.4 sowie A.5 entnommen werden.

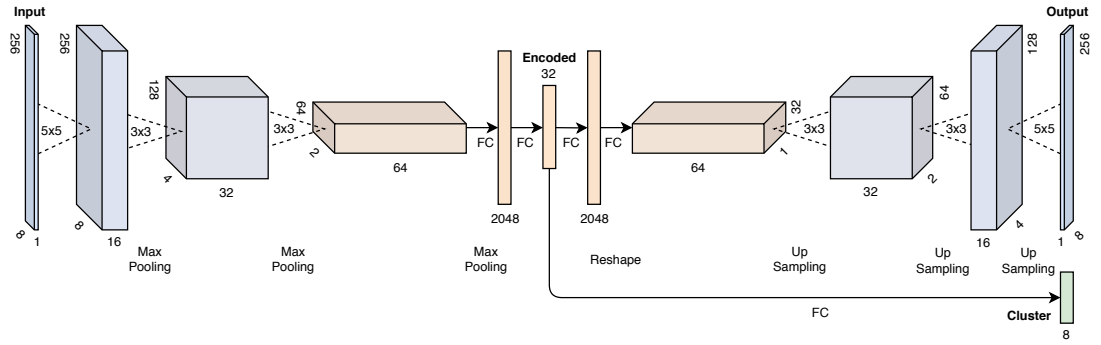


Abbildung 6.7: Architektur des Clustering-Autoencoders für Ceilometerdaten

Eine Anpassung der Eingangs- und Ausgangsdatendimensionalität ist nicht vonnöten, da das Format der Messdatenschnitte dem der transformierten MNIST-Ziffern gleicht. Jedoch wurde die Anzahl der zu findenden Cluster auf 8 verringert. Dies beruht darauf, dass gemäß Abschnitt 3.1 folgende Klassen von Phänomenen innerhalb eines Messdatenschnittes auftreten können: Wolken, Aerosolschichten und Niederschlag. Hieraus ergibt sich über die Potenzmenge $\mathcal{P}$ der Klassen die gewählte Clusteranzahl. Außerdem wurde die Größe des kodierten Merkmalsvektors auf 32 gesetzt, um die erhöhte Komplexität der Eingangsdaten zu kompensieren. Schließlich wurde noch die zuvor eingesetzte Aktivierungsfunktion ReLU einheitlich durch die ELU Funktion (siehe Gleichung A.3)

[1] Sowohl die Aufstellung einer Wahrheitsmatrix als auch die Analyse der Klassenreinheiten benötigen jeweils korrekte Labels zum Abgleich der innerhalb eines Clusters vorkommenden Klassen.

ersetzt, um einem eventuellen Auftreten des sogenannten *dying neuron problems* [27] entgegenzuwirken. Die Aktivierungsfunktion des Ausgabe-Layers des Autoencoders bleibt unverändert die Sigmoid-Funktion.

6.2.3 Auswertung

Wie schon im Vorexperiment konnte auch hier ein Lernfortschritt ohne Überanpassung (engl. *overfitting*) beobachtet werden. Ebenso war es möglich, den Rekonstruktionsverlust L_r trotz zeitgleicher Minimierung der KL-Divergenz zur Zielverteilung L_c nicht signifikant ansteigen zu lassen. Gemäß dieser Lernmetriken konnte also auch mit den realen Ceilometer-Daten ein Lernerfolg erzielt werden.

Da für die Ceilometer-Messdatenschnitte keine Labels vorliegen, ist eine automatische Zuweisung von Klassen zu jedem der Cluster nicht möglich. Demnach entfällt ebenso das Bilden einer Wahrheitsmatrix oder eines Klassenreinheit-Diagramms, wie für die Bewertung der MNIST-Klassifikation genutzt. Ferner existieren keine Informationen über die Verteilung beziehungsweise die Anzahl der Vorkommnisse einzelner Phänomene innerhalb der generierten Messdatenschnitten. Daher ist eine Analyse der Größe der gebildeten Cluster wenig sinnvoll.

Um dennoch einen Eindruck vom Clusteringergebnis gewinnen zu können, wurden alle Testdatensätze mithilfe des trainierte Netzes evaluiert und den entsprechenden Clustern zugeordnet. Dem folgte eine Untersuchung mehrerer Stichproben je Cluster. Ferner zeigen die Abbildungen A.3 und 6.8 mehrere Messdatenschnitte pro Clusters, jeweils für den frisch initialisierten, sowie den trainierten Clustering-Autoencoder. Hierbei sind die dargestellten Datensätze so gewählt, dass sie aus allen Elementen eines Clusters denjenigen mit der höchsten Clusterzugehörigkeit entsprechen. Diese ist hierbei ein Maß für die Wahrscheinlichkeit, dass ein Datum dem jeweiligen Cluster angehört und entspricht somit dem normierten Wert des Ausgabeneurons des jeweiligen Clusters im Clustering-Layer (siehe Abschnitt 4.5.3). Mithilfe dieser Darstellung lassen sich die jeweils repräsentativsten Datensätze in kompakter Form begutachten, um hieraus einen Eindruck für die durch das jeweilige Cluster repräsentierten Eigenschaften zu erhalten.

Betrachtet man die in Abbildung 6.8 dargestellten Ausschnitte, so lassen sich einige der in Abschnitt 3.1 beschriebenen Phänomene sofort wiederfinden. Innerhalb des Clusters 1 sind Messdatenschnitte ohne Wolken und ohne prägnante Aerosolschichten gruppiert, wohingegen ausgeprägte Aerosolschichten innerhalb der Cluster 4 und 6 repräsentiert

