

**Segmentierung der Marsoberfläche mit Hilfe  
von unüberwachtem tiefem Clustering**

**Bachelorarbeit**

**Merlin Scholz  
26. März 2022**

Supervisors:

Prof. Dr.-Ing. Gernot A. Fink

Dominik Kossmann, M.Sc.

Fakultät für Informatik  
Technische Universität Dortmund  
<http://www.cs.tu-dortmund.de>



# INHALTSVERZEICHNIS

---

1	EINLEITUNG	5
1.1	Motivation	5
1.2	Rahmen der Arbeit	6
2	GRUNDLAGEN	7
2.1	Analyse der Marsoberfläche	7
2.1.1	Merkmale der Marsoberfläche	7
2.1.2	Remote Sensing Instruments	9
2.2	Convolutional Neural Networks	10
2.2.1	Konvolution	12
2.2.2	Convolutional Layer	13
2.2.3	Activation Layer	15
2.2.4	Pooling Layer	16
2.2.5	Fully Connected Layer	18
2.2.6	Batch Normalization	18
2.2.7	Loss Function	18
2.2.8	Gradientenverfahren	20
2.2.9	Backpropagation	23
3	VERWANDTE ARBEITEN	27
3.1	Bildsegmentierung	27
3.1.1	DEC – Deep Embedded Clustering	27
3.1.2	Bildsegmentierung auf Basis eines Bildclusterings	29
3.2	Bildclustering	30
3.2.1	Texturbasiertes Clustering	31
3.3	Kratererkennung	34
3.3.1	Kratererkennung über Neuronale Netze	35
4	METHODIK	37
4.1	Funktionsweise der Segmentierung	37
4.2	Eigenschaften der Initialisierung	41
4.2.1	Filterbänke	43
4.2.2	Größe der Filter	43
4.2.3	Gewichtungen der Parameter	43
4.2.4	Anzahl der Cluster	44
4.3	Eigenschaften des Segmentierungsalgorithmus	44

4.4	Netzwerkarchitektur	44	
4.4.1	Abbruchkriterium	44	
4.4.2	Aktivierungsfunktionen	45	
4.4.3	Pooling Layer	45	
4.4.4	Fully Connected Layers	46	
4.4.5	Anzahl der Konvolutionsschichten	46	
4.5	Anpassungen zur Segmentierung von mehrfarbigen Fotografien	46	
5	EXPERIMENTE	49	
5.1	Datensätze	49	
5.1.1	Domänenspezifisch	49	
5.1.2	Allgemein	50	
5.2	Modifizierungen der Initialisierung	50	
5.2.1	Filterbänke	50	
5.2.2	Größe der Filter	53	
5.2.3	Gewichtung der Parameter	57	
5.2.4	Anzahl der Cluster	59	
5.3	Modifizierungen der Netzwerkarchitektur	61	
5.3.1	Abbruchkriterium	61	
5.3.2	Aktivierungsfunktionen	63	
5.3.3	Pooling Layer	67	
5.3.4	Fully Connected Layers	67	
5.3.5	Anzahl der Konvolutionsschichten	70	
5.3.6	Anpassungen zur Segmentierung von mehrfarbigen Fotografien	71	
5.4	Metriken	72	
5.4.1	Domänenspezifisch	72	
5.4.2	Allgemein	74	
5.5	Evaluation	77	
5.5.1	Domänenspezifisch	77	
5.5.2	Allgemein	77	
5.6	Vergleich	77	
5.6.1	Domänenspezifisch	77	
5.6.2	Allgemein	79	
5.7	Diskussion	81	
5.7.1	Domänenspezifisch	81	
5.7.2	Allgemein	81	
5.7.3	Weitere Anwendungsgebiete	82	

6	FAZIT	83	
6.1	Zusammenfassung	83	
6.2	Fazit	84	
6.3	Zukünftige Arbeiten	86	
A	GRAPHEN	87	
A.1	Graphen zu Unterabschnitt 5.3.1 „Abbruchkriterium“	87	



## EINLEITUNG

---

### 1.1 MOTIVATION

Neuronale Netzwerke werden in den letzten Jahren häufig zur Objekterkennung und Bildsegmentierung genutzt. Ein weit bekanntes Anwendungsbeispiel findet sich in der Entwicklung des autonomen Fahrens, bei welcher das Bild in verschiedene Klassen wie z. B. Fahrbahn, Ampel oder Passant segmentiert wird. Dies geschieht für gewöhnlich durch überwachtes Lernen, also unter der Voraussetzung einer Ground Truth, aus denen das Netzwerk lernen kann, wie es die jeweiligen Klassen erkennen kann.

Diese Ground Truths werden meist von Hand erschaffen. So können diese im Straßenverkehr von jedermann erstellt werden, während in spezifischeren Anwendungsgebieten meist Domänenexperten dazu benötigt werden. Dies ist auch der Fall bei der Analyse der Marsoberfläche. Aufgrund der schlichten Größe des Datensatzes, und der Notwendigkeit von Experten in diesem Fachgebiet, ist es hier weder leicht, noch zeiteffizient oder kostengünstig möglich, manuell eine Ground Truth zu entwickeln.

Es besteht zwar die Möglichkeit bereits vorhandene Ground Truths zu nutzen: So könnten bspw. Aufnahmen der (stärker erkundeten) Mondoberfläche oder gar Erdaufnahmen angepasst werden, um als Trainingsdatensatz zur Analyse der Marsoberfläche zu dienen. Dies wirft allerdings neue Probleme auf. Bei bspw. sich stark verändernden Lichtverhältnissen, Aufnahmewinkeln, o. Ä. kann ein vortrainiertes Neuronales Netz meistens keine optimalen Ergebnisse erzielen, eben weil es nicht auf diese neuen Rahmenbedingungen trainiert wurde.

Neben dem überwachten Lernen existiert auch das unüberwachte Lernen. Dieses arbeitet ohne Ground Truth, stattdessen muss dem Algorithmus eine andere Methode übermittelt werden, anhand welcher er erkennen kann, wie korrekt ein Ergebnis ist. Dies geschieht z. B. über einen Algorithmus, der das Problem zwar ohne neuronale Netze löst, allerdings keine so hohe Genauigkeit besitzt.

## 1.2 RAHMEN DER ARBEIT

In dieser Arbeit wird versucht, das Problem der Segmentierung der Marsoberfläche durch unüberwachtes Lernen zu lösen. Begonnen wird mit der Implementierung eines einfachen Algorithmus zur Segmentierung auf Basis des Ansatzes von nach [Kan18], welcher klassische Clusteringalgorithmen zur Initialisierung nutzt, die anschließend durch ein neuronales Netzwerk weiter verfeinert werden (vgl. Unterabschnitt 3.1.2). Dieser Prozess wird über verschiedene Ansätze, wie z. B. das Ersetzen der Initialisierungsmethode, die Veränderung der Netzwerkarchitektur, oder die Analyse des Einflusses bestimmter Hyperparameter weiter optimiert. Anschließend werden diese Ergebnisse miteinander verglichen, und die beste Kombination von ihnen ausgewählt, um mit alternativen Algorithmen zur Bildsegmentierung – im Allgemeinen oder domänenspezifisch – verglichen zu werden.



## GRUNDLAGEN

---

### 2.1 ANALYSE DER MARSOBERFLÄCHE

#### 2.1.1 Merkmale der Marsoberfläche

Auf dem Mars ist eine Vielzahl an unterschiedlichen Oberflächenmerkmalen vorhanden. Diese wurden erstmals im Jahre 1964 fotografisch während der Mariner 4-Mission der NASA aufgenommen. [Sid18] Seitdem wurden durch den Fortschritt der Technik innerhalb verschiedener Mars-Missionen Aufnahmen mit immer höheren Auflösungen erfasst. Heutzutage kann die Marsoberfläche sogar ohne explizite Mars-Mission erfasst werden, z. B. durch das Hubble-Weltraumteleskop.

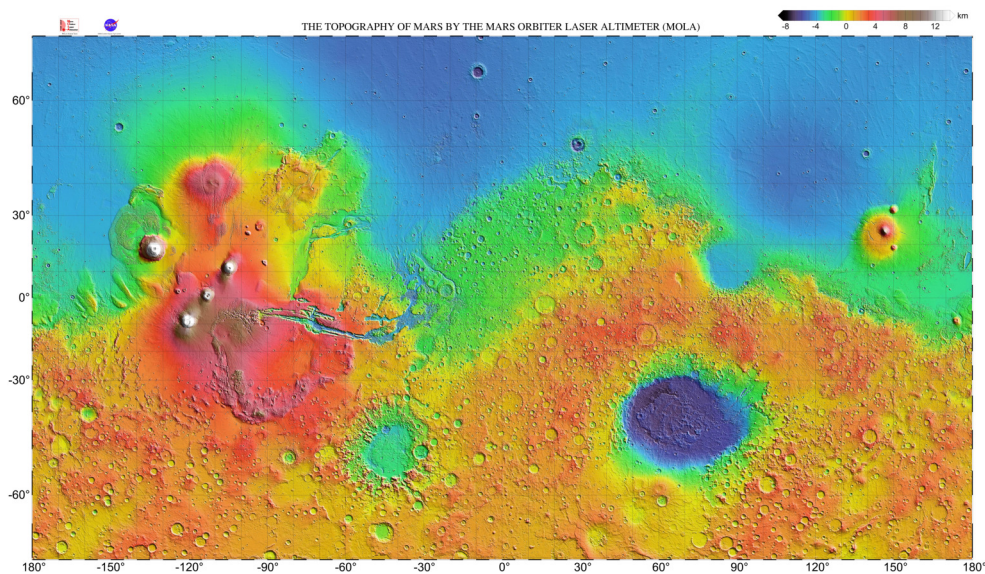


Abbildung 2.1.1: Topografie des Mars, aus [NAS00]. In der blau-grün gefärbten, tieferen Region sind wenige Höhenunterschiede sichtbar, während in der südlichen Hemisphäre deutliche lokale Höhenunterschiede erkennbar sind.

Die Marsoberfläche wird oft in zwei Terrains unterteilt, welche anhand einer topografischen Karte gut sichtbar sind (vgl. Abbildung 2.1.1). Die nördlichen Hemisphäre besitzt eine wüstenartige Oberfläche: Hier befinden sich vergleichsweise wenige Krater und andere Oberflächenmerkmale. Radarsonden-Analysen dieser Regionen zeigen allerdings, dass unter dieser merkmalsarmen Oberfläche mehrere kreisförmige Strukturen zu erkennen sind. Dies deutet darauf hin, dass dieser Teil der Oberfläche überdeckt wurde, möglicherweise als Folge eines vergleichsweise großen Einschlages oder eines endogenen Prozesses. [Gre13, Kap. 7]

Auf der südlichen Hemisphäre hingegen sind vergleichsweise viele Merkmale sichtbar. Ihre Oberfläche wurde durch unterschiedliche Prozesse geprägt, welche wahrscheinlich heutzutage noch aktiv sind. Auf ihr sind allerdings relativ wenige Einschlagskrater sichtbar, dies deutet auf ein junges Alter hin. [Gre13, Kap. 7]

Die wohl signifikantesten Merkmale der Oberfläche sind u. a.: [Gre13, Kap. 7]

**EINSCHLAGSKRATER** Einschlagskrater können sich stark in der Größe unterscheiden: So haben die größten Exemplare einen Durchmesser von bis zu 1800 km, während auch eine Großzahl an sub-km Kratern existieren. Diese Eigenschaft erschwert die zuverlässige Erkennung erheblich, da die angewandte Methode in der Lage sein muss, die zur Erkennung genutzten Merkmale stark in der Größe zu variieren. Es existieren zwar verschiedene Variationen, die meisten Krater gleichen aber einfachen kreisförmigen Vertiefungen, wie in Abbildung 2.1.2a dargestellt.

**VULKANARTIGE MERKMALE** Mehr als die Hälfte der Marsoberfläche ist von Vulkanen bedeckt. Es existieren verschiedene Arten von Vulkanen mit jeweils unterschiedlichen Merkmalsausprägungen (vgl. Abbildung 2.1.2b und Abbildung 2.1.2c). [Gre13, Kap. 7]

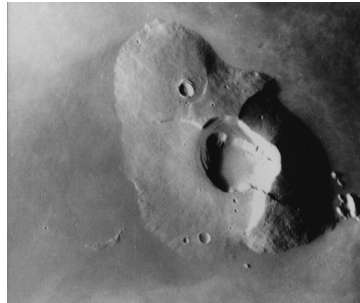
**TEKTONISCHE MERKMALE** Tektonische Veränderungen der Marsoberfläche können verschiedene Ursachen haben, wie z. B. das Auftreten von Vulkanen, Einschlägen oder anderen Verformungen der Lithosphäre. [Gre13, Kap. 7] Hierbei wird oft zwischen ausdehnenden und komprimierenden tektonischen Prozessen unterschieden. Während ausdehnende Veränderungen zu Schluchten bzw. Gräben führen (vgl. Abbildung 2.1.2d), werden durch die Komprimierung Hügelketten erschaffen.

**OBERFLÄCHENBESCHAFFENHEIT** Auch ohne Einwirkungen von Kratern, Vulkanen, etc. weist die Marsoberfläche unterschiedliche Beschaffenheiten auf. So sind – insbesondere an den Polen – oft Gletscher zu finden (vgl. Abbildung 2.1.2e). Des Weiteren

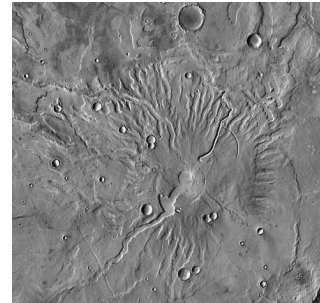
kann sich diese Beschaffenheit auch durch Erosionen verändern bzw. verändert haben, sowohl durch Wind als auch durch Wasser.