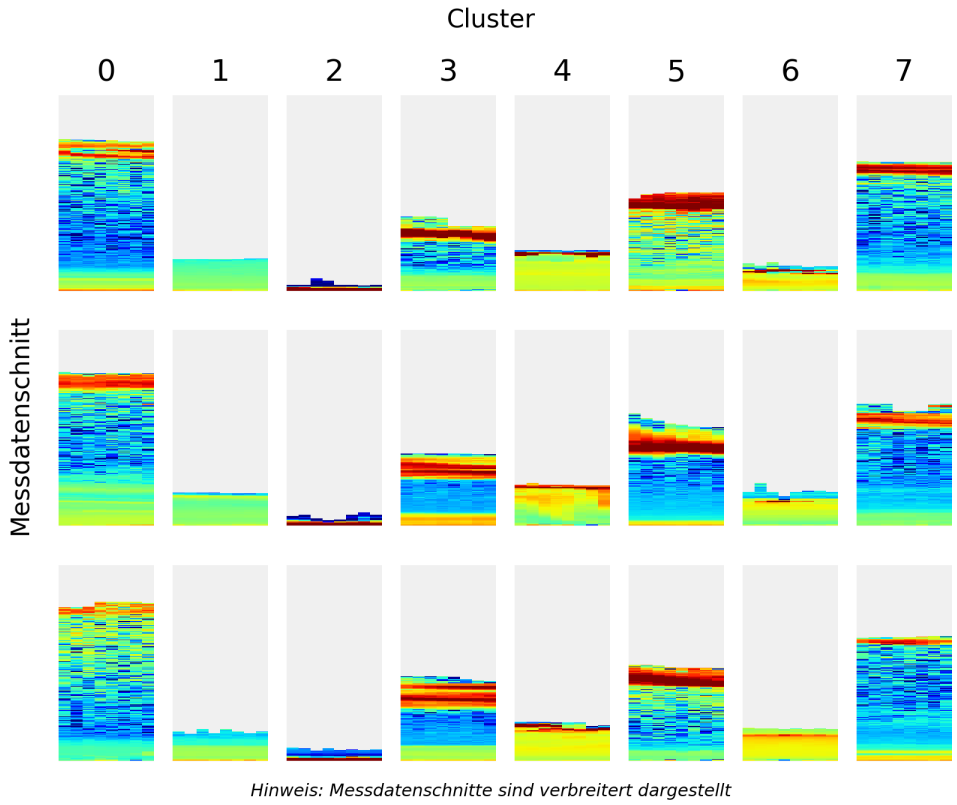
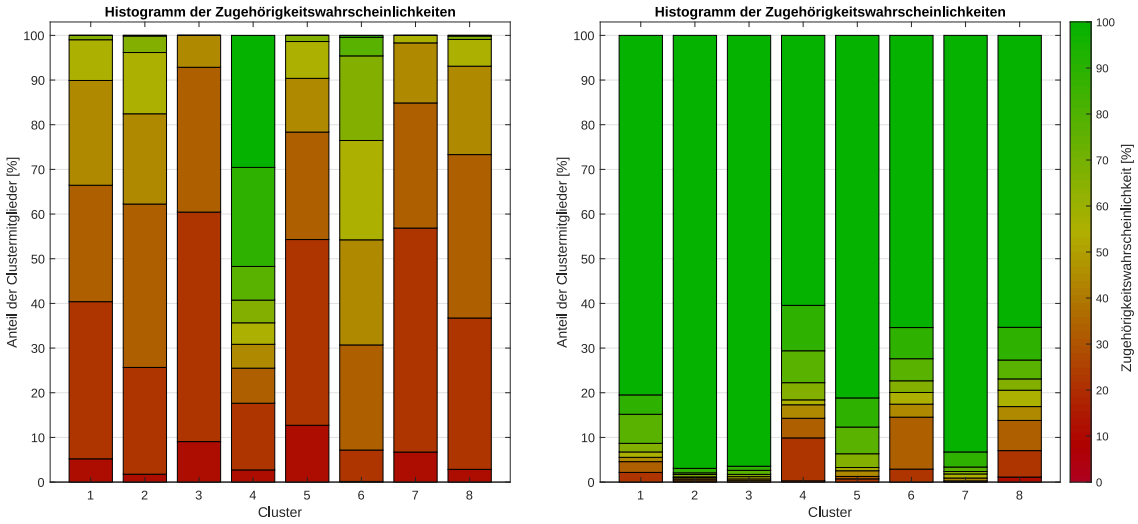


Abbildung 6.8: Ceilometer-Messdatenschnitte mit der höchsten Clusterzugehörigkeit nach Training

sind. Bereiche, welche ein Vorkommen von Bodennebel aufweisen, finden sich im Cluster 2. Die restlichen Cluster beinhalten Wolken verschiedener Intensitäten sowie Höhen. Ein Cluster, welches Niederschlag als prominentes Phänomen beinhaltet, konnte innerhalb dieses Experiments jedoch nicht nachgewiesen werden.

Zusätzlich zu den Messdatenschnitten mit der höchsten Clusterzugehörigkeit zeigt Abbildung 6.9 die Verteilung dieser für die Gesamtheit aller Testdatensätze je Cluster. Hierzu wurde für jedes Cluster ein Histogramm über die jeweiligen Clusterzugehörigkeiten der einzelnen Datensätze erstellt. Die Breite der Wertebereiche der Histogrammklassen beträgt jeweils $0.1 \mathrel{\hat{=}} 10\%$, wodurch sich insgesamt 10 Klassen ergeben. Jede Säule des Diagramms stellt ein einzelnes Cluster dar, wobei die Farbe der unterschiedlichen Bereiche der Clusterzugehörigkeit, welche von der jeweiligen Histogrammklasse repräsentiert wird, entspricht. Die Höhe einer jeden Klasse steht hierbei für den auf der Ordinatenachse dargestellten prozentualen Anteil der Mitglieder eines Clusters.



(a) Initiales k-Means Clustering

(b) Trainierter Clustering-Autoencoder

Abbildung 6.9: Vergleich der Verteilung der Clusterzugehörigkeiten aller Messdatenschnitte je Cluster

Diese Analyse zeigt, dass direkt nach Abschluss des Pre-Trainings und der darauf folgenden Initialisierung der Cluster die getroffene Clusterzuordnung vieler Messdatenschnitte noch nicht von hohem Vertrauen in die Zuweisung geprägt ist. So weisen zum Beispiel über 10% der Mitglieder des Clusters 5 eine unter 20% liegende Wahrscheinlichkeit auf, überhaupt zu genau diesem Cluster zugehörig zu sein[2]. Ein weiteres Beispiel hierfür ist Cluster 3, bei welchem 60% der Datensätze unterhalb einer Zugehörigkeitswahrscheinlichkeit von 30% liegen. Betrachtet man hingegen den weiter trainierten Clustering-Autoencoder, so zeigt sich eine deutliche Steigerung in der Eindeutigkeit der Clusterzuordnungen. Beispielsweise besitzen die Elemente der zuvor als problematisch identifizierten initialen Cluster 5 und 3 nun zu 80% beziehungsweise sogar 95% eine Clusterzugehörigkeit von mindestens 90%. Die optimierten Cluster sind demnach besser von einander separiert und weniger Messdatenschnitte sind im Merkmalsraum zwischen mehreren Clustern positioniert. Diese Verbesserung zeigt anschaulich die Auswirkungen der Optimierung hin zur gewählten Zielverteilung.

[2]Somit besteht eine Gesamtwahrscheinlichkeit von über 80%, dass der Messdatenschnitt zu einem der anderen Cluster zugehörig ist. Der korrespondierende Merkmalsvektor weist also zu mehreren Clustermittelpunkten eine hohe Ähnlichkeit auf, wobei die des zugewiesenen Clusters am höchsten ist.

6.2.4 Ergebnis

Es konnte gezeigt werden, dass sich das vorgestellte Verfahren ebenfalls auf die realen Ceilometerdatensätze erfolgreich anwenden lässt. Trotz freier Wahl der Hyperparameter konnte bereits ein nutzbares Clustering-Ergebnis erzielt werden. Hierbei war es möglich Cluster nachzuweisen, welche jeweils Messdatenschnitte mit Vorkommnissen von Wolken, Aerosolschichten oder Nebel beinhalteten. Ferner konnte ein Cluster für Ausschnitte ohne detektierbare Phänomene gefunden werden. Somit ist eine einfache Zuordnung dieser Labels zu den jeweiligen Clustern problemlos möglich. Nichtsdestotrotz wurde ebenfalls gezeigt, dass noch Bedarf an der Optimierung der Hyperparameter besteht, um das Clusteringergebnis weiter zu verbessern. Diese beinhalten unter anderem die genutzte Clusteranzahl, die Größe des enkodierten Merkmalsvektors oder auch die Wahl der Zielverteilung. Vorschläge zur Wahl der Hyperparameter als auch zur generellen Verbesserung des Verfahrens werden detailliert in Abschnitt 7.2 besprochen.

7 Fazit

7.1 Zusammenfassung

Ziel dieser Arbeit war die Entwicklung eines prototypischen Verfahrens zur unüberwachten Merkmalsextraktion aus Aerosol-Rückstreuprofilen. Dies ist unter Zuhilfenahme neuronaler Faltungsnetze (CNNs) geschehen. Es wurde ein Verfahren eingeführt, welches Ausschnitte aus Ceilometer-Messreihen durch einen Convolutional-Autoencoder in komprimierte Merkmalsvektoren überführt und diese anschließend für ein Clustering nutzt. Ferner wird die gebildete Clusterzuweisung mit weiterem Trainingsverlauf parallel zur Autoencoder-Rekonstruktion fortlaufend optimiert.

Durch zwei Experimente konnte die Funktionalität dieses Verfahrens, einmal anhand des MNIST-Zifferndatensatzes sowie darauf folgend auch für reale Messdaten aus LIDAR-Ceilometern, gezeigt werden. Hierbei erwies sich sowohl der Autoencoder-Ansatz zum Erzeugen der komprimierten Merkmalsvektoren, als auch die anschließende Optimierung des Clusterings durch Minimierung der KL-Divergenz zu einer gewählten Zielverteilung, als geeignet. In diesem Rahmen war es bereits möglich, einige der zu detektierenden meteorologischen Phänomene in Clustern zu gruppieren. Auf Basis dieser Cluster konnten die jeweiligen Messdatenschnitte dann halbautomatisch mit Labels versehen werden.

Es wurde eine Softwarelösung implementiert welche es erlaubt, die einzelnen Teilschritte des Verfahrens modular auszuführen. Somit wird es ermöglicht, unterschiedliche Netzarchitekturen mit jeweils frei wählbaren Hyperparametern zu trainieren und miteinander zu vergleichen. Hiermit konnte eine Basis geschaffen werden, auf welcher im Rahmen weiterer Arbeiten in diesem Bereich aufgebaut werden kann. Neben dem entwickelten Python-Programm konnten außerdem verschiedene Eigenschaften der vorliegenden Messdaten analysiert werden, welche unter anderem für die Vorverarbeitung der Eingangsdaten von Nutzen sind.

7.2 Ausblick

Es konnte im Rahmen der beiden Experimente erfolgreich gezeigt werden, dass das vorgestellte Verfahren für die Ceilometer-Messdaten geeignet ist. Nichtsdestotrotz stellt diese Arbeit lediglich eine Basis dar, welche im Rahmen weiterführender wissenschaftlicher Arbeiten ausgebaut sowie optimiert werden kann.

Wie bereits in der Ergebnisbewertung des zweiten Versuchs (siehe Abschnitt 6.2.4) festgehalten, lässt sich das mit den hier gewählten Hyperparametern erzielte Clustering noch weiter verbessern. Dazu sind zusätzliche Versuche mit unterschiedlichen Bedingungen vonnöten. Außerdem sind im Verlauf der Arbeit einige Überlegungen aufgekommen, welche leider aufgrund des zeitlich begrenzten Rahmens nicht mehr in dieser Arbeit behandelt werden konnten. Dennoch ist es sinnvoll, an dieser Stelle kurz auf einen Teil dieser Vorschläge einzugehen, um diese gegebenenfalls in Folgeexperimenten zu nutzen.

Einer der zu spezifizierenden Hyperparameter ist die Größe des komprimierten Merkmalsvektors, also die Anzahl der Neuronen des mittleren Autoencoders-Layers. Er ist ausschlaggebend für die Güte der erzeugten komprimierten Repräsentationen. Wird er zu groß gewählt, so ist das Netz in der Lage, spezielle Eigenheiten einzelner Datensätze[1] zu lernen, worunter die Fähigkeit zur Verallgemeinerung leidet. Wird er zu gering gewählt, ist das Netz nicht in der Lage alle benötigten Informationen zu kodieren und die Qualität der Rekonstruktion leidet. Im Allgemeinen lässt sich jedoch sagen, dass je repräsentativer einzelne Features des Merkmalsvektors sind, umso besser lassen sich diese Clustern. Im Rahmen weiterer Versuche ist zu prüfen, inwieweit die Dimension des Merkmalsvektors verringert werden kann, ohne signifikante Qualitätsverluste in der Rekonstruktion der Messdatenschnitte zu erhalten beziehungsweise in wieweit diese Verluste zur Clustering-Verbesserung tolerierbar sind.

Neben der Dimensionalität des Merkmalsvektors kann ebenso die Anzahl der zu findenden Cluster angepasst werden. Zur Bestimmung der optimalen Clusteranzahl bieten sich Methoden wie beispielsweise die sogenannte Ellenbogen-Methode (engl. *elbow method*) [28] oder ein auf der Rate-Verzerrungs-Theorie (engl. *rate-distortion theory*) aufbauendes Verfahren [29] an. Außerdem ist ein Vergleich der Silhouettenkoeffizienten [30] zur Bestimmung der optimalen Clusteranzahl möglich.

[1] Dies kann ebenso Signalrauschen einschließen, falls die Größe deutlich zu hoch gewählt wird.

Einen weiteren wichtigen Parameter bildet die angestrebte Zielverteilung, zu welcher die KL-Divergenz minimiert wird. Entspricht das optimierte Clustering nicht der gewünschten Vorstellung oder verschlechtert weiteres Training dieses sogar, so kann die Wahl einer anderen Zielverteilung vorgenommen werden. Beispielsweise lässt sich die Größe der gebildeten Cluster über diesen Parameter beeinflussen. Sind weitere Eigenschaften über die Eingangsdaten bekannt, so können diese im Rahmen weiterer Experimente an dieser Stelle ebenfalls mit einfließen.