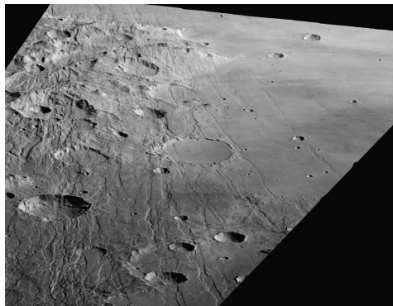
(a) Beispiel für einen Krater



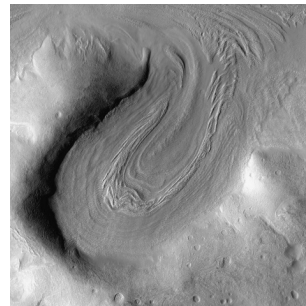
(b) Beispiel für einen (ehemaligen) Vulkan



(c) Beispiel für einen (ehemaligen) Vulkan mit strahlenförmigen Ausbuchtungen



(d) Beispiel für mehrere Gräben



(e) Beispiel für einen Gletscher

Abbildung 2.1.2: Beispiele für Strukturen auf der Marsoberfläche, alle aus [Gre13, Kap. 7]

Diese Faktoren sorgen für eine große Varietät an optisch unterschiedlichen Oberflächen des Mars. Erkennbar sind sie auf Fotografien nur durch das flach einfallende Licht, welches Erhebungen und Täler durch Schatten und hellere Regionen sichtbar macht. Durch diesen Effekt werden auch die unterschiedlichen, zu analysierenden Texturen erkennbar.

### 2.1.2 Remote Sensing Instruments

Von der Marsoberfläche existieren verschiedenste Arten von Aufnahmen für unterschiedliche Analysen. Dabei sind gewisse Parameter, wie z. B. die aufgenommene Wellenlänge, den Winkel zur Oberfläche, die Auflösung, Brennweite, Ort der Aufnah-

me und viele weitere wichtig, damit ein Bild zur Analyse einer gewissen Eigenschaft geeignet ist.

Für die hier genutzten Anwendungsfälle eignen sich Instrumente gut, die sichtbares Licht aufnehmen. Obwohl Farbaufnahmen in vielerlei Hinsicht von Vorteil zur Analyse wären, sind diese nicht großflächig vorhanden, daher werden Graustufenaufnahmen als Kompromiss benutzt. Diese besitzen den Vorteil, dass nur ein Drittel des Speichers bei den Berechnungen benötigt wird, da statt separaten Rot-, Grün- und Blauwerten nur ein Helligkeitswert gespeichert und verarbeitet werden muss.

Des Weiteren sollten die Bilder eine vergleichsweise hohe Auflösung haben, damit auch kleinere Merkmale gut erkannt werden können. Dies stellt einen Konflikt mit einer weiteren Anforderung dar, da ein Datensatz gesucht wird, der einen Großteil der Oberfläche abdeckt.

Außerdem eignen sich Aufnahmen gut, die senkrecht zur Oberfläche entstanden sind, so dass Objekte auf dieser möglichst denselben Maßstab besitzen.

Unter Berücksichtigung dieser Faktoren ergeben sich u. a. zwei geeignete Instrumente: Die *Context Camera (CTX)* des *Mars Reconnaissance Orbiters* der NASA [MBIC<sup>+</sup>07] und das *Thermal Emission Imaging System (THEMIS)* der *Mars Odyssey* [SAB<sup>+</sup>04]. Letzteres fertigt zwar Infrarot-Aufnahmen an, für diese Aufnahmen existiert allerdings eine Kraterdatenbank, welche zur Evaluation dieses Algorithmus genutzt werden kann.

## 2.2 CONVOLUTIONAL NEURAL NETWORKS

Neuronale Netze werden oftmals als eine Weiterentwicklung oder Optimierung des maschinellen Lernens betrachtet: Man setzt auf mehrere Schichten, auch Layers genannt, von vergleichsweise einfachen linearen Funktionen um zur gewünschten Approximation zu kommen. [Har17]

Eine der einfachsten Formen eines neuronalen Netzes besteht aus dem Multilayer-Perceptron: Dieses besteht aus mehreren der zuvor erwähnten Layers, von denen jede mehrere einzelne Perceptronen enthält. Zwischen einer Eingabe- und einer Ausgabeschicht existieren somit auch noch eine beliebige Anzahl an sogenannten Hidden Layers.

In diesem Kontext wird ein Perceptron auch als Neuron bezeichnet. Ein typisches Neuron verarbeitet einen Eingabevektor indem es dies mit einem Gewichtungsvektor elementweise multipliziert und anschließend einen Bias-Vektor aufaddiert. Die Addition dieses Bias ist nötig um die gesamte Aktivierungsfunktion entlang der x-Achse verschieben zu können, da sonst manche gewünschten Ergebnisse nicht erreicht

werden können. [GP17, Kap. 2] Ein typisches Neuron ist in Abbildung 2.2.1 dargestellt. Nach dessen Durchlaufen werden die Elemente dieses Vektor aufaddiert, und auf diese Summe eine Aktivierungsfunktion (vgl. Unterabschnitt 2.2.3) angewandt. Während das Netzwerk trainiert wird, passen die einzelnen Perceptronen diese zwei Vektoren, also die Gewichtungen und die Bias, so an, dass die Ergebnisse weitgehend optimiert werden (vgl. Unterabschnitte 2.2.8 und 2.2.9). Diese Gewichtungen sind anfangs zufällig bestimmt. [Har17, LJY19]

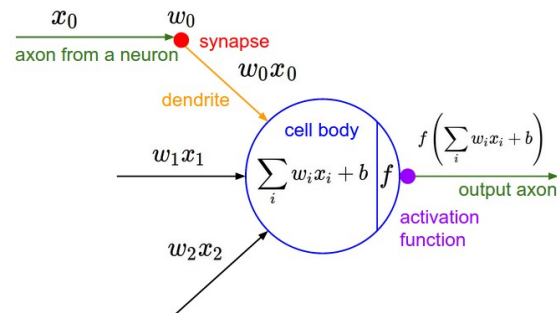


Abbildung 2.2.1: Funktionsweise eines Neurons, aus [LJY19]

Die Aneinanderkettung dieser Schichten führt dazu, dass zwischen ihnen eine immer abstraktere Form der Eingabedaten entsteht.

Convolutional Neural Networks beschreiben eine Teilmenge der neuronalen Netze, in der die jeweilige Netzwerkarchitektur mindestens eine Convolutional Layer (auch Faltungsschicht genannt, vgl. Unterabschnitt 2.2.1) enthält. Die Nutzung dieser Convolutional Layer zieht oft die Nutzung einer Pooling Layer (Unterabschnitt 2.2.4) mit sich. [GBC16]

Obwohl Convolutional Neural Networks theoretisch dazu geeignet sind die meisten Daten mit einer gitterähnlichen Struktur zu verarbeiten, erfreuen sie sich im Bereich der Bilddatenanalyse der größten Beliebtheit. [GBC16, Kap. 9] Eine wichtige Ursache für diese Beliebtheit liegt bspw. in dem Erfolg bei dem Bild-Klassifizierungswettbewerb ImageNet. Während dort im Jahr 2011 die Gewinnergruppe mit klassischen Klassifizierungsalgorithmen einen Top-5-Score von 74,3 % erzielt hat, wurden im Jahr 2012 mit einem CNN (AlexNet, [KSH12]) erstmals ein Wert von 83,6 % erreicht. Von diesem Punkt an wurde die Bestenliste der darauffolgenden Jahre durch Convolutional Neural Networks dominiert. [Fra18, Kap. 1]

Neben der genannten Objekterkennung eignen sie sich auch zur Bildsegmentierung. So werden sie bspw. erfolgreich in der Entwicklung von selbstfahrenden Autos eingesetzt. [KU19]

Die Architektur eines typischen, einfachen Convolutional Neural Networks ist wie folgt [LJY19]:

$$\text{INPUT} \rightarrow (\text{CONVOLUTIONAL} \rightarrow \text{ACTIVATION} \rightarrow \text{POOLING})^n \\ \rightarrow \text{FULLYCONNECTED}$$

Da CNNs primär auf Bilddateien angewandt werden, wird von diesem Punkt an von einer Bilddatei als Eingabe (INPUT) ausgegangen. Hier beschreiben die ersten beiden Dimensionen die Breite und Höhe, während die dritte Dimension die Farbwerte für den jeweiligen Pixel angibt. Die Konvolutions-, Aktivierungs-, Pooling- und Fully-Connected-Layers werden in den folgenden Unterabschnitten erläutert.

### 2.2.1 Konvolution

Die Konvolution, auch Faltung genannt, ist eine mathematische Operation auf zwei beliebigen Funktionen  $f(x)$  und  $g(y)$ , die eine dritte Funktion, die Konvolution  $s(x) = f(x) * g(y)$  ergibt, welche beschreibt, wie sich die Verläufe der beiden Funktionen gegenseitig beeinflussen:

$$s(x) = (f * g)(x) = \int_{-\infty}^{\infty} f(y)g(x - y)dy \quad (2.2.1)$$

In einem Großteil der Anwendungsfälle der Faltung bestehen diese beiden Funktionen aus einer Eingabefunktion  $x$  und einer Gewichtungsfunktion  $w$ , die Ausgabe ist die Merkmalsdimension. In einer praktischen Anwendung wären dies bspw. eine Reihe von zeitabhängigen Messwerten  $x(t)$  und eine Gewichtungsfunktion  $w(a)$  in Abhängigkeit des Alters der Messung.  $w(a)$  wird im Kontext der Konvolution auch Kernel genannt. Für ungültige Zeiten (bspw. Messwerte welche sich in der Zukunft befinden würden) gilt  $w = 0$  [GBC16, Kap. 9]:

$$s(t) = (x * w)(t) = \int_{-\infty}^{\infty} x(a)w(t - a)da \quad (2.2.2)$$

Unter der Annahme, dass die Eingabewerte nicht stetig sondern diskret sind (bspw. zeitliche Messwerte in regelmäßigen Abständen) ergibt sich vereinfacht [GBC16, Kap. 9]:

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a) \quad (2.2.3)$$

Diese Funktion lässt sich für zweidimensionale Eingabedaten, wie z. B. eine Bilddatei  $I$  und einen zweidimensionalen Kernel  $K$  erweitern [GBC16, Kap. 9]:

$$S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(m, n)K(i-m, j-n) \quad (2.2.4)$$

Eine alternative Betrachtungsweise dieser Formel basiert auf dem Hadamard-Produkt. Angenommen, es existieren eine Matrix  $I \in \mathbb{R}^2$  und eine Matrix  $K \in \mathbb{R}^{m \times n}$ .

Sei Matrix  $H = K \odot I'$  definiert als Hadamard-Produkt aus  $K$  und  $I' = I_{x,y}$  mit  $x \in [i-m, i]$  und  $y \in [j-n, j]$ .

Dann gilt:

$$S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(m, n)K(i-m, j-n) = \sum_{h \in H} h \quad (2.2.5)$$

Diese Formel ist allerdings nur für Eingabebilder mit nur einem Farbwert pro Pixel gültig, also Graustufenbilder. Für ein Bild mit beliebig vielen Farbwertsdimensionen gilt:

$$S(i, j) = (I * K)(i, j) = \sum_m \sum_n \sum_o I(m, n, o)K(i-m, j-n, o) \quad (2.2.6)$$

Es ist zu beachten, dass der Farbvektor immer komplett verarbeitet wird, statt einem Teilausschnitt wie es mit Höhe und Breite geschieht.

### 2.2.2 Convolutional Layer

Klassische Schichten von neuronalen Netzen nutzen Matrixmultiplikationen der kompletten Eingabe mit einer Parametermatrix um ihre Ausgaben zu berechnen, d. h. jeder einzelne Ausgabewert entsteht aus einer Berechnung basierend auf jedem einzelnen Eingabewert. Obwohl diese Taktik in vielen Einsatzbereichen gut funktioniert, stößt sie insbesondere bei der Bilddatenanalyse an ihre Grenzen, da sie nicht gut skaliert. [LJY19] So benötigt eine Convolutional Layer zum Erkennen eines Merkmals nur einen

Kernel mit einer Anzahl von meistens unter 100 px. [GBC16] Ein weiterer Pluspunkt besteht daraus, dass durch Convolutional Layers bestimmte Merkmale von unterschiedlichen Stellen der Eingabe extrahiert werden können, wie später beschrieben wird.

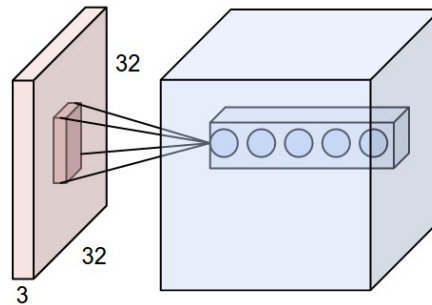


Abbildung 2.2.2: Funktionsweise der Konvolutions-Schicht, aus [LJY19]

Eine konvolutionelle Schicht besteht aus einer Menge von Neuronen, welche die Eingabe über den eben beschriebenen Konvolutionsoperator  $S(i, j)$  auf genau ein Merkmal untersuchen. Daraus folgt, dass die Anzahl der Neuronen in eben dieser Schicht gleich der Größe der entstehenden Merkmalsdimension ist (vgl. Abbildung 2.2.2).

Das Hinzufügen von konvolutionellen Schichten führt allerdings zu mehr Hyperparametern, die optimiert werden können [LJY19]:

- Die Dimensionen des Kernels  $k_1$  und  $k_2$  (obwohl dieser fast immer quadratisch ist, oftmals 3, 5 oder 7)
- Die Anzahl der Neuronen/der Merkmalsdimensionen
- Die Größe der Schrittweite (auch Stride genannt). Hier wird von einem Wert von 1 ausgegangen, was die Größe der Ausgabe nicht verändert.
- Das Padding, also wie sich die Konvolution an den Rändern der Eingabedaten verhält. Hier wird von Zero-Padding ausgegangen, d.h. dass außerhalb der Ränder der Eingabe alle Werte gleich Null sind. Dies hat ebenfalls zur Folge, dass es die Größe der Ausgabe unverändert lässt.

Es gilt es zu beachten, dass der Konvolutionsoperator für alle Werte  $i$  und  $j$  der Eingabedimensionen aufgerufen wird, so dass die Höhe und Breite der Ausgabe (bis auf einige Ausnahmen) identisch zu denen der Eingabe ist. Des Weiteren ist der Kernel pro Neuron konstant. Dies hat u. a. zur Folge, dass das gleiche Merkmal an verschiedenen Stelle in der Eingabe erkennen kann.