Ferner besteht die Möglichkeit des Vergleichs unterschiedlicher Netzarchitekturen. Hierbei kann unter anderem erprobt werden, inwiefern die Nutzung einer dem Seitenverhältnis der Messdatenschnitte angepassten Filtergröße die Qualität der entstehenden Merkmalsvektoren beeinflusst. Für die erste Faltungsebene könnten somit beispielsweise Filter auf voller Breite genutzt werden, um eine gezielte Abtastung einzelner Höhenschichten zu erreichen. Durch den Aufbau der entwickelten Softwarelösung ist es möglich, an dieser Stelle nahezu beliebige Netzarchitekturen zu vergleichen. Diese sind lediglich insofern limitiert, als dass sie unüberwacht trainiert werden müssen.

Zuletzt sei auf die Integration des Verfahrens in den Rahmen eines *Active Learning* Szenarios [31] eingegangen. Bei diesen Lernverfahren wird dem Lernalgorithmus die Möglichkeit gegeben, ein Orakel nach einem korrekten Label für den jeweiligen Datensatz zu befragen. Dieses Orakel ist in der klassischen Anwendung durch einen Menschen mit dem benötigten Expertenwissen realisiert. Ziel dieser Methode ist eine Reduktion der benötigten Annotationszeit (engl. *annotation time*). Ein weiteres Experiment kann prüfen, inwieweit eine Vorschaltung des vorgestellten Verfahrens die benötigte Interaktion mit dem Orakel minimieren kann. Hierbei kann beispielsweise jede Abfrage des Lernalgorithmus, bevor sie an den annotierenden Menschen weitergereicht wird, zuerst durch das hier vorgestellte Verfahren bewertet werden. Kommt es hierbei zu einer Aussage mit einer hohen Korrektheitswahrscheinlichkeit, so kann die manuelle Befragung des Orakels übersprungen werden. Lediglich für Messdatenschnitte, die nicht mit einer hohen Wahrscheinlichkeit zuzuordnen sind, kann der Mensch befragt werden. Dieses Vorgehen könnte den Annotationsaufwand für die große Menge an Ceilometer-Messdatensätzen drastisch reduzieren.

8 Urheberrechtshinweis

Die genutzten LIDAR-Ceilometer Messdatensätze stammen aus dem Ceilometer-Messnetz des deutschen Wetterdienstes (DWD)[1]. Die Daten wurden mit Einverständnis des DWD im Rahmen dieser Arbeit zu Forschungszwecken genutzt und unterliegen dem Urheber- oder Leistungsschutzrecht.

[1] Siehe: https://www.dwd.de/DE/forschung/atmosphaerenbeob/zusammensetzung_atmosphaere/aerosol/inh_nav/aerosolprofile_node.html (Abgerufen am 18.03.2019)

Literatur

[1] M. Hervo, Y. Poltera und A. Haefele, "An empirical method to correct for temperature-dependent variations in the overlap function of CHM15k ceilometers", *Atmospheric Measurement Techniques*, Jg. 9, Nr. 7, S. 2947–2959, 2016. DOI: 10.5194/amt-9-2947-2016. Adresse: https://www.atmos-meas-tech.net/9/2947/2016/.

[2] G. Lufft Mess- und Regeltechnik GmbH. (2019). Lufft CHM 15k "NIMBUS" Produktwebsite, Adresse: https://www.lufft.com/products/cloud-height-snow-depth-sensors-288/ceilometer-chm-15k-nimbus-2300/ (besucht am 15. 03. 2019).

[3] R. Rew und G. Davis, "NetCDF: an interface for scientific data access", *IEEE Computer Graphics and Applications*, Jg. 10, Nr. 4, S. 76–82, Juli 1990, ISSN: 0272-1716. DOI: 10.1109/38.56302.

[4] NetCDF Developer Community. (Nov. 2018). Network Common Data Form (NetCDF): Documentation. Version 4.6.2, Adresse: https://www.unidata.ucar.edu/software/netcdf/docs/index.html (besucht am 15. 03. 2019).

[5] M. Wiegner und A. Geiß, "Aerosol profiling with the Jenoptik ceilometer CHM15kx", *Atmospheric Measurement Techniques*, Jg. 5, Nr. 8, S. 1953–1964, 2012. DOI: 10.5194/amt-5-1953-2012. Adresse: https://www.atmos-meas-tech.net/5/1953/2012/.

[6] L. van der Maaten, "Learning a Parametric Embedding by Preserving Local Structure", in *Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics*, D. van Dyk und M. Welling, Hrsg., Ser. Proceedings of Machine Learning Research, Bd. 5, Hilton Clearwater Beach Resort, Clearwater Beach, Florida USA: PMLR, Apr. 2009, S. 384–391. Adresse: http://proceedings.mlr.press/v5/maaten09a.html.

[7] D. H. Ackley, G. E. Hinton und T. J. Sejnowski, "A Learning Algorithm for Boltzmann Machines", *Cognitive Science*, Jg. 9, Nr. 1, S. 147–169, 1985. DOI: 10.1207/s15516709cog0901_7. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1207/s15516709cog0901_7. Adresse: https://onlinelibrary.wiley.com/doi/abs/10.1207/s15516709cog0901_7.

[8] Y. LeCun, L. Bottou, G. B. Orr und K.-R. Müller, "Efficient BackProp", in *Neural Networks: Tricks of the Trade, This Book is an Outgrowth of a 1996 NIPS Workshop*, London, UK, UK: Springer-Verlag, 1998, S. 9–50, ISBN: 3-540-65311-2. Adresse: http://dl.acm.org/citation.cfm?id=645754.668382.

[9] K. K. Pal und K. S. Sudeep, "Preprocessing for image classification by convolutional neural networks", in *2016 IEEE International Conference on Recent Trends in Electronics, Information Communication Technology (RTEICT)*, Mai 2016, S. 1778–1781. DOI: 10.1109/RTEICT.2016.7808140.

[10] D. E. Rumelhart, G. E. Hinton und R. J. Williams, "Learning Internal Representation by Error Propagation", in. Jan. 1986, Bd. Vol. 1.

[11] R. Bellman, *Dynamic Programming*. Dover Publications, 1957, ISBN: 9780486428093.

[12] K. P. F.R.S., "LIII. On lines and planes of closest fit to systems of points in space", *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, Jg. 2, Nr. 11, S. 559–572, 1901. DOI: 10.1080/14786440109462720. eprint: https://doi.org/10.1080/14786440109462720. Adresse: https://doi.org/10.1080/14786440109462720.

[13] Y. Lecun, L. Bottou, Y. Bengio und P. Haffner, "Gradient-based learning applied to document recognition", *Proceedings of the IEEE*, Jg. 86, Nr. 11, S. 2278–2324, Nov. 1998, ISSN: 0018-9219. DOI: 10.1109/5.726791.

[14] Y. LeCun und Y. Bengio, "The Handbook of Brain Theory and Neural Networks", in, M. A. Arbib, Hrsg., Cambridge, MA, USA: MIT Press, 1998, Kap. Convolutional Networks for Images, Speech, and Time Series, S. 255–258, ISBN: 0-262-51102-9. Adresse: http://dl.acm.org/citation.cfm?id=303568.303704.

[15] F.-F. Li, J. Johnson und S. Yeung. (März 2019). Stanford CS class CS231n: Convolutional Neural Networks for Visual Recognition, Adresse: https://cs231n.github.io/ (besucht am 25. 03. 2019).

[16] J. Xie, R. B. Girshick und A. Farhadi, "Unsupervised Deep Embedding for Clustering Analysis", in *ICML*, 2016.

[17] X. Guo, X. Liu, E. Zhu und J. Yin, "Deep Clustering with Convolutional Autoencoders", S. 373–382, Okt. 2017. DOI: 10.1007/978-3-319-70096-0_39.

[18] J. MacQueen, "Some methods for classification and analysis of multivariate observations.", English, 1967.

[19] D. Arthur und S. Vassilvitskii, "K-means++: The Advantages of Careful Seeding", in *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, Ser. SODA '07, New Orleans, Louisiana: Society for Industrial und Applied Mathematics, 2007, S. 1027–1035, ISBN: 978-0-898716-24-5. Adresse: http://dl.acm.org/citation.cfm?id=1283383.1283494.

[20] M. Ester, H.-P. Kriegel, J. Sander und X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise", AAAI Press, 1996, S. 226–231.

[21] Student, "The Probable Error of a Mean", *Biometrika*, Jg. 6, Nr. 1, S. 1–25, März 1908, ISSN: 0006-3444. DOI: 10.1093/biomet/6.1.1. eprint: http://oup.prod.sis.lan/biomet/article-pdf/6/1/1/605641/6-1-1.pdf. Adresse: https://doi.org/10.1093/biomet/6.1.1.

[22] L. van der Maaten und G. Hinton, "Visualizing Data using t-SNE", *Journal of Machine Learning Research*, Jg. 9, S. 2579–2605, 2008. Adresse: http://www.jmlr.org/papers/v9/vandermaaten08a.html.

[23] D. J. C. MacKay, *Information Theory, Inference & Learning Algorithms*. New York, NY, USA: Cambridge University Press, 2002, ISBN: 0521642981.

[24] V. Dumoulin und F. Visin, *A guide to convolution arithmetic for deep learning*, 2016. eprint: arXiv:1603.07285.

[25] A. Odena, V. Dumoulin und C. Olah, "Deconvolution and Checkerboard Artifacts", *Distill*, 2016. DOI: 10.23915/distill.00003. Adresse: http://distill.pub/2016/deconv-checkerboard.

[26] Y. LeCun und C. Cortes, "MNIST handwritten digit database", 2010. Adresse: http://yann.lecun.com/exdb/mnist/.

[27] L. Lu, Y. Shin, Y. Su und G. Karniadakis, "Dying ReLU and Initialization: Theory and Numerical Examples", März 2019.

[28] T. Kodinariya und P. Dan Makwana, "Review on Determining of Cluster in K-means Clustering", *International Journal of Advance Research in Computer Science and Management Studies*, Jg. 1, S. 90–95, Jan. 2013.

[29] C. A. Sugar und G. M. James, "Finding the Number of Clusters in a Dataset", *Journal of the American Statistical Association*, Jg. 98, Nr. 463, S. 750–763, 2003. DOI: 10.1198/016214503000000666. eprint: https://doi.org/10.1198/016214503000000666. Adresse: https://doi.org/10.1198/016214503000000666.

[30] P. Rousseeuw, "Silhouettes: A Graphical Aid to the Interpretation and Validation of Cluster Analysis", *J. Comput. Appl. Math.*, Jg. 20, Nr. 1, S. 53–65, Nov. 1987, ISSN: 0377-0427. DOI: 10.1016/0377-0427(87)90125-7. Adresse: http://dx.doi.org/10.1016/0377-0427(87)90125-7.

[31] B. Settles, "Active Learning Literature Survey", *Computer Sciences Technical Report 1648 - University of Wisconsin–Madison*, 2009. Adresse: http://burrsettles.com/pub/settles.activelearning.pdf.

A Anhang

A.1 Benötigte Programmbibliotheken

Die Implementation nutzt die im Quellcode A.1 aufgelisteten Python Programmbibliotheken. Diese können direkt mithilfe von `pip`[1] aus der beiliegenden `requirements.txt` eingelesen werden.

```
absl-py==0.7.0
astor==0.7.1
cftime==1.0.3.4
cycler==0.10.0
gast==0.2.2
grpcio==1.19.0
h5py==2.9.0
Keras==2.2.4
Keras-Applications==1.0.7
Keras-Preprocessing==1.0.9
kiwisolver==1.0.1
Markdown==3.0.1
matplotlib==3.0.3
mock==2.0.0
netCDF4==1.4.3.2
numpy==1.16.2
opencv-python==4.0.0.21
pandas==0.24.1
pbr==5.1.3
protobuf==3.7.0
pydot==1.4.1
pyparsing==2.3.1
python-dateutil==2.8.0
pytz==2018.9
PyYAML==5.1
scikit-learn==0.20.3
```

[1] https://pypi.org/project/pip/

```
scipy==1.2.1
six==1.12.0
sklearn==0.0
tensorboard==1.13.1
tensorflow==1.13.1
tensorflow-estimator==1.13.0
termcolor==1.1.0
Werkzeug==0.14.1
```

Quellcode A.1: Benötigte Python Bibliotheken (`requirements.txt`)