Erhält eine konvolutionelle Schicht mit 12 Neuronen also bspw. ein RGB-Eingabebild der Größe  $64 \text{ px} \times 64 \text{ px}$ , so erzeugt es mit einem Stride von 1 als Ausgabe eine dreidimensionale Matrix der Größe  $64 \times 64 \times 12$ , jede der Schichten der dritten Dimensionen deutet auf das Vorhandensein eines speziellen Merkmals an einer gewissen Stelle im Eingabebild hin.

### 2.2.3 Activation Layer

Auf eine konvolutionelle Schicht folgt fast immer eine Aktivierungsschicht. Diese werden dazu genutzt um eine gewisse Nicht-Linearität in den Verlauf des neuronalen Netzes einzuführen. Nicht-Linearität ist in vielen neuronalen Netzen notwendig, da die Probleme, die sie lösen sollen, nicht-linear sind. [GBC16, Kap. 6]

Selbst wenn nach der letzten Schicht eines mehrschichtigen neuronalen Netzes eine Aktivierungsschicht hinzugefügt werden würde, hätte dies zur Folge, dass die vorherigen Schichten wie eine einzige lineare Schicht agieren, was keine Vorteile gegenüber einer einzigen Schicht hat. Somit sollte auf jede einzelne Schicht eine Aktivierungsschicht folgen, um verschiedenste, komplizierte Zusammenhänge zwischen Ein- und Ausgabedaten besser approximieren zu können. [Fra18, Kap. 3]

In Abbildung 2.2.3 sind drei der am meisten verbreiteten Aktivierungsfunktionen zu sehen.

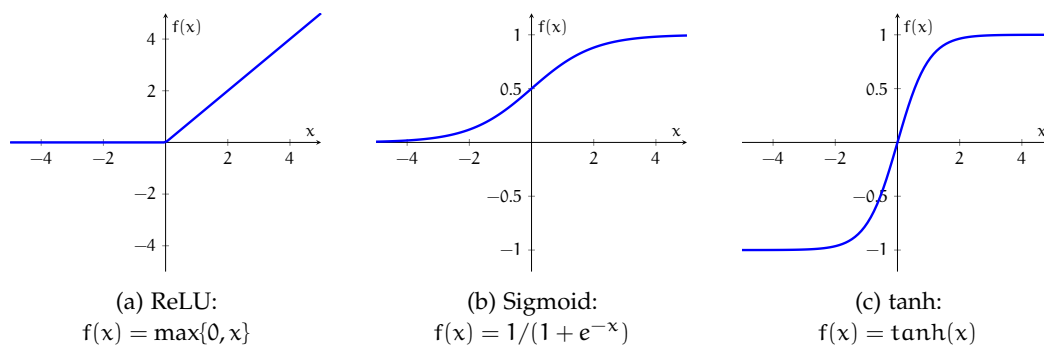


Abbildung 2.2.3: Drei Aktivierungsfunktionen

Die intuitivste Aktivierungsfunktion, die Einheitssprungfunktion, wird in neuronalen Netzen nur für Binärklassifizierungsprobleme in der letzten Schicht genutzt. Dies rührt daher, dass sie keine Ableitung besitzt und daher nicht zum Lernen über Backpropagation (vgl. Unterabschnitt 2.2.9) geeignet ist.

Die ReLU-Aktivierungsfunktion (vgl. Abbildung 2.2.3a) ist die wohl einfachste (nicht-lineare) Funktion, die als Aktivierungsfunktion geeignet ist. Diese Einfachheit führt zu einer vergleichsweise guten Performance, und deshalb großer Beliebtheit in neuronalen Netzen mit vielen Schichten. Ihr Nachteil besteht darin, dass für Bereich  $x < 0$ , praktisch gesehen kein Lernprozess stattfindet, da sie dort immer 0 beträgt. [NIGM18] Zur Lösung dieses Problem es wurden Alternativen, wie z. B. die Leaky-ReLU-Funktion entwickelt (vgl. auch Unterabschnitt 5.3.2).

Die Sigmoid-Funktion eignet sich gut als Aktivierungsfunktion, da sie im Bereich nahe der y-Achse eine hohe Steigung aufweist, während die Veränderungen in anderen Bereichen relativ klein ausfallen. Damit eignet sie sich insbesondere gut zur Klassifizierung. [NIGM18]

Die  $\tanh(x)$ -Funktion ist in ihrer Rolle als Aktivierungsfunktion relativ ähnlich zur Sigmoid-Funktion, mit dem Unterschied dass sie im negativen x-Bereich auch ins Negative verläuft, statt sich an 0 anzunähern. Wie die Sigmoid-Funktion eignet sie sich gut zur Klassifizierung, je nach Anwendungsfall sogar besser, da ihre Ableitung nahe der y-Achse steiler ist. [NIGM18]

Es existieren noch weitere Aktivierungsfunktionen, diese zu Analysieren ist allerdings außerhalb des Rahmens dieser Arbeit.

#### 2.2.4 Pooling Layer

Konvolutionelle Schichten in neuronalen Netzen können die selben Merkmale von unterschiedlichen Stellen extrahieren. Die Position, an der diese Merkmale erkannt wurden, werden auch an die nächste Schicht weitergegeben, d. h. kleine Veränderungen an der Position eines Merkmals führen zu einer veränderten Merkmalsdimension. [GBC16, Kap. 9] Dies kann insbesondere beim Einsatz von mehreren konsekutiven Convolutional Layers zu einer erhöhten Störanfälligkeit führen, deshalb wird oftmals eine Pooling-Layer dazu eingesetzt, das resultierende Netzwerk invariant zu einzelnen Merkmalspositionen zu machen. Des Weiteren dient eine Pooling-Layer dazu, Informationen aus verschiedenen Bildbereichen zu abstrahieren und langsam global zu aggregieren. [GBC16]

Ein weiteres Problem des Einsatzes von ausschließlich konvolutionellen Schichten besteht daraus, dass sie schnell zu einem hohen Anstieg der Parameterzahl im Netzwerk führen können: Jede neue Schicht erzeugt einen weiteren Datenwürfel der Größe  $H \times W \times F$ , wobei  $H$  und  $W$  die Höhe und Breite des Eingabebildes, und  $F$  die Größe der Merkmalsdimension ist. Alle diese Werte werden im Training des neuronalen Netzes optimiert, was zu Performanceeinbußen führen kann. [LJY19]

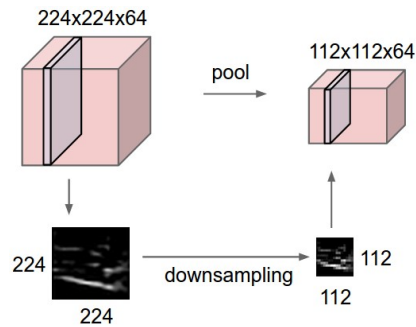


Abbildung 2.2.4: Funktionsweise der Pooling-Layer, aus [LJY19]

Die Pooling-Schicht verarbeitet alle Merkmalsdimensionen ihrer Eingabe unabhängig voneinander.

Für jede Merkmalsdimension überläuft der rezeptive Bereich – ähnlich zur Konvolutions-Schicht – die jeweilige Schicht des Würfels in Höhe und Breite, und berechnet je nach Pooling-Art einen Wert, welcher als positionsabhängige Ausgabe weitergeleitet wird. Dieser Prozess ist in Abbildung 2.2.4 dargestellt.

Dieser Prozess besitzt zwei Hyperparameter: Die Stride (Schrittweite)  $S$  und die eigentliche Größe  $F_1 \times F_2$ . [LJY19]

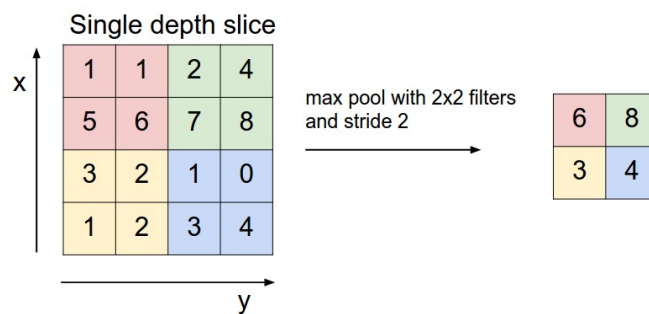


Abbildung 2.2.5: Max-Pooling anhand eines Beispiels, aus [LJY19]

Beim Max-Pooling übernimmt die Poolingoperation den maximalen Wert des von ihr überdeckten Eingabebereiches (vgl. Abbildung 2.2.5).

### 2.2.5 Fully Connected Layer

Viele Convolutional Neural Networks besitzen (mindestens) eine Fully Connected Layer. Das Ziel dieser besteht daraus, die Resultate der konvolutionellen und Pooling-Schichten zu klassifizieren, da diese nur die Wahrscheinlichkeiten zurückgeben, mit der ein gewisses Merkmal an einer gewissen Position erkannt wird. Aus diesem Grund werden oft eine oder mehrere Fully Connected Layers am Ende eines ansonsten fertigen neuronalen Netzes eingesetzt. [GP17, Kap. 4]

Die Funktionsweise einer Fully Connected Layer ist identisch zu einer Schicht eines Multilayer-Perceptrons, welches in Abschnitt 2.2 beschrieben ist.

### 2.2.6 Batch Normalization

Die in [IS15] vorgestellte Technik der Batch Normalization erleichtert die korrekte Initialisierung eines neuronalen Netzes. Das Ziel des genannten Papers bestand ursprünglich daraus, die „Internal Covariate Shift“ zu minimieren, also wie sehr sich die Verteilung der Eingaben der folgenden Schichten verändert, wenn sich die Parameter einer vorherigen Schicht verändern. Sie wird dementsprechend normalerweise zwischen Fully Connected/Convolutional Layers und der darauffolgenden, nichtlinearen Aktivierungsfunktion eingefügt. [LJY19]

Mathematisch betrachtet funktioniert diese Schicht, indem sie die Gauß-Verteilung ihrer Eingabe berechnet, und diese Ergebnisse an die nächste Schicht im Netzwerk weitergibt. [LJY19] Bekommt sie so bspw. aus einer Quelle Eingabewerte im Intervall  $[0, 1]$ , und aus einer weiteren Quelle Werte aus  $[0, 1000]$ , so werden diese Werte (bei gleicher Gewichtung der Eingabewerte der nächsten Schicht) sehr unterschiedlich gewertet. Dieses Problem wird durch die Batchnormalisierung behoben.

In [STIM18] wurde zusätzlich gezeigt, dass durch die Nutzung von Batch Normalization auch die Verlustfunktion geglättet wird, was oft zu einer geringeren Dauer des Training führt. Insgesamt erhöht Batch Normalization also die Stabilität eines Netzes.

### 2.2.7 Loss Function

Wie gut ein neuronales Netz eine Zielfunktion approximiert wird über den Loss gemessen, d. h. das Ziel eines neuronalen Netzes besteht daraus, den Loss in jeder Iteration weiter zu minimieren. Die Lossfunktion, auch Verlustfunktion genannt, beschreibt, wie stark sich die vom Netz berechneten Werte von den zu erzielenden Werten (beim überwachten Lernen meistens die Ground Truth) unterscheiden.

Auch hier existieren verschiedenste Loss-Funktionen. Im Folgenden wird genauer auf den Cross-Entropy-Loss eingegangen. Dieser ist besonders für Klassifizierungsprobleme geeignet. Er ist definiert als [LJY19]:

$$H(p, q) = - \sum_x p(x) \log q(x) \quad (2.2.7)$$

Dabei gibt  $p$  die korrekten Klassen als one-hot-kodierter Vektor an:  $p$  ist also ein Vektor mit  $n$  Elementen, wobei  $n$  die Anzahl der möglichen Klassen ist.

Es gilt:  $f_i$  ist gleich dem  $i$ -ten Element eines Eingabevektors  $f$ .

Eine korrekte Klasse wird mit einer 1 am entsprechenden Element im Vektor kodiert, alle anderen Felder sind gleich 0. Ist für die jeweilige Eingabe nun bspw. die  $m$ -te Klasse korrekt, so gilt  $p_m = 1$  und  $p_{[1, m-1]} = p_{[m+1, n]} = 0$ . Es ist zu beachten, dass der Cross-Entropy-Loss ausschließlich die Wahrscheinlichkeiten für korrekte Klassen bewertet.

$q$  hingegen gibt die berechnete Wahrscheinlichkeit an, dass eine jeweilige Klasse auf die Eingabe zutrifft. Da für  $q$  eine Wahrscheinlichkeitsverteilung, welche sich auf 1 aufsummiert benötigt wird, wird als dessen Eingabe meistens die Softmax-Funktion der vorhergehenden Werte genutzt. Die Softmax-Funktion kann wie die ReLU- oder die Sigmoid-Funktion als Aktivierungsfunktion betrachtet werden. Sie ist definiert durch [LJY19]:

$$S_i = \frac{e^{f_i}}{\sum_j e^{f_j}} \quad (2.2.8)$$

Kombiniert ergibt sich so also für den Cross-Entropy-Loss [LJY19] eines Elementes  $i$ :

$$L_i = - \sum_x p(x) \log \left( \frac{e^{f_i}}{\sum_j e^{f_j}} \right) \quad (2.2.9)$$

Für den durchschnittlichen Loss über alle Klassifizierungen ergibt sich so:

$$L = \frac{1}{N} \sum_{i=1}^N L_i \quad (2.2.10)$$

Hier ist allerdings zu beachten, dass es für ein neuronale Netz mehrere optimale Gewichtungen und Bias geben kann: Würde man bspw. alle Gewichtungen mit

dem selben Skalar multiplizieren, würden diese immer noch zu dem selben Ergebnis führen. Um möglichst kleine Gewichtungen und Biase zu bestimmen, wird ein Regularisierungsterm eingeführt um die Gewichtungen möglichst gering zu halten [LJY19]:

$$L = \frac{1}{N} \sum_{i=1}^N L_i + \lambda R(W) \quad (2.2.11)$$

mit  $\lambda$  als einem Hyperparameter zur Gewichtung und

$$R(W) = \sum_k \sum_l W_{k,l}^2 \quad (2.2.12)$$

### 2.2.8 Gradientenverfahren

Das Ziel eines neuronalen Netzes besteht daraus, die Werte der Lossfunktionen weitgehend zu minimieren. Dies wäre rein theoretisch durch klassische Analysis möglich, indem man die Ableitungen der Funktionen des Netzes berechnet und aus diesen anschließend die Extrempunkte. In der Praxis besitzt solch ein Netzwerk allerdings zu viele Variablen, als dass die Extremalstellen in einem annehmbaren Zeitraum berechnet werden könnten. [Nie15, Kap. 1] AlexNet, der Gewinner der ImageNet-Challenge 2012 besitzt bspw. etwa 60 Millionen trainierbare Parameter (vgl. Abschnitt 2.2).