A.2 Klassifikation von MNIST Ziffern

A.2.1 Autoencoder Model

Layer (type)	Output Shape	Param #
input (InputLayer)	(None, 256, 8, 1)	0
conv2d_1 (Conv2D)	(None, 256, 8, 16)	416
max_pooling2d_1 (MaxPooling2D)	(None, 128, 4, 16)	0
conv2d_2 (Conv2D)	(None, 128, 4, 32)	4640
max_pooling2d_2 (MaxPooling2D)	(None, 64, 2, 32)	0
conv2d_3 (Conv2D)	(None, 64, 2, 64)	18496
max_pooling2d_3 (MaxPooling2D)	(None, 32, 1, 64)	0
flatten_1 (Flatten)	(None, 2048)	0
encoded (Dense)	(None, 10)	20490
dense_1 (Dense)	(None, 2048)	22528
reshape_1 (Reshape)	(None, 32, 1, 64)	0
conv2d_4 (Conv2D)	(None, 32, 1, 64)	36928
resize_layer_1 (ResizeLayer)	(None, 64, 2, 64)	0
conv2d_5 (Conv2D)	(None, 64, 2, 32)	18464
resize_layer_2 (ResizeLayer)	(None, 128, 4, 32)	0
conv2d_6 (Conv2D)	(None, 128, 4, 16)	4624
resize_layer_3 (ResizeLayer)	(None, 256, 8, 16)	0
output (Conv2D)	(None, 256, 8, 1)	401

```
Total params: 126,987
Trainable params: 126,987
Non-trainable params: 0
_________________________________________________________________
```

Quellcode A.2: Autoencoder Model zur Klassifikation von MNIST Ziffern

A.2.2 Clustering-Autoencoder Model

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input (InputLayer)           (None, 256, 8, 1)         0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 256, 8, 16)        416
_________________________________________________________________
max_pooling2d_1 (MaxPooling2D) (None, 128, 4, 16)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 128, 4, 32)        4640
_________________________________________________________________
max_pooling2d_2 (MaxPooling2D) (None, 64, 2, 32)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 64, 2, 64)         18496
_________________________________________________________________
max_pooling2d_3 (MaxPooling2D) (None, 32, 1, 64)       0
_________________________________________________________________
flatten_1 (Flatten)          (None, 2048)              0
_________________________________________________________________
encoded (Dense)              (None, 10)                20490
_________________________________________________________________
dense_1 (Dense)              (None, 2048)              22528
_________________________________________________________________
reshape_1 (Reshape)          (None, 32, 1, 64)         0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 32, 1, 64)         36928
_________________________________________________________________
resize_layer_1 (ResizeLayer) (None, 64, 2, 64)         0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 64, 2, 32)         18464
_________________________________________________________________
resize_layer_2 (ResizeLayer) (None, 128, 4, 32)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 128, 4, 16)        4624
_________________________________________________________________
```

```
resize_layer_3 (ResizeLayer)     (None, 256, 8, 16)    0
_________________________________________________________________
cluster (ClusteringLayer)        (None, 10)            100
_________________________________________________________________
output (Conv2D)                  (None, 256, 8, 1)     401
=================================================================
Total params: 127,087
Trainable params: 127,087
Non-trainable params: 0
_________________________________________________________________
```

Quellcode A.3: Clustering-Autoencoder Model zur Klassifikation von MNIST Ziffern

A.2.3 Clustering-Beispiele

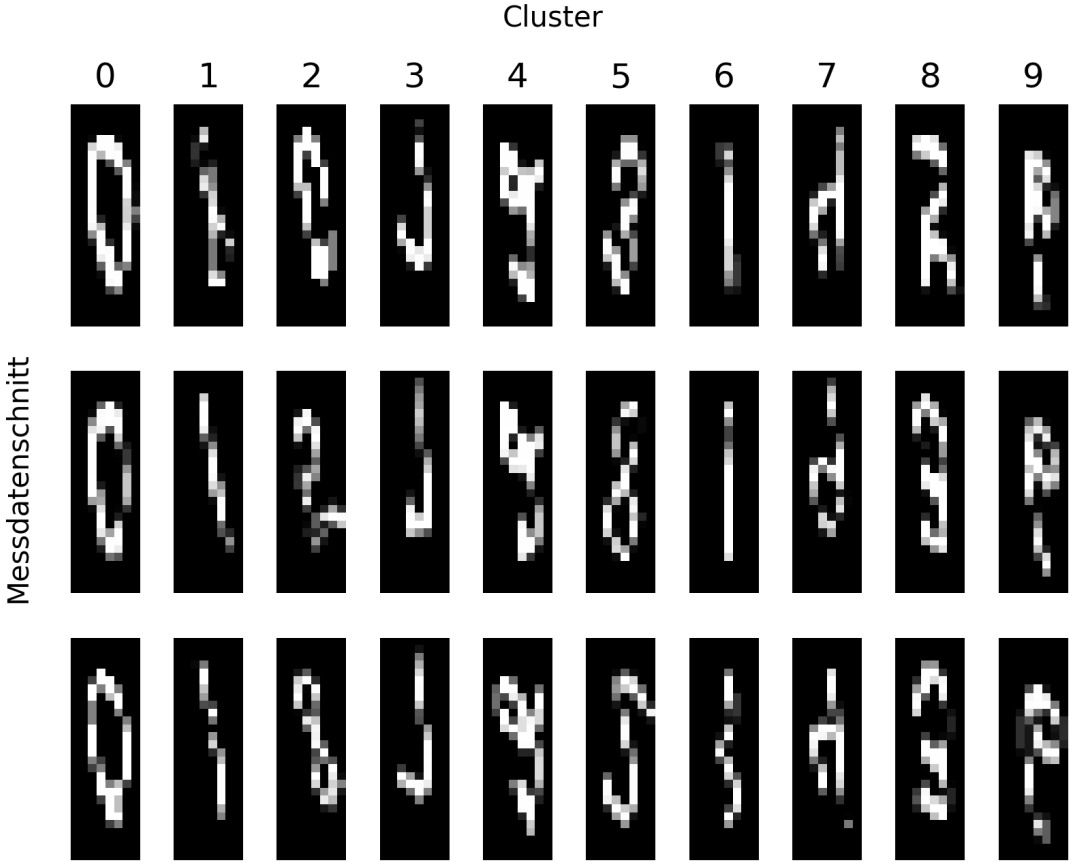


Abbildung A.1: MNIST-Ziffern mit der höchsten Clusterzugehörigkeit nach k-Means Initialisierung

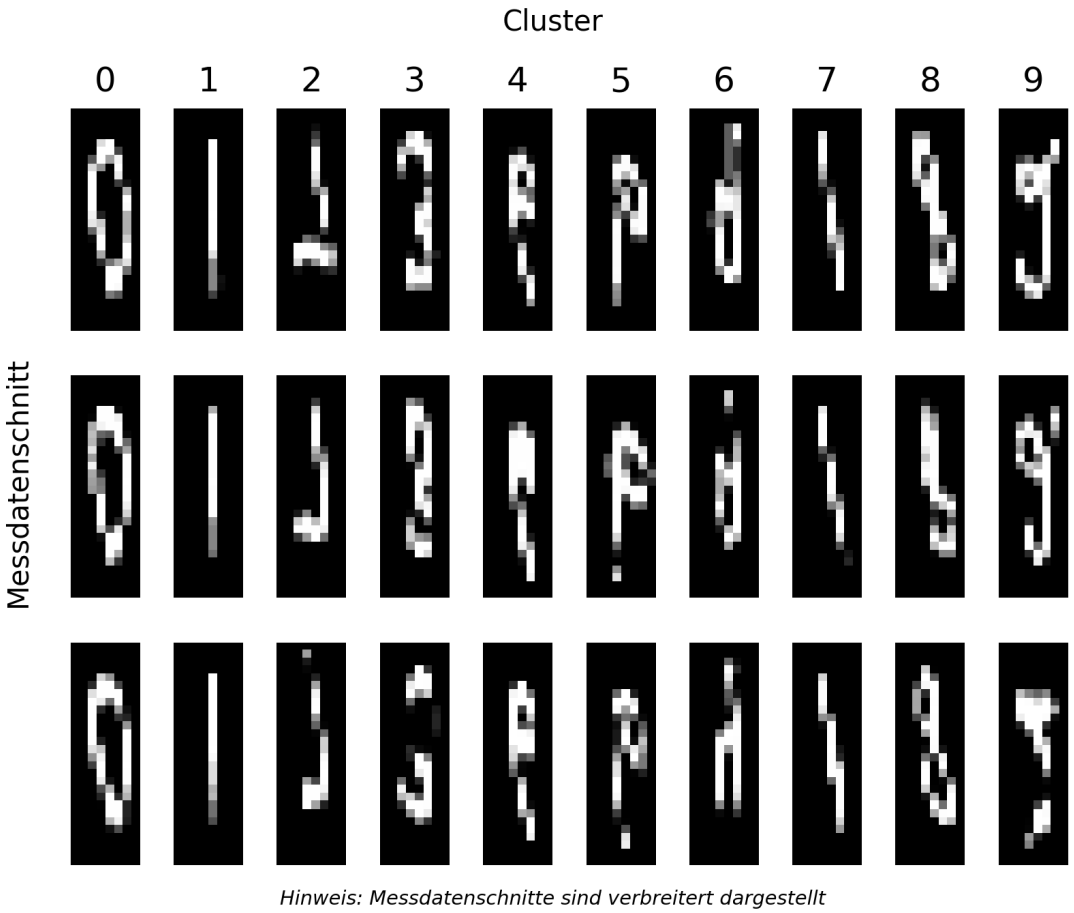


Abbildung A.2: MNIST-Ziffern mit der höchsten Clusterzugehörigkeit nach Training

A.3 Clustering von Aerosol-Rückstreuprofilen

A.3.1 Normalisierungsparameter

Variable	Wert	Beschreibung
`max`	5.657 e−06 $m^{-1}sr^{-1}$	Globale maximale Rückstreuintensität
`min`	−8.944 e−07 $m^{-1}sr^{-1}$	Globale minimale Rückstreuintensität

Tabelle A.1: Genutzte Normalisierungsparameter

A.3.2 Autoencoder Model

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input (InputLayer)           (None, 256, 8, 1)         0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 256, 8, 16)        416
_________________________________________________________________
max_pooling2d_1 (MaxPooling2D) (None, 128, 4, 16)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 128, 4, 32)        4640
_________________________________________________________________
max_pooling2d_2 (MaxPooling2D) (None, 64, 2, 32)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 64, 2, 64)         18496
_________________________________________________________________
max_pooling2d_3 (MaxPooling2D) (None, 32, 1, 64)       0
_________________________________________________________________
flatten_1 (Flatten)          (None, 2048)              0
_________________________________________________________________
encoded (Dense)              (None, 32)                65568
_________________________________________________________________
dense_1 (Dense)              (None, 2048)              67584
_________________________________________________________________
reshape_1 (Reshape)          (None, 32, 1, 64)         0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 32, 1, 64)         36928
_________________________________________________________________
resize_layer_1 (ResizeLayer) (None, 64, 2, 64)         0
```

```
29 _________________________________________________________________
30 conv2d_5 (Conv2D)              (None, 64, 2, 32)     18464
31 _________________________________________________________________
32 resize_layer_2 (ResizeLayer)  (None, 128, 4, 32)    0
33 _________________________________________________________________
34 conv2d_6 (Conv2D)              (None, 128, 4, 16)    4624
35 _________________________________________________________________
36 resize_layer_3 (ResizeLayer)  (None, 256, 8, 16)    0
37 _________________________________________________________________
38 output (Conv2D)                (None, 256, 8, 1)     401
39 =================================================================
40 Total params: 217,121
41 Trainable params: 217,121
42 Non-trainable params: 0
43 _________________________________________________________________
```

Quellcode A.4: Autoencoder Model zum Clustering von Ceilometerdaten

A.3.3 Clustering-Autoencoder Model

```
1  _________________________________________________________________
2  Layer (type)                 Output Shape              Param #
3  =================================================================
4  input (InputLayer)           (None, 256, 8, 1)         0
5  _________________________________________________________________
6  conv2d_1 (Conv2D)            (None, 256, 8, 16)        416
7  _________________________________________________________________
8  max_pooling2d_1 (MaxPooling2D) (None, 128, 4, 16)      0
9  _________________________________________________________________
10 conv2d_2 (Conv2D)            (None, 128, 4, 32)        4640
11 _________________________________________________________________
12 max_pooling2d_2 (MaxPooling2D) (None, 64, 2, 32)      0
13 _________________________________________________________________
14 conv2d_3 (Conv2D)            (None, 64, 2, 64)         18496
15 _________________________________________________________________
16 max_pooling2d_3 (MaxPooling2D) (None, 32, 1, 64)      0
17 _________________________________________________________________
18 flatten_1 (Flatten)          (None, 2048)              0
19 _________________________________________________________________
20 encoded (Dense)              (None, 32)                65568
21 _________________________________________________________________
22 dense_1 (Dense)              (None, 2048)              67584
23 _________________________________________________________________
24 reshape_1 (Reshape)          (None, 32, 1, 64)         0
```

```
25 _________________________________________________________________
26 conv2d_4 (Conv2D)                (None, 32, 1, 64)     36928
27 _________________________________________________________________
28 resize_layer_1 (ResizeLayer)     (None, 64, 2, 64)     0
29 _________________________________________________________________
30 conv2d_5 (Conv2D)                (None, 64, 2, 32)     18464
31 _________________________________________________________________
32 resize_layer_2 (ResizeLayer)     (None, 128, 4, 32)    0
33 _________________________________________________________________
34 conv2d_6 (Conv2D)                (None, 128, 4, 16)    4624
35 _________________________________________________________________
36 resize_layer_3 (ResizeLayer)     (None, 256, 8, 16)    0
37 _________________________________________________________________
38 cluster (ClusteringLayer)        (None, 8)             256
39 _________________________________________________________________
40 output (Conv2D)                  (None, 256, 8, 1)     401
41 =================================================================
42 Total params: 217,377
43 Trainable params: 217,377
44 Non-trainable params: 0
45 _________________________________________________________________
```