Die Funktion  $C(w, b)$  sei die zu minimierende Lossfunktion in Abhängigkeit von den Gewichtungen und Bias des neuronalen Netzes. Zur einfacheren Erklärung gilt, o.B.d.A., dass im Netzwerk nur zwei trainierbare Parameter existieren. Somit ist das Ziel, die Parameter der Funktion  $C(v_1, v_2)$  so zu optimieren, dass das Funktionsergebnis minimiert wird. Grafisch ist eine Beispielfunktion  $C$  in Abbildung 2.2.6 dargestellt.

Das Gradientenverfahren beschreibt einen Prozess zur Lösung dieses Problems: Bei diesem Verfahren beginnt man bei einem Startpunkt auf dem Funktionsgraphen. Von dort aus wird ermittelt, in welche Richtung der Graph (hier) absteigend verläuft. Man schreitet nun in diese Richtung und wiederholt das Verfahren. Dies geschieht so lange, bis man an einem Punkt angelangt, von welchem aus es in keine Richtung mehr bergab geht. Dieser Punkt ist ein lokales Minimum. Es gilt zu beachten, dass die Weite des jeweiligen Schrittes variabel ist: Es kann somit geschehen, dass am Ende des Verfahrens, der jeweilige Punkt um das lokale Minimum herum schwankt.

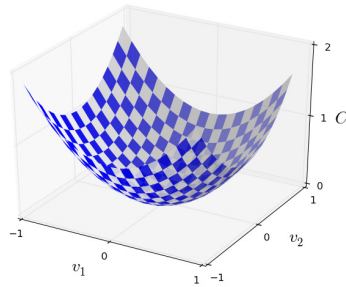


Abbildung 2.2.6: Funktion  $C$  in Abhängigkeit von  $v_1$  und  $v_2$ , aus [Nie15, Kap. 1]

Der Prozess lässt sich anschaulich dadurch darstellen, dass man einen Ball an einem beliebigen Punkt auf dem Graphen fallen lässt. Ignoriert man Reibung, Trägheit und andere physikalische Eigenschaften, rollt dieser immer in Richtung des nächsten lokalen Minimums.

Mathematisch betrachtet ändert sich von einem beliebigen Punkt aus der Wert der Funktion  $C$  um  $\Delta C$ , wenn jeweils eine Entfernung von  $\Delta v_1$  in  $v_1$ -Richtung, und  $\Delta v_2$  in  $v_2$ -Richtung zurückgelegt wird: [Nie15, Kap. 1]

$$\Delta C \approx \frac{\partial C}{\partial v_1} \Delta v_1 + \frac{\partial C}{\partial v_2} \Delta v_2 \quad (2.2.13)$$

Diese Gleichung ist nur eine Annäherung, da statt der Steigung zwischen  $v_1$  und  $\Delta v_1$  nur die Steigung an der Stelle  $v_1$  betrachtet wird. Dies gilt analog für  $v_2$ . Es gelte nun:  $\Delta v$  sei der Vektor der Differenzen in  $v$

$$\Delta v \equiv (\Delta v_1, \Delta v_2)^T \quad (2.2.14)$$

Und  $\nabla C$  sei der Gradient von  $C$ :

$$\nabla C \equiv \left( \frac{\partial C}{\partial v_1}, \frac{\partial C}{\partial v_2} \right)^T \quad (2.2.15)$$

Durch Einsetzen in Gleichung 2.2.13 gilt nun:

$$\Delta C \approx \nabla C \Delta v \quad (2.2.16)$$

Es sei nun:

$$\Delta v = -\eta \nabla C \quad (2.2.17)$$

Wobei  $\eta$  ein beliebiger, positiver Wert sei. Dieser wird auch *learning rate* – Lernrate – genannt.  $\eta$  sollte so klein gewählt werden, als dass Gleichung 2.2.13 eine gute Annäherung bleibt [Nie15, Kap. 1], aber groß genug, als dass die Anzahl der nötigen Iterationen gering gehalten wird. Daraus folgt:

$$\Delta C \approx -\eta \nabla C \cdot \nabla C = -\eta \|\nabla C\|^2 \quad (2.2.18)$$

Da  $\|\nabla C\|^2$  immer größer/gleich 0 ist, verringert sich der Wert C solange die Annahme in Gleichung 2.2.17 gilt.

Es werden also  $\Delta v_1$  und  $\Delta v_2$  nun in jeder Iteration nach dieser Annahme gewählt, um den Wert von C in jedem Durchlauf des Prozesses weiter zu verringern. Nach ausreichend vielen Iterationen erhält man die Koordinaten eines lokalen Minimums.

Der Prozess des Gradientenverfahrens stößt allerdings in diesem Anwendungsfall an seine Grenzen, da hier globale statt lokale Extrempunkte gesucht werden. Ein weiteres Problem besteht daraus, dass die Verlustfunktion C vergleichsweise komplex ist, da sie aus dem Durchschnitt der Verlustfunktionen aller Klassen besteht (vgl. Gleichung 2.2.11).

Beide Probleme lassen sich durch die Nutzung des stochastischen Gradientenverfahrens lösen: Dabei wird, pro Iteration, statt der Durchschnitt aller Lossfunktionen, nur eine einzigen, zufällig ausgewählten Verlustfunktion betrachtet. [SSBD14, Kap. 14.1]

Sollte man nun an einem lokalen Minimum einer Verlustfunktion angelangt sein, so wird in der nächsten Iteration eine andere Funktion betrachtet, welche an dieser Stelle möglicherweise kein Minimum besitzt.

Des Weiteren wird der Rechenaufwand drastisch reduziert, da die Startkoordinaten des jeweils nächsten Durchlaufs anhand nur einer Funktion, statt anhand des Durchschnitts aller Verlustfunktionen berechnet wird.

Ein praktischer Nachteil des stochastischen Gradientenverfahrens besteht allerdings daraus, dass bei dem klassischen Gradientenverfahren die Ergebnisse der unterschiedlichen Verlustfunktionen parallel berechnet werden können, dies beim stochastischen Gradientenverfahren nicht möglich ist, da das Ergebnis einer Iteration auf dem der vorherigen Iteration basiert. Es wird also ein Kompromiss beschlossen, in dem in jedem Durchgang eine Teilmenge der Verlustfunktionen genutzt wird. [Bot12, Kap. 19.5.1]



Dies vereint die Vorteile beider Gradientenverfahren (die Meidung der lokalen Minima des stochastischen Gradientenverfahrens und den möglichen Parallelismus des klassischen Gradientenverfahrens).

Alle in diesem Unterabschnitt erklärten Verfahren lassen sich analog auf eine größere Anzahl an Parametern anwenden.

### 2.2.9 Backpropagation

Um das Gradientenverfahren anwenden zu können, wird der Gradient  $\nabla C_x$  der Verlustfunktion  $C_x$  benötigt. Da in einem neuronalen Netz der Trainingsdatensatz fest steht, muss dieser Gradient nur in Abhängigkeit der Gewichtungen  $w$  und Biase  $b$  des Netzes gebildet werden. Gesucht werden also  $\frac{\partial C}{\partial w}$  und  $\frac{\partial C}{\partial b}$  für jeweils jede Gewichtung und Bias des Netzes.

Außerdem sei die Aktivierung, also die Ausgabe eines beliebigen Neurons definiert als: [Nie15, Kap. 2]

$$a_j^l = \sigma \left( \sum_k w_{jk}^l a_k^{l-1} + b_j^l \right) \quad (2.2.19)$$

Sei nun  $w^l$  die Gewichtungsmatrix einer Schicht  $l$ . Ein Wert  $w_{jk}^l$  gibt also wie oben die Gewichtung vom  $k$ -ten Neurons in der  $(l-1)$ -ten Schicht zum  $j$ -ten Neuron in der  $l$ -ten Schicht. Analog existiert ein Biasvektor  $b^l$ , welcher für einen Wert  $b_j^l$  den Bias des  $j$ -ten Neurons der  $l$ -ten Schicht angibt.  $\sigma$  ist die Aktivierungsfunktion des jeweiligen Neurons. Es existieren insgesamt  $L$  Schichten.

Diese Gleichung lässt sich in Vektorform wie folgt darstellen: (nach [Nie15, Kap. 2])

$$a^l = \sigma(w^l a^{l-1} + b^l) \quad (2.2.20)$$

Ein wichtiger Vorteil dieser vektorisierten Formeln ist (neben ihrer Lesbarkeit) die Möglichkeit, sie parallel einfacher zu berechnen.

Des Weiteren seien die gewichteten Eingaben  $z^l$  einer Schicht  $l$  definiert durch: (nach [Nie15, Kap. 2])

$$z^l \equiv w^l a^{l-1} + b^l \quad (2.2.21)$$

Daraus folgt  $a^l = \sigma(z^l)$ .

Durch Backpropagation kann immer nur das Fehlersignal  $\delta_j^l$  des  $j$ -ten Neurons der  $l$ -ten Schicht berechnet werden. Dieser ist definiert als: (nach [Nie15, Kap. 2])

$$\delta_j^l \equiv \frac{\partial C}{\partial z_j^l} = \frac{\partial C}{\partial a_j^l} \frac{\partial a_j^l}{\partial z_j^l} = \frac{\partial C}{\partial a_j^l} \sigma'(z_j^l) \quad (2.2.22)$$

Er gibt also das Verhältnis der Veränderung der Verlustfunktion des Netzes zur Veränderung einer gewichtete Eingabe  $z_j^l$  an. Für die Ausgabe der letzten Schicht  $L$  des Netzwerkes folgt:

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} \sigma'(z_j^L) \quad (2.2.23)$$

Dies lässt sich auch in vektorisierter Form schreiben: (nach [Nie15, Kap. 2])

$$\delta^L = \nabla_{a^L} C \odot \sigma'(z^L) \quad (2.2.24)$$

Wie in Unterabschnitt 2.2.1 beschreibt der  $\odot$ -Operator das Hadamard-Produkt der jeweiligen Matrizen und  $\nabla_{a^l} C$  ist der Vektor aller partiellen Ableitungen  $\frac{\partial C}{\partial a_j^l}$  einer Schicht  $l$ .

Um den Fehler  $\delta^l$  anhand des Fehlers der nächsten Schicht  $\delta^{l+1}$  wird Gleichung 2.2.24 so umgeformt, dass sie die Veränderung von  $C$  in Abhängigkeit von  $z^{l+1}$  statt in Abhängigkeit von  $z^l$  darstellt. Die vollständige Herleitung ist in [Nie15, Kap. 2] zu finden. Aus ihr ergibt sich:

$$\delta^l = \left( (w^{l+1})^T \delta^{l+1} \right) \odot \sigma'(z^l) \quad (2.2.25)$$

Durch die Kombination der Gleichungen 2.2.24 und 2.2.25 lässt sich der Fehler in einer beliebigen Schicht berechnen.

Um nun anhand der Fehler die jeweiligen Gradienten der Verlustfunktion in Bezug auf die Gewichtungen und Biase zu berechnen gilt nun nach der Herleitung in [Nie15, Kap. 2]:

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \quad (2.2.26)$$

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l+1} \delta_j^l \quad (2.2.27)$$

Mithilfe dieser Formeln lässt sich nun, wie für das Gradientenverfahren benötigt, effizient der Gradient der Verlustfunktion  $C$  für eine Eingabe  $x$  in Abhängigkeit bestimmter Parameter berechnen. Es sein  $a^1 = x$ . (nach [Nie15, Kap. 2])

Zuerst werden für jedes Neuron jeder Schicht  $l$  die Aktivierung und die gewichtete Eingabe berechnet:

$$z^l = w^l a^{l-1} + b^l \quad (2.2.28)$$

$$a^l = \sigma(z^l) \quad (2.2.29)$$

Anschließend wird der Fehlervektor der letzten Schicht  $L$  berechnet (vgl. Gleichung 2.2.24):

$$\delta^L = \nabla_{a^L} C_x \odot \sigma'(z^L) \quad (2.2.30)$$

Dieser wird nun rückwärts durch die Schichten propagiert (vgl. Gleichung 2.2.25):

$$\delta^l = \left( (w^{l+1})^T \delta^{l+1} \right) \odot \sigma'(z^l) \quad (2.2.31)$$

Daraus lassen sich mithilfe der Formeln 2.2.26 und 2.2.27 die Gradienten der Verlustfunktion in Bezug auf die Gewichtungen und Biase herleiten:

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l \quad (2.2.32)$$

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \quad (2.2.33)$$

Da nun diese Gradienten bekannt sind, lässt sich das in Unterabschnitt 2.2.8 beschriebene Gradientenverfahren erfolgreich anwenden.



## VERWANDTE ARBEITEN

---

Da in dieser Arbeit verschiedene Forschungsbereiche miteinander kombiniert werden, ist dieses Kapitel in zwei Abschnitte eingeteilt: Es wird mit einigen verwandten Arbeiten im Bereich der Bildsegmentierung begonnen, dabei wird insbesondere auf die Nutzung von neuronalen Netzen eingegangen. Anschließend werden Clusteringprozesse betrachtet, welche zur Initialisierung der Segmentierungsalgorithmen genutzt werden.

Zuletzt folgt die Analyse der Marsoberfläche, insbesondere der Kratererkennung.