Quellcode A.5: Clustering-Autoencoder Model zum Clustering von Ceilometerdaten

A.3.4 Clustering-Beispiele

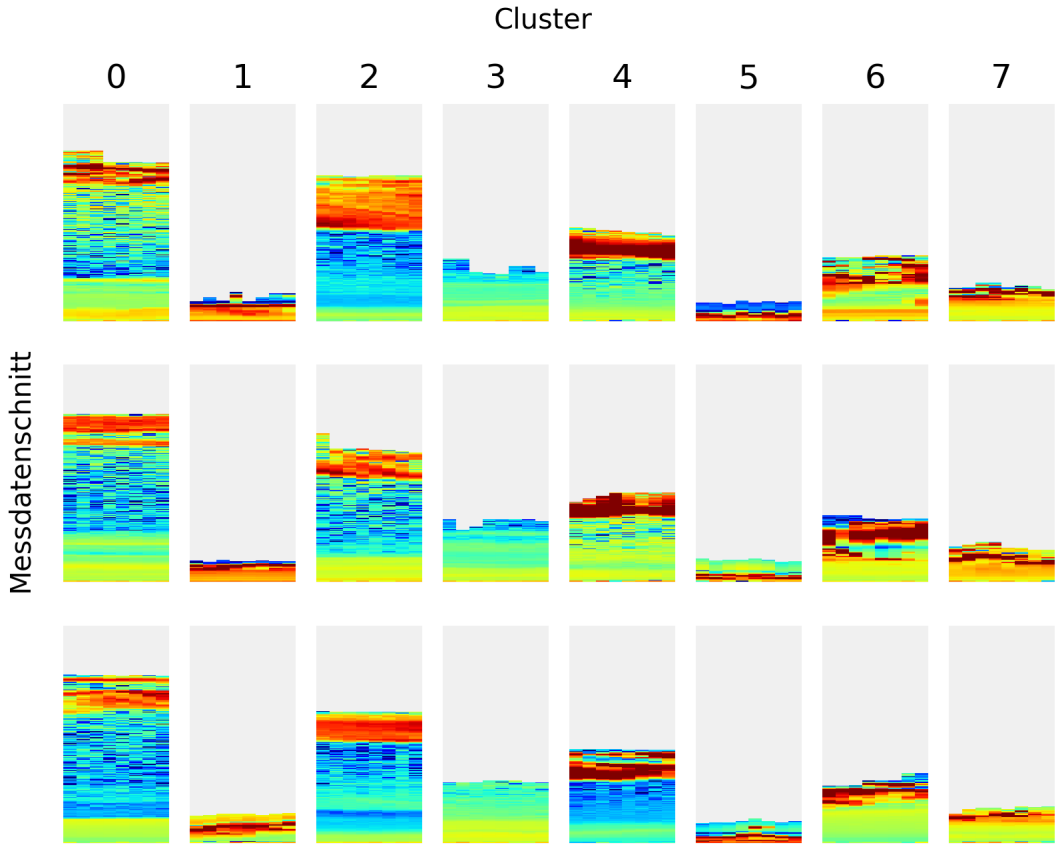


Abbildung A.3: Ceilometer-Messdatenschnitte mit der höchsten Clusterzugehörigkeit nach k-Means Initialisierung

A.4 Aktivierungsfunktionen

Logistische Funktion (Sigmoid):

$$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}} \tag{A.1a}$$

$$f'(x) = \sigma(x)(1 - \sigma(x)) \tag{A.1b}$$

Rectified Linear Unit (ReLU):

$$f(x) = \begin{cases} 0 & \text{für } x < 0 \\ x & \text{für } x \geq 0 \end{cases} \tag{A.2a}$$

$$f'(x) = \begin{cases} 0 & \text{für } x < 0 \\ 1 & \text{für } x \geq 0 \end{cases} \tag{A.2b}$$

Exponential Linear Unit (ELU):

$$f(\alpha, x) = \begin{cases} \alpha(e^x - 1) & \text{für } x \leq 0 \\ x & \text{für } x > 0 \end{cases} \tag{A.3a}$$

$$f'(\alpha, x) = \begin{cases} f(\alpha, x) + \alpha & \text{für } x \leq 0 \\ 1 & \text{für } x > 0 \end{cases} \tag{A.3b}$$

Glossar

Autoencoder Künstliches neuronales Netz, welches unüberwacht eine nichtlineare Abbildung von Eingangsdaten auf komprimierte Merkmalsvektoren lernt.

Ceilometer Gerät zur optischen Bestimmung der Wolkenhöhe. Je nach Typ ist auch die Bestimmung weiterer Parameter möglich.

Clusteranalyse Verfahren zur Gruppierung von Objekten mit ähnlichen Eigenschaften.

Clustering-Layer Layer in einem neuronalen Netz, welcher eine Wahrscheinlichkeitsverteilung über die Clusterzugehörigkeit eines Datensatzes ausgibt.

Hyperparameter Parameter eines neuronalen Netzes, welche nicht gelernt werden, sondern vor dem Training manuell gewählt werden müssen.

LIDAR LIDAR (engl. *light detection and ranging*) ist eine optische Methode zur Messung von Abständen, Geschwindigkeiten sowie atmosphärischer Parameter.

Messdatenschnitt Für die Datenverarbeitung vorbereiteter Teilausschnitt eines Messdatensatzes.

Rückstreuprofil Die Menge mehrerer von einem LIDAR-Ceilometer gemessenen Rückstreuintensitäten über alle Höhenbereiche einer oder mehrerer Messungen.

Troposphäre Unterste Schicht der Erdatmosphäre vom Erdboden bis zum Beginn der Stratosphäre. Durchschnittlich 15 km hoch..

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Gemäß der Allgemeinen Prüfungs- und Studienordnung ist zusammen mit der Abschlussarbeit eine schriftliche Erklärung abzugeben, in der der Studierende bestätigt, dass die Abschlussarbeit „– bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit [(§ 18 Abs. 1 APSO-TI-BM bzw. § 21 Abs. 1 APSO-INGI)] – ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt wurden. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen."

Quelle: § 16 Abs. 5 APSO-TI-BM bzw. § 15 Abs. 6 APSO-INGI

Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: _____

Vorname: _____

dass ich die vorliegende Bachelorarbeit – bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit – mit dem Thema:

Merkmalsextraktion durch Methoden des unüberwachten maschinellen Lernens zur Klassifikation von Aerosol-Rückstreuprofilen aus LIDAR-Ceilometern

ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

_____	_____	_____
Ort	Datum	Unterschrift im Original