### 3.1 BILDSEGMENTIERUNG

#### 3.1.1 DEC – Deep Embedded Clustering

In [XGF16] wird eine Methode zur Bildsegmentierung durch neuronale Netze ohne vorherige Ground Truth vorgestellt: Das Ziel besteht daraus,  $N$  Punkte  $\{x_n \in X\}_{n=1}^N$  in  $k$  verschiedene Cluster mit den jeweiligen Clusterzentren  $\mu_j, j \in [1, k]$  einzuteilen. Entgegen des naiven Ansatzes, direkt die Farbwerte der einzelnen Pixel als Merkmalsraum  $X$  zu nutzen und zu clustern, wird in [XGF16] nicht direkt anhand der Farbwerte geclustert, stattdessen wird dieser zuerst über eine nichtlineare Funktion  $f_\theta : X \rightarrow Z$  in einen Merkmalsraum  $Z$  abgebildet.  $\theta$  ist dabei ein erlernbarer Parameter, sodass das genutzte Neuronale Netz zwei Parameter zeitgleich optimiert:  $\theta$ , und die Position der Clustermitten  $\mu_j$ .

Der DEC-Algorithmus besteht aus zwei Phasen:

1. Parameterinitialisierung:

Um die Startparameter für  $\theta$  zu Generieren wird ein Stacked Autoencoder (SAE, [VLL<sup>+</sup>10]) schichtenweise aufgebaut. Jede Schicht des SAEs ist ein rauschunterdrückender Autoencoder, welcher wie folgt definiert ist: [XGF16]

$$\tilde{x} \sim \text{Dropout}(x) \tag{3.1.1}$$

$$h = g_1(W_1 \tilde{x} + b_1) \tag{3.1.2}$$

$$\tilde{h} \sim \text{Dropout}(h) \tag{3.1.3}$$

$$y = g_2(W_2 \tilde{h} + b_2) \tag{3.1.4}$$

Zuerst wird ein Encoder-Decoder-Paar dazu trainiert, ein Signal wiederherzustellen, nachdem dieses eine Encoder-Schicht durchlaufen hat. Anschließend wird der Parameter  $h$  dazu genutzt, ein weiteres Paar zu trainieren (vgl. Abbildung 3.1.1). Am Ende dieses Lernens besteht  $\theta$  aus der Menge aller Gewichtungen  $W$  und Bias  $b$  der verschiedenen Encoder Schichten. Diese Encoder-Schichten beschreiben nun die Funktion  $f_\theta$ , welche den originalen Merkmalsraum  $X$  des Bildes in den reduzierten Merkmalsraum  $Z$  umwandelt.

Die initialen Clustermitten werden durch den k-Means-Algorithmus auf dem Merkmalsraum  $Z$  generiert.

## 2. Parameteroptimierung:

Das eigentliche Lernen des DEC-Algorithmus findet in zwei, sich gegenseitig abwechselnden Phasen statt:

Zuerst wird eine Zuweisung von den Datenpunkten des Merkmalsraumes zu den Koordinaten der Clusterzentren berechnet. Zu diesem Zweck wird eine Wahrscheinlichkeitsverteilung, die t-Verteilung genutzt:

$$f(t) = \frac{\Gamma\left(\frac{\nu+1}{2}\right)}{\sqrt{\nu\pi}\Gamma\left(\frac{\nu}{2}\right)} \left(1 + \frac{t^2}{\nu}\right)^{-\frac{\nu+1}{2}} \quad (3.1.5)$$

Dabei ist  $\Gamma$  die Gamma-Funktion und  $\nu$  die Anzahl der Freiheitsgrade.

Diese Funktion berechnet also die Wahrscheinlichkeit, dass ein bestimmter Datenpunkt einem bestimmten Cluster zugewiesen wird.

Anschließend werden die Positionen der Clustermitten  $\mu_j$  und die Parameter  $\theta$  der Funktion  $f_\theta$  optimiert. Die zu verringernde Verlustfunktion ist an dieser Stelle die Kullback-Leibler-Divergenz. Somit wird hier eine Funktion gelernt, welche sich möglichst wenig von der oben beschriebenen, iterativ generierten t-Verteilung unterscheidet. Der eigentliche Optimierungsprozess der Clustermitten und der Parameter für  $\theta$  findet über das Gradientenverfahren (vgl. Unterabschnitt 2.2.8) statt.

Zusammengefasst lernt der Algorithmus in der ersten Phase seiner Ausführung, wie er nach der Anwendung von Dropout-Schichten die ursprüngliche Eingabe bestmöglich wiederherstellen kann, er lernt also wie signifikant bestimmte Merkmale zur Wiederherstellung sind. Diese Initialisierung wird im zweiten Schritt weiter optimiert. Der Vorteil dieser und ähnlicher Methoden besteht daraus, dass das Netzwerk nicht

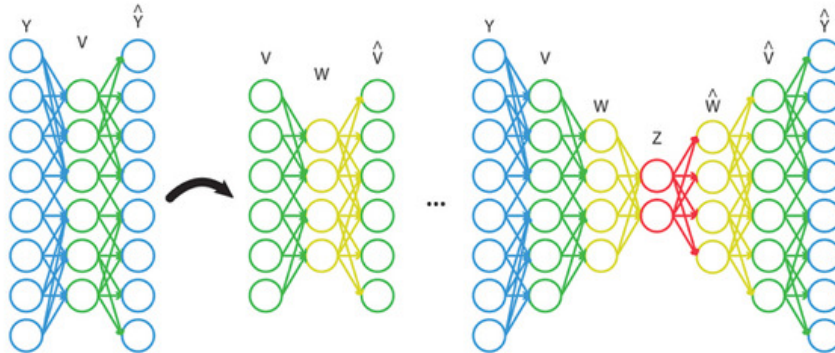


Abbildung 3.1.1: Initialisierung eines Stacked Autoencoders, modifiziert, aus [BK15]. Zu Beginn existiert ein Encoder-Paar (blau) bestehend aus Encoder  $Y$  und Decoder  $\hat{Y}$  mit einer Zwischenausgabe  $V$  (links). Anschließend wird diese Ausgabe  $V$  dazu genutzt, ein weiteres Autoencoder-Paar so zu trainieren, dass es die Eingabe  $V$  durch  $\hat{V}$  approximieren kann (mitte). Dies geschieht iterativ so lange, bis die Anzahl der gewünschten Schichten erreicht ist. Zuletzt werden die so generierten Autoencoder-Paare verschachtelt (rechts).

trainiert werden muss und somit also auch keine Ground Truth benötigt. Dies ermöglicht dessen Einsatz in Bereichen in denen es zu aufwändig ist, Ground Truths manuell zu erstellen. Eine weitere Eigenschaft besteht daraus, dass diese Methode sich für den Einsatz auf unterschiedlichsten Eingabedaten eignet, da sie dynamisch zur Laufzeit lernt, nach welchen Kriterien sie clustern sollte.

### 3.1.2 Bildsegmentierung auf Basis eines Bildclusterings

Ein vergleichbarer, wenn auch einfacherer Ansatz wird in [Kan18] beschrieben. Auch dieser Ansatz erstellt eine Mapping-Funktion  $c_n = f(x_n)$ , die jedem der  $N$   $p$ -dimensionalen Pixel mit dem Merkmalsvektor  $\{x_n \in \mathbb{R}^p\}_{n=1}^N$  eines Eingabebildes ein Clusterlabel  $c$  mit  $\{c_n \in \mathbb{Z}\}_{n=1}^N$  zuordnet. Anders als beim DEC-Algorithmus wird aber auf Autoencoder verzichtet und zusätzlich direkt im Merkmalsraum der Farbwerte geclustert.

Statt das neuronale Netz selbst eine Initialisierung lernen zu lassen, wird in diesem Algorithmus auf Clusteringalgorithmen aus der Bildverarbeitung zurückgegriffen, insbesondere wird der SLIC-Algorithmus [ASS<sup>+</sup>10] genutzt.

Eine Segmentierung wird durch einen iterativen Prozess erreicht, in welchem ein anfangs untrainiertes neuronales Netz eine Bildsegmentierung erzeugt, welche an-

schließlich mithilfe der im Vorhinein erstellten, konstanten Segmentierung optimiert wird. Diese Optimierung besteht daraus, dass für jedes Cluster des SLIC-Algorithmus überprüft wird, welches Label das neuronale Netz am häufigsten den Pixeln im Cluster zugewiesen wurde. Anschließend wird jedes Cluster mit eben diesem häufigsten Label gefüllt und die Differenz zwischen der Ausgabe des Netzwerkes und diesem neu generierten „Ziel“ betrachtet. Der gesamte Algorithmus ist in Abbildung 3.1.2 dargestellt, eine genauere Beschreibung findet sich in Abschnitt 4.1.

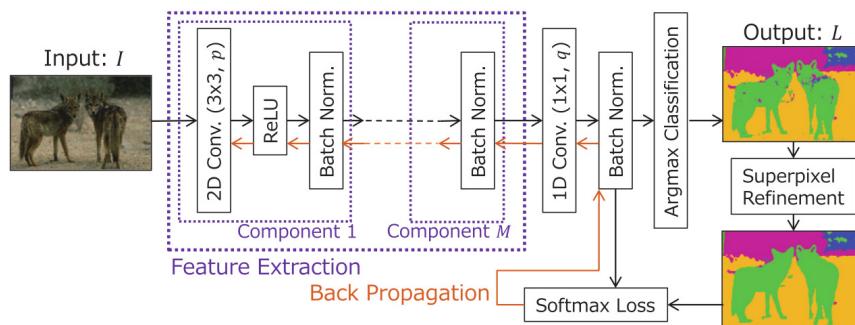


Abbildung 3.1.2: Vorgehensweise nach Kanezaki, aus [Kan18]

Es gilt zu beachten, dass zur Initialisierung ein möglichst feines Clustering genutzt wird, da im Laufe des Prozesses mehrere einzelne Cluster miteinander vereint werden, bis die gewünschte Anzahl an Segmenten erreicht wird.

Insgesamt lernt dieser Algorithmus iterativ die Ausgabe der Clusteringfunktion anzunähern, mit dem Unterschied, dass er in diesem Prozess mehrere Clusterzentren miteinander vereinen kann. Da in [Kan18] auf Basis des SLIC-Algorithmus gearbeitet wird, lernt das neuronale Netz hauptsächlich anhand der Farbwerte einzelner Pixel und deren Entfernungen zu ähnlichen Pixeln, wie es eine Eingabe segmentieren soll. Damit kann es als eine vereinfachte Version des DEC-Algorithmus betrachtet werden, da dieser dynamisch bestimmt, Anhand welcher Merkmale das Clustering geschehen soll.

### 3.2 BILDCLUSTERING

Der in Unterabschnitt 3.1.2 vorgestellte Algorithmus benötigt zur Initialisierung ein relativ feines Clustering der Eingabedatei. Erreicht wird dieses über den SLIC Superpixel Algorithmus [ASS<sup>+</sup>10]. Dieser besteht aus der Anwendung des k-Means Algorithmus auf einem fünfdimensionalen Merkmalsraum, bestehend aus drei Dimensionen für die



jeweiligen Farbwerte eines Pixels im CIELAB Farbraum und jeweils einer Dimensionen für die X- und Y-Koordinaten des Pixels. [ASS<sup>+</sup>10] Der k-Means-Algorithmus teilt eine beliebige Anzahl von Messpunkten (in diesem Fall je ein Merkmalsvektor pro Pixel) in eine vorher festgelegte Menge an Clustern ein.

Dieser Algorithmus ist weitverbreitet und liefert oft gute Ergebnisse auf unterschiedlichen Eingaben, er eignet sich allerdings nur bedingt um Graustufenbilder zu clustern. Da nicht garantiert ist, dass Eingabedateien mehrfarbig sind, eignet sich eine farbbasierte Clustering-Methode wie SLIC nicht immer um diese zuverlässig zu clustern.

Des Weiteren liefert SLIC bei Eingaben in denen ein Segment starke Hell-/Dunkel-Differenzen besitzt keine optimalen Ergebnisse, da es primär anhand der Farbwerte bzw. Helligkeitswerte clustert. Diese Problematik ist genauer in Abschnitt 4.2 beschrieben.

### 3.2.1 Texturbasiertes Clustering

Neben dem farbbasierten Clustering existiert auch das texturbasierte Clustering. Bei diesem werden weniger die Farbwerte von benachbarten Pixeln, sondern vielmehr die Texturen in benachbarten Bereichen verglichen.

Eine Methode des texturbasierten Clusterings wird in [JF91] beschrieben: Dort wird eine Reihe von Gabor-Filtern dazu benutzt, die Textur des Bildes zu analysieren. Dieser Prozess verläuft wie folgt:

Zuerst wird eine Reihe von Gabor-Filtern – auch Filterbank genannt – erstellt. Ein Beispiel, welches nach der originalen Ausarbeitung nachgestellt wurde, findet sich in Abbildung 3.2.1. Es gilt zu beachten dass jeder Filter in mehrmals in unterschiedlichen Größen erstellt wird.

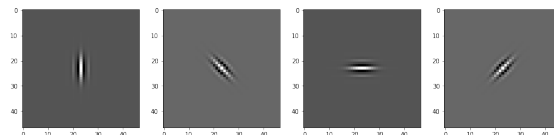


Abbildung 3.2.1: Zum texturbasierten Clustering genutzte Filterbank

Diese werden anschließend über das in den Unterabschnitten 2.2.1 und 2.2.2 beschriebene Konvolutionsverfahren auf die Eingabedatei angewandt. Dieser Vorgang resultiert in einem Datenwürfel, bei welchem die ersten beiden Dimensionen gleich der Höhe und Breite der Eingabebilddatei sind, und die dritte Dimension gleich der

Anzahl der genutzten Filter ist. Somit enthält jede Schicht des Würfels Informationen darüber, wie sehr und wo im Bild das Muster der jeweiligen Filterbank „erkannt“ wird. Dies ist in Abbildung 3.2.2 sichtbar.

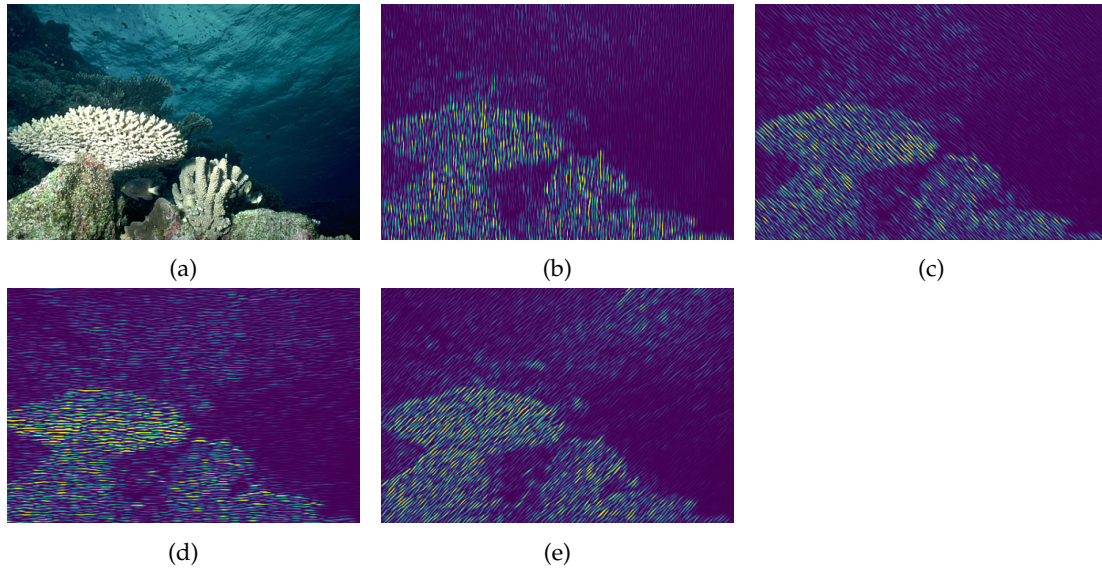


Abbildung 3.2.2: Ergebnisse (b-e) der Konvolution des Beispieldbildes (a, aus [AMFM11]) mit den jeweiligen Filtern

Ähnlich zur Vorgehensweise von SLIC werden dem Datenwürfen zusätzliche Informationen über die jeweiligen X- und Y-Koordinaten hinzugefügt. Anschließend wird der k-Means-Algorithmus auf diesen Datenwürfel angewandt, es werden also Pixel mit ähnlichen Texturen und geringer Distanz zueinander in Cluster zusammengefasst.

In der Praxis sind für die erfolgreiche Anwendung dieser Methode allerdings noch einige Optimierungen notwendig: [mat15]

**WEICHZEICHNUNG** Da die Anwendung des Konvolutionsverfahrens, wie in Abbildung 3.2.2 sichtbar, zu einer streifenartigen Merkmalsextraktion führen kann, macht es in vielen Fällen Sinn, die jeweiligen Merkmalsebenen weichzuzeichnen. Als Radius eignet sich hier ein Wert, welcher groß genug ist um die Streifen zu entfernen, aber klein genug ist, als dass die Ränder der jeweiligen erkannten Texturen nicht in benachbarte Texturen überfließen.

**RÄUMLICHER BEZUG** Damit diese Cluster eine (bessere) räumliche Beziehung zueinander haben, wird zu den Merkmalsdimensionen je eine Schicht hinzugefügt,

welche ausschließlich mit den X-Koordinaten der jeweiligen Pixel gefüllt ist. Selbiges geschieht für die jeweiligen Y-Koordinaten. Da diese vom k-Means Algorithmus auch zusammen geclustert werden, entsteht ein örtlicher Bezug unter den Clustern. Dies wurde auch in der ursprünglichen Ausarbeitung [JF91] berücksichtigt.

**NORMALISIERUNG** Die jeweiligen Schichten sollten vor der Anwendung des k-Means-Algorithmus normalisiert werden um zu vermeiden, dass manche Merkmale stärker gewichtet werden als andere. Dies ist insbesondere wichtig, wenn wie oben genannt die Farb- oder Positionswerte den Merkmalsdimensionen hinzugefügt werden, da diese sich oft auf unterschiedlichen Skalen gegenüber den Konvolutionsergebnissen befinden.

**FARBDIMENSIONEN** Wenn die Eingabedatei eine RGB-Aufnahme ist, können diese drei Farbkanäle zur besseren Unterscheidung von unterschiedlichen Clustern mit ähnlichen Texturen genutzt werden. Dazu wird die Aufnahme in ihre drei Farbkanäle aufgeteilt und diese drei Schichten anschließend als Merkmalsdimensionen dem Datenwürfel hinzugefügt. Obwohl dies nicht in [JF91] aufgeführt wird, ist es in manchen Anwendungsbereichen zur gängigen Praxis geworden.

Das schrittweise Hinzufügen dieser Optimierungen ist in Abbildung 3.2.3 sichtbar. Neben der in Abbildung 3.2.1 gezeigten Filterbank existieren alternative Filterbänke, u. a.:

- Die Leung-Malik (LM) Filterbank [LM01]
- Die Schmid (S) Filterbank [Scho1]
- Die Maximum Response (MR) Filterbank [Vis]

Diese drei Filterbänke besitzen im Gegensatz zu der in [mat15] vorgestellten Filterbank auch Filter in verschiedenen Größen und rotationsinvariante Filter. Sie sind in Abbildung 3.2.4 dargestellt.

Die Besonderheit der Maximum Response-Filterbank besteht daraus, dass sie zwar 38 Filtern besitzt, nach der Anwendung der Konvolution aber alle bis auf die 6 Filter mit der stärksten Reaktion auf die Eingabe an den k-Means-Algorithmus weitergegeben werden. Zusätzlich werden immer die rotationsinvarianten Filter genutzt.

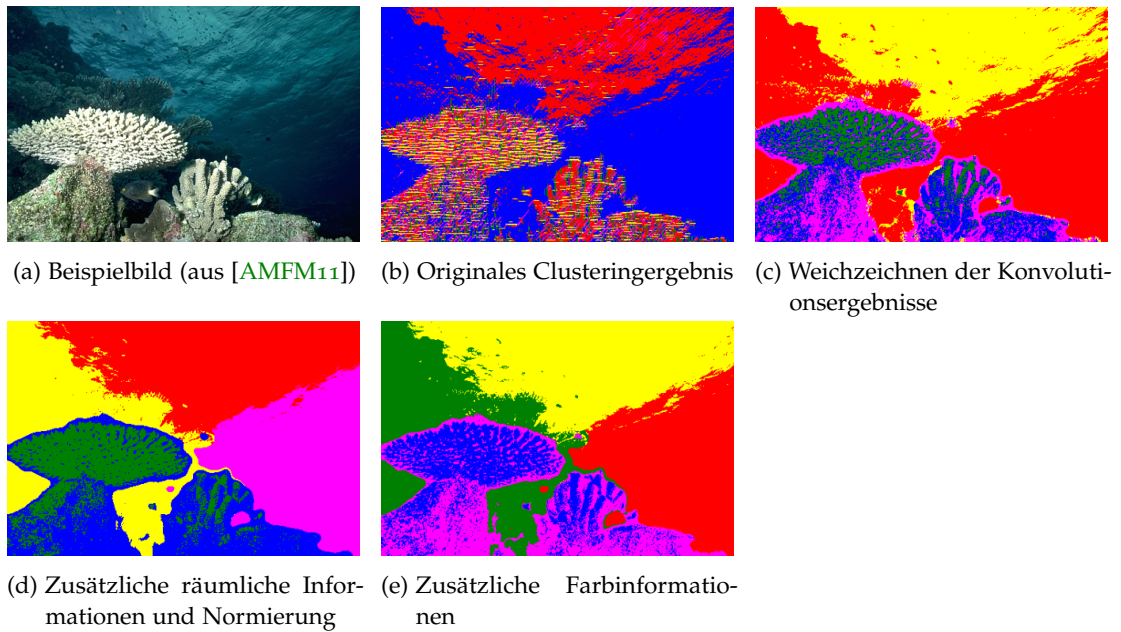


Abbildung 3.2.3: Optimierung des Clusteringverfahrens

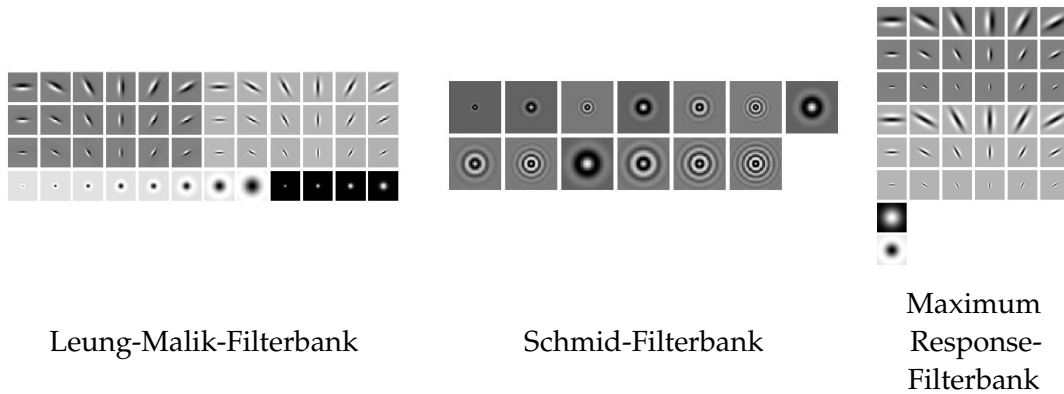


Abbildung 3.2.4: Darstellung dreier Filterbänke, aus [Vis]

### 3.3 KRATERERKENNUNG

In [BDS10] und dessen Fortsetzung [BDS12] wird ein neuer Ansatz zur Kratererkennung vorgestellt. Zuvor wurden diese meist manuell katalogisiert, dies resultierte darin, dass nur die größten Krater dokumentiert wurden, oder darin, dass nur ver-

gleichsweise kleine Bereiche innerhalb eines akzeptablen Zeitraums verarbeitet werden konnten. Daher legen die Autoren insbesondere Wert auf die korrekte Erkennung von kleineren Kratern.

Die erarbeitete Vorgehensweise beginnt damit, dass über einen möglichst effizienten Algorithmus (hier der Algorithmus aus [US09]) eine Vorsortierung von Krater-Kandidaten berechnet wird. Als Alternative zu diesem Algorithmus werden [BSP07] und [SL10] genannt.

Der genutzte Algorithmus ist relativ effizient, da er parallel alle Merkmale eliminiert, die nicht auf Krater hindeuten, statt wie frühere Algorithmen auf eine Brute-Force-Methode zurückzugreifen. Zur Erkennung von Kratern (bzw. Krater-Kandidaten) wird hier die Tatsache genutzt, dass diese auf Abbildungen meistens aus nebeneinander liegenden, starken Schatten- und Spitzlicht-Regionen bestehen.

Nach dieser Vorsortierung und Pre-Processing Schritten (in Form von Histogramm-Optimierung) werden in [BDS10, BDS12] neun Bitmasken (siehe Abbildung 3.3.1) in verschiedenen Positionen, Größen und Ausrichtungen über die Kandidaten gelegt. Die Wahrscheinlichkeit, dass der Kandidat ein Krater ist, berechnet sich aus der Übereinstimmung zwischen den Bitmasken und dem eigentlichen Kandidatenbild. Abschließend werden die Ergebnisse mithilfe eines angepassten AdaBoost Algorithmus optimiert. Das Post-Processing besteht aus der Eliminierung von ungewöhnlich geformten Kratern.



Abbildung 3.3.1: Die neun, zur Merkmalsextrahierung genutzten Bitmasken, aus [BDS12]

In Abbildung 3.3.2 sind die von AdaBoost ausgewählten, am stärksten gewichteten Bitmasken-Überlagerungen dargestellt. Es ist zu beachten, dass der Krater im Hintergrund nur ein Beispiel ist, und nicht alle Krater, die von den jeweiligen Bitmasken überlagert werden, darstellt.

### 3.3.1 Kratererkennung über Neuronale Netze

Auf der Basis des erwähnten, automatisch generierten Marskrater-Datensatz, wird in [CLLD16] ein neuronales Netzwerk daraufhin trainiert, selbst unterscheiden zu können, ob ein Kraterkandidat auch wirklich ein Krater ist. Die Architektur des Netzwerkes ist in Abbildung 3.3.3 zu sehen. Zur Evaluierung der Ergebnisse wird hier das 10-fache Kreuzvalidierungsverfahren genutzt. Dies bedeutet, dass der Eingabe-

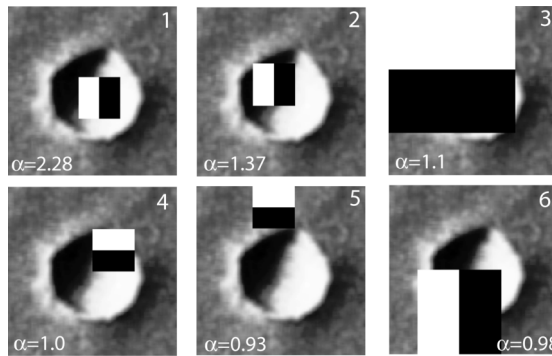


Abbildung 3.3.2: Die sechs am stärksten gewichteten Bitmasken, aus [BDS12]

datensatz zehn-geteilt wird, und jeweils neun Teile zum trainieren und ein Teil zum Evaluieren genutzt wird. Das Ergebnis ist der Durchschnitt der jeweiligen  $F_1$ -Scores.

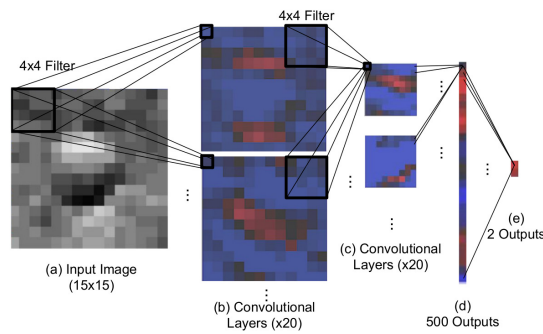


Abbildung 3.3.3: Die Architektur des Netzwerkes, aus [CLLD16]

Der Unterschied zwischen dieser Methode und dem hier vorgestellten Ansatz besteht darin, dass in dieser Methode nur bewertet wird, ob Kraterkandidaten wirkliche Krater sind oder nicht, während hier ein Ansatz entwickelt wird, der aus einer kompletten Aufnahme der Marsoberfläche einzelne Krater(kandidaten) erkennt.

Für die Segmentierung der Marsoberfläche wird auf dem Ansatz von Kanezaki *et. al.* [Kan18] aus Unterabschnitt 3.1.2 aufgebaut. Das Ziel besteht daraus, eine Eingabebild-datei ohne vorhandene Ground Truth durch neuronale Netze zu segmentieren.

In den folgenden Abschnitten wird beschrieben, welche Veränderungen vorgenommen wurden, um den Ansatz für die Segmentierung der Marsoberfläche zu optimieren.

Nachdem kurz auf die gesamte Funktionsweise des Algorithmus eingegangen wird, folgt eine Übersicht über den Initialisierungsalgorithmus und Wege diesen zu optimieren. Dazu zählen bspw. die Nutzung alternativer Filterbänke, deren Größen, Gewichtungen der Merkmale und Anzahl der zu generierenden Cluster. Anschließend wird der eigentliche Algorithmus versucht zu verbessern, gefolgt von einzelnen Anpassungen der Architektur des neuronalen Netzes und einiger weiterer Hyperparameter.

#### 4.1 FUNKTIONSWEISE DER SEGMENTIERUNG

Die grundsätzliche Funktionsweise des Algorithmus aus [Kan18] findet sich in Abschnitt 3.1.2, eine gröbere Übersicht ist in Abbildung 4.1.1 zu finden.

Das Ziel des Algorithmus besteht darin, eine Mapping-Funktion zu generieren, welche jedem der  $N$   $p$ -dimensionalen Pixel der Eingabedatei genau ein Clusterlabel  $c$  zuordnet.

Im Folgenden gilt für eine beliebige Mapping-Funktion  $F$  die Konvention:

1.  $F$  sei eine Mapping-Funktion
2.  $P_{F,z}$  ist die Menge aller Koordinaten der Pixel im Cluster  $z$  generiert durch  $F$
3. Des Weiteren gibt  $Z_F$  die Menge aller durch  $F$  generierten Cluster an.

Der erste Schritt bei der Erstellung der Segmentierung ist die Initialisierung. Bei dieser wird ein übermäßig feines Clustering der Eingabedatei  $x$  erstellt, dies geschieht über klassische Clusteringalgorithmen wie SLIC [ASS<sup>+</sup>10]. Die Initialisierung besteht also folglich aus einer Mapping-Funktion,  $I(x_{ij}), I : \mathbb{R}^p \rightarrow \mathbb{Z}$ , die jedem  $p$ -dimensionalen Pixel  $x_{ij}$  mit den Koordinaten  $i$  und  $j$  des Eingabebildes ein Cluster

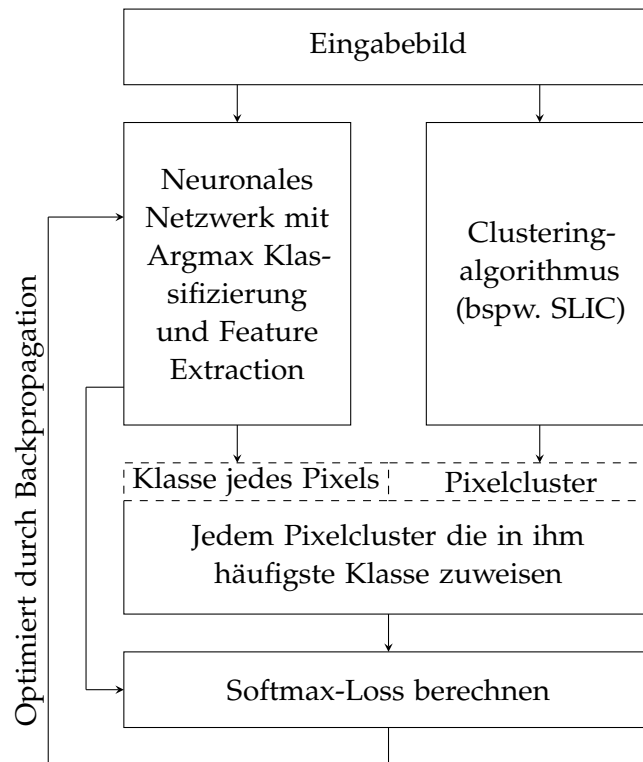


Abbildung 4.1.1: Übersicht über die Funktionsweise des Algorithmus. Während der Ausführung wird im Vorwärts-Durchlauf das Bild segmentiert, im Rückwärts-Durchlauf werden die Parameter des neuronalen Netzes optimiert.

zuordnet. Mithilfe des initialen Clusterings kann das neuronale Netzwerk erlernen, nach welchen Kriterien es bei der Segmentierung verfahren soll. Der wesentliche Unterschied zu [XGF16] besteht folglich daraus, dass das Netzwerk bei dem dortigen Ansatz anhand der Autoencoder selbst erlernt, welche Merkmale relevant für die Segmentierung sind. Dies geschieht dadurch, dass durch den Autoencoder Merkmale extrahiert werden, welche anschließend über den k-Means Algorithmus geclustert werden. In [Kan18] hingegen wird ausschließlich ein externer Clusteringalgorithmus dazu genutzt, die wichtigen Merkmale zu bestimmen. An dieser Stelle ist erwähnenswert, dass viele externe Clusteringalgorithmen, die bei diesem Zweck Anwendung finden können, intern auch ein k-Means Clustering berechnen. So wird auch bei dem SLIC-Algorithmus, welcher in der originalen Implementierung nach [Kan18] genutzt wird, eine abgewandelte Version des k-Means Algorithmus angewandt. [ASS<sup>+</sup>10]



In [Kan18] beginnt der eigentliche Lernprozess damit, dass die Eingabedatei das neuronale Netzwerk durchläuft. Durch diesem Prozess entsteht eine weitere Mapping-Funktion  $F_e(x_{ij})$ ,  $F_e : \mathbb{R}^p \rightarrow \mathbb{Z}$ , wobei  $e$  die Anzahl der bisherigen Epochen, die das Netzwerk durchlaufen hat, angibt. Anschließend werden für jedes Cluster in  $Z_{F_e}$  die Koordinaten der in ihm enthaltenen Pixel bestimmt.  $\langle i, j \rangle$  wird hier und im Folgenden zur Beschreibung eines Tupels bestehend aus X- und Y-Koordinaten genutzt.

$$P_{F_e, z \in Z_{F_e}} = \{\langle i, j \rangle \mid F_e(x_{ij}) = z\} \quad (4.1.1)$$

Selbiges geschieht für die Initialisierung:

$$P_{I, z \in Z_I} = \{\langle i, j \rangle \mid I(x_{ij}) = z\} \quad (4.1.2)$$

Nun wird für jedes Cluster in  $Z_I$  bestimmt, welcher Wert aus  $F_e$  am häufigsten in dem jeweiligen Cluster vorkommt, und das Cluster anschließend damit gefüllt. Das dadurch entstehende Clustering sei  $F'_e$ . Durch diesen Schritt wird dem neuronalen Netzwerk ein Zielclustering zugeführt, welches die Clusterpositionen und -größen des ursprünglichen Clusterings berücksichtigt, dieser Prozess wäre also äquivalent zur wahrscheinlichkeitsbasierten Zuweisung der Datenpunkte des Merkmalsraumes zu den einzelnen Clusterzentren im DEC-Algorithmus (vgl. Unterabschnitt 3.1.1, „Parameteroptimierung“).

Anschließend wird der Loss zwischen dem ursprünglichen Ergebnis des Netzes  $F_e$  und des neu generierten Ziels  $F'_e$  berechnet, und eine Backpropagation über das Netzwerk ausgeführt. Dieser Prozess wird wiederholt, bis ein Abbruchkriterium (vgl. Unterabschnitt 4.4.1) erreicht wird. Die insgesamt zu berechnende Mapping-Funktion ist nun das letzte erzeugte Mapping  $F_e$ . Er lässt sich wie folgt als vereinfachter Python-Code darstellen. Es ist zwischen zwei Datentypen für die Speicherung von Clustern zu unterscheiden:

- `tags`: Eine zweidimensionale Matrix mit der Höhe und Breite des Eingabebildes, bei welcher jeder Pixel durch die entsprechende Nummer des ihm zugeordneten Clusters ersetzt ist.
- `coordinates`: Eine Liste von Mengen der Koordinatentupel, welche in einem bestimmten Cluster enthalten sind. Analog zu der Darstellung  $P_{F_e, z}$  aus den vorherigen Paragraphen.

---

**Algorithmus 1** Algorithmus nach [Kan18]

---

**Eingabe:** Bilddatei  $x$ **Ausgabe:** Clustering der Eingabebilddatei

```
1 def gen_coordinates(tags):
2     clusters = []
3     for c in np.unique(tags):
4         clusters.append(numpy.where(c == tags)[0])
5     return clusters
6
7 init_tags = skimage.segmentation.slic(x)
8 init_coordinates = gen_coordinates(init_tags)
9 while abbruchkriterium() is False:
10     current_result = network.step(x)
11     for c in init_coordinates:
12         target_result[c] = scipy.stats.mode(current_result[c])
13     loss = CrossEntropyLoss(current_result, target_result)
14     network.backpropagate()
15     network.optimize()
16 return current_result
```

---

Hierbei sind:

- `numpy.where(x)`<sup>1</sup>: Gibt Elemente zurück, bei welchen die angegebene Bedingung  $x$  zutrifft.
- `skimage.segmentation.slic(x)`<sup>2</sup>: Der SLIC-Algorithmus aus [ASS<sup>+</sup>10], angewandt auf ein Eingabebild  $x$ . Gibt Ergebnisse in der `coordinates`-Darstellung zurück.
- `scipy.stats.mode(x)`<sup>3</sup>: Der statistische Modus der Menge  $x$ .
- `abbruchkriterium()`: Vgl. Unterabschnitt 4.4.1
- `step`, `CrossEntropyLoss`, `backpropagate`, `optimize`: Die üblichen Methoden zur Evolution eines neuronalen Netzes aus PyTorch<sup>4</sup>.

---

1 <https://docs.scipy.org/doc/numpy/reference/generated/numpy.where.html>

2 <https://scikit-image.org/docs/dev/api/skimage.segmentation.html#skimage.segmentation.slic>

3 <https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.mode.html>

4 <https://pytorch.org/>

Durch die Tatsache, dass in jeder Iteration ein Cluster mit dem am häufigsten in ihm enthaltenen Wert gefüllt wird, verringert sich in der jeweils neu generierten Zielsegmentierung  $F'_e$  die Anzahl der Segmente. Da das neuronale Netz sich diesem Ziel dynamisch anpasst, folgt daraus, dass ein Netzwerk welches in der ersten Epoche bspw. 100 Segmente generiert, nach einigen Epochen wesentlich weniger Segmente erzeugt. In diesem Prozess lernt das Netzwerk auch, die Clusteringkriterien der Initialisierung zu lernen und anhand derer seine eigene Segmentierung anzupassen. Dies ist dadurch begründet, dass die Zielsegmentierung zwar oft unterschiedliche Cluster enthält, die Begrenzungen dieser allerdings immer gleich der Begrenzungen des initialen Clusterings sind. Grafisch ist der Prozess der Verringerung der Segmentanzahl in Abbildung 5.3.1 zu sehen: Die Anzahl der Segmente steigt ab und konvergiert gegen einen Wert, welcher pro Eingabeaufnahme unterschiedlich ist. Da ein einzelnes oder sehr wenige Segmente allerdings meistens eine schlechte Segmentierung bedeutet, muss das neuronale Netzwerk an der richtigen Stelle abgebrochen werden. Unterabschnitt 4.4.1 beschäftigt sich mit diesem Prozess.

Es ist zu beachten, dass für jede Iteration die selbe Initialisierung  $I$  benutzt wird.

#### 4.2 EIGENSCHAFTEN DER INITIALISIERUNG

Eine Veränderung des ursprünglichen Algorithmus, welche starken Einfluss besitzt, besteht aus der Modifizierung des Initialisierungsalgorithmus.

Der in [Kan18] genutzte SLIC-Algorithmus eignet sich zwar gut für die meisten mehrfarbigen Fotografien, da die Aufnahmen der Marsoberfläche allerdings nur in Graustufen vorhanden sind, würden so hier keine guten Resultate produziert werden.

So ergibt ein Clustering des Kraters aus Unterabschnitt 2.1.1 durch den SLIC-Algorithmus [ASS<sup>+</sup>10] das in Abbildung 4.2.1b sichtbare Ergebnis.<sup>5</sup> Dort ist erkennbar, dass der Krater in jeweilige Licht- und Schattenregionen (bedingt durch den Lichteinfall im flachen Winkel) unterteilt wird. Außerdem wird die raue Struktur ringförmig um den Krater herum schlecht erfasst: An dieser Stelle wird jeder Hügel separat als einerseits helle, andererseits dunkle Stelle markiert. Das Phänomen, dass ein Krater durch eine starke Differenz an Licht- und Schattenregionen erkannt wird wird sich im Großen und Ganzen zwar in Abschnitt 3.3 zunutze gemacht, ist hier allerdings ungewollt.

Wenn nun der in Unterabschnitt 3.1.2 beschriebene Ansatz verfolgt wird, wird das neuronale Netz daraufhin trainiert, eine Aufnahme anhand ihrer Helligkeits-

<sup>5</sup> SLIC-Implementierung: scikit-image

Parameter: compactness=5, n\_segments=10, enforce\_connectivity=False

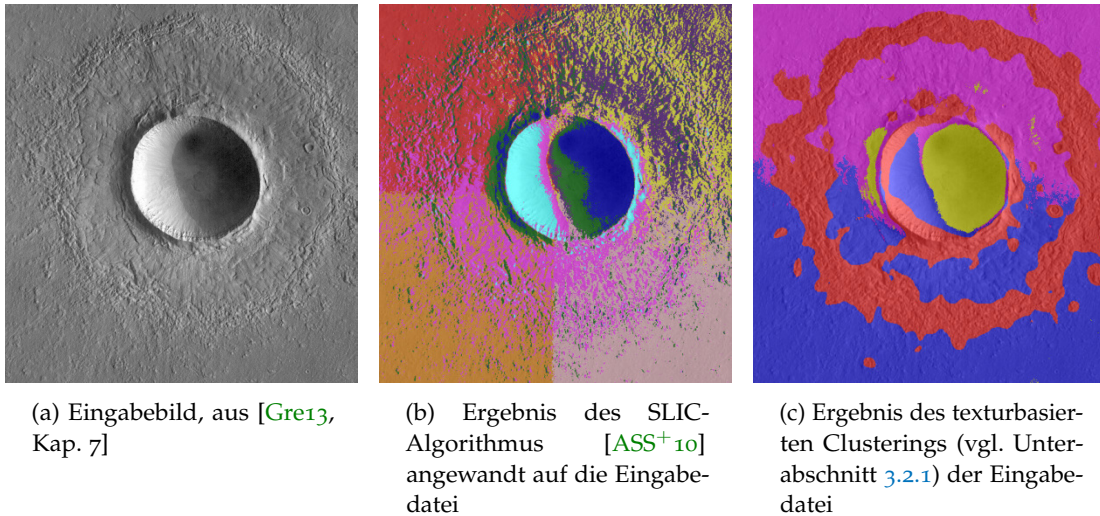


Abbildung 4.2.1: Clustering eines Graustufenbildes eines Kraters der Marsoberfläche

informationen hin zu trainieren. Da dies nicht gewollt ist, wird statt einem farb-/helligkeitsbasierten Clusteringalgorithmus wie SLIC ein texturbasiertes Clustering genutzt.

Anstelle des SLIC-Clusterings wird nun die in Unterabschnitt 3.2.1 vorgestellte Methode des texturbasierten Clusterings verwendet. Von dieser Stelle an sind, wenn nicht anders benannt, die Gewichtungen der verschiedenen Werte für die Farbwerte, X-/Y-Positionen und Reaktion auf Gaborfilter alle gleich 1. Diese Parameter werden in Unterabschnitt 4.2.3 genauer erläutert und verbessert.

Das Resultat eines Clusterings durch die genutzte, texturbasierte Methode<sup>6</sup> ist in Abbildung 4.2.1c sichtbar: Man erkennt, dass der „Ring“ um den eigentlichen Einschlagskrater eine eigenständige Textur besitzt, welche unterschiedlich zu dem Rest der Oberfläche ist. Eine ähnliche Oberflächenstruktur ist direkt um den Krater herum vorhanden. Beide Vorkommnisse dieser ähnlichen Struktur werden vom texturbasierten Clustering erfasst, in ein Segment aufgeteilt und (hier durch eine rote Färbung) markiert.

<sup>6</sup> Kombiniert mit der Filterbank aus [JF91]

#### 4.2.1 Filterbänke

Hier stellt sich nun die Frage, welche der vorgestellten Filterbänke sich gut dazu eignet, die Eingabedatei zu clustern. Die größten Unterschiede zwischen den einzelnen Filterbänken bestehen darin, dass einige von ihnen rotationsinvariante Filter enthalten, und andere in mehreren Größen vorhanden sind. Die Größendifferenz lässt sich in der Anwendung des Algorithmus dadurch ausgleichen, dass jeder Filter zusätzlich in weitere Größen skaliert wird, und in jeder Skalierungsstufe angewandt wird. Die Rotationsinvarianz würde sich höchstens durch die Rotation der Filter approximieren lassen, da so allerdings nicht jeder Winkel abgedeckt werden kann, ist diese Methode ungeeignet. Ein Vergleich verschiedener Filterbänke ist in Unterabschnitt [5.2.1](#) zu finden.

Neben der Auswahl einer geeigneten Filterbank lassen sich innerhalb dieser noch die einzelnen Gewichtungen anpassen: So könnten z. B. die Gewichtung der Koordinaten der Pixel zueinander oder die Relevanz von ähnlichen Farbwerten modifiziert werden, zusätzlich spielt die Skalierung der Filter eine wichtige Rolle.

#### 4.2.2 Größe der Filter

Die Skalierung der einzelnen Filter in den jeweiligen Filterbänken ist Variabel und muss dementsprechend an die Dimensionen des jeweiligen Eingabebildes angepasst werden. Da alle Filterbänke (bis auf die nach [\[JF91\]](#)) jeweils gleiche Filter in unterschiedlichen Größen enthalten, muss nur ein Gesamtwert angepasst werden, statt der Größe jeder einzelnen Skalierung.

Der Skalierungsfaktor  $s$  gibt an, mit welchem Faktor die ursprüngliche Seitenlänge des Filters multipliziert wird,  $s = 1$  bedeutet somit, dass der Filter eine Seitenlänge von 49 px beibehält. In Unterabschnitt [5.2.2](#) werden unterschiedliche Skalierungsfaktoren verglichen.

#### 4.2.3 Gewichtungen der Parameter

Da in dem Prozess des Clusterings alle Optimierungen aus Unterabschnitt [3.2.1](#) miteingeschlossen sind, verarbeitet k-Means einen Datenwürfel, welcher auch die Koordinaten und Farbwerte der jeweiligen Pixel miteinbeschließt. Nun stellt sich die Frage, wie stark diese jeweils gewichtet werden sollten, um ein akzeptables Ergebnis zu erhalten. Aus diesem Grund sind in Unterabschnitt [5.2.3](#) ie Resultate des Clusterings mit unterschiedlichen Werten für die jeweiligen Gewichtungen zu sehen.

Diese Gewichtungen sind dabei relativ zu der Gewichtung eines einzigen Merkmals, welche durch die Gabor-Filter erstellt wurden.

#### 4.2.4 *Anzahl der Cluster*

Bis an diese Stelle haben sich vier Cluster sehr gut zur Visualisierung der unterschiedlichen Ergebnisse nutzen lassen, dies ändert sich allerdings wenn die jeweiligen Clusterings an das darauf folgende neuronale Netz weitergegeben werden. Die Anzahl der Cluster mit denen begonnen werden soll ist ein entscheidender Wert bei dem Prozess der Segmentierung: Ist diese Zahl zu gering, kann das Netzwerk dieses Clustering nicht weiter verfeinern, ist sie jedoch zu groß, so sind innerhalb eines Clusters nicht genug Informationen vorhanden, als dass das Netzwerk erlernen kann, nach welchen Kriterien das Bild ursprünglich aufgeteilt wurde. Ein Vergleich unterschiedlicher Werte für diese Anzahl der Startcluster ist in Unterabschnitt [5.2.4](#) sichtbar.

### 4.3 EIGENSCHAFTEN DES SEGMENTIERUNGSALGORITHMUS

Der Algorithmus aus [\[Kan18\]](#), also das Füllen der Cluster der dynamisch erstellen Zielsegmentierung mit den derzeitigen Werten, wird größtenteils unverändert gelassen. Eine kleinere Modifikation, die sich allerdings doch als hilfreich erwiesen hat, ist, dass dieser genannte Vorgang nicht in jeder Iteration des Netzes geschieht, sondern nur alle  $n$  Iterationen. Dies ermöglicht dem Netzwerk mehrere Epochen zum Lernen der Methodik jeweiligen Segmentierung, bevor diese wieder verändert wird.

### 4.4 NETZWERKARCHITEKTUR

Die Netzwerkarchitektur im originalen Paper stellt ein einfaches Netzwerk zur Objekterkennung oder Bildsegmentierung dar: Es durchläuft iterativ drei mal eine Reihe mit je einer Convolutional Layer, gefolgt von einer Batch Normalization. Im Folgenden wird geprüft, ob eine Modifikation dieser Architektur in diesem domänenspezifischen Anwendungsfall zu Verbesserungen der Ergebnisse führen kann.

#### 4.4.1 *Abbruchkriterium*

Die originale Implementierung nutzt die Anzahl der Segmente, die das neuronale Netz erzeugt, als Abbruchkriterium für das Netzwerk. Ist diese geringer als ein im Vorhinein festgelegter Wert, so wird das Training des Netzes gestoppt und die aktuelle

Segmentierung als Ergebnis ausgegeben. Neben dieser simplen Methode lässt sich auch der Wert der Verlustfunktion als Abbruchkriterium nutzen. Da dieser allerdings pro Eingabedatei unterschiedlich zueinander ausfällt, kann dieser nicht direkt genutzt werden.

Eine Lösungsmöglichkeit besteht daraus, den Median der relativen Änderung zur jeweils vorherigen Epoche in den letzten  $n$  Epochen zu betrachten. Diese Änderungsrate sinkt während der Laufzeit des Training, da die weitere Optimierung der Segmentierung immer geringer ausfällt. Die Wahl von  $n$  ist entscheidend für die erfolgreiche Funktionsweise dieser Methode, da ein zu kleiner Wert bedeuten könnte, dass das Netzwerk abgebrochen wird, wenn vorzeitig eine lokale Extremstelle auftritt, also z. B. innerhalb von einigen Epochen wenig neue Informationen gelernt werden. Ist  $n$  allerdings zu groß, werden zu viele Epochen betrachtet, sodass eine aktuelle Senkung dieser Änderung erst spät bemerkt wird.

Ein weiterer Ansatz zur Lösung dieses Problems ist die Nutzung des Wertes der Ableitung der Verlustfunktion in der aktuellen Epoche. Dies ist allerdings nicht direkt möglich, da die Verlustfunktion nicht stetig definiert ist, sondern nur an den Stellen der jeweiligen Epochen. Dies lässt sich dadurch ausgleichen, dass die bekannten Punkte durch eine stetige Funktion approximiert werden, welche anschließend abgeleitet werden kann.

Der Vergleich dieser Abbruchkriterien ist in Unterabschnitt [5.3.1](#) aufgeführt.

#### 4.4.2 Aktivierungsfunktionen

In der originalen Implementierung nach [\[Kan18\]](#) enthält das neuronale Netz die ReLU-Funktion als Aktivierungsfunktion vor der Batch Normalization-Layer. Es wird nun untersucht, ob das Entfernen oder Ersetzen einer solchen Aktivierungsfunktionen die Clusteringergebnisse verbessern kann.

In Unterabschnitt [5.3.2](#) werden die am weitesten verbreiteten Aktivierungsfunktionen verglichen: Die ReLU-, die Sigmoid- und die tanh-Funktion (vgl. Unterabschnitt [2.2.3](#)). Außerdem wird geprüft, ob das Entfernen der vorhandenen ReLU-Funktion zu einer Verbesserung der Ergebnisse führen kann.

#### 4.4.3 Pooling Layer

Auch auf Pooling Layers wurde im ursprünglichen Paper verzichtet. Pooling Layers können, wie in Unterabschnitt [2.2.4](#) beschrieben, dabei helfen, Informationen aus verschiedenen Bildbereichen zu abstrahieren und aggregieren, es könnte also in dieser

Hinsicht zu besseren Ergebnissen führen. Ein Problem allerdings ist, dass Pooling Layers die Auflösung des Bildes merklich verringern, wird solch eine Schicht bspw. mit einer Kernelgröße von  $F_1 = F_2 = 2$  genutzt, enthält die Ausgabe nur ein Viertel der Pixel der Eingabedatei. Dies ist zwar ein geringeres Problem bei der Objekterkennung, kann allerdings bei der Segmentierung, bei welcher es auf die möglichst genaue Bestimmung der Kanten von Strukturen ankommt, zu einer Genauigkeitseinbuße führen. Außerdem kann je nach Kernelgröße und weiteren Hyperparametern der Fall eintreten, dass sehr feine Oberflächenmerkmale durch das Pooling verloren gehen. Die Entsprechenden Resultate dieser Experimente sind in Unterabschnitt 5.3.3 zu finden.

#### 4.4.4 *Fully Connected Layers*

Wie in Unterabschnitt 2.2.5 werden Fully Connected Layers in Convolutional Neural Networks dazu eingesetzt, die Resultate der ihr vorhergehenden Schichten zu klassifizieren. Angewandt auf das hier genutzte Netzwerk könnte ihre Nutzung zur Folge haben, dass verschiedene lokale Oberflächenmerkmale besser im globalen Kontext erkannt werden können und ggf. in die selben Cluster wie ihnen ähnliche Regionen eingeteilt werden. Ob dies auch in der praktischen Anwendung der Fall ist, wird in Unterabschnitt 5.3.4 untersucht.

#### 4.4.5 *Anzahl der Konvolutionsschichten*

Die Anzahl der Konvolutionsschichten stellt einen weiteren Hyperparameter dar, welcher Einfluss auf die Qualität der Resultate des Netzes hat. Die originale Implementierung des Algorithmus aus [Kan18] setzt diese Anzahl als dynamisch und vom Nutzer bestimmbar fest, mit einem Standardwert von einer Eingabeschicht, welche die drei Merkmalsdimensionen auf einen Datenwürfel mit 100 Merkmalsdimensionen abbildet, gefolgt von standardmäßig zwei weiteren Konvolutionsschichten. Ob dies ein geeigneter Wert für den hier genutzten Anwendungsfall darstellt wird in Unterabschnitt 5.3.5 überprüft.

### 4.5 ANPASSUNGEN ZUR SEGMENTIERUNG VON MEHRFARBIGEN FOTOGRAFIEN

Mehrfarbige Eingabedateien besitzen mehr Merkmale, welche zur Analyse genutzt werden können, da sie zusätzlich Informationen über den Rot, Grün-, und Blauwert der einzelnen Pixel enthalten. Außerdem sind die Motive allgemeiner Fotografieren (wie z. B. die des BSDS500-Datensatzes, vgl. Unterabschnitt 5.1.2) unterschiedlich



zu denen, die bisher genutzt wurden. Aus diesen Ursachen eignen sich die bisher genannten Parameter nicht universell sowohl zur Segmentierung einer Aufnahme der Marsoberfläche, als auch zur Segmentierung „normaler“ Fotografien.



## EXPERIMENTE

---

### 5.1 DATENSÄTZE

Der vorgestellte Algorithmus wird auf zwei Datensätzen evaluiert: Einerseits wird ein allgemeiner Datensatz zur Überprüfung der Ergebnisse genutzt, aber auch ein domänenspezifischer Datensatz, über den die Analyse der Marsoberfläche evaluiert wird.

#### 5.1.1 Domänenspezifisch

Die in diesem Kapitel zur Optimierung der Parameter genutzten, domänenspezifischen Datensätze stammen größtenteils von der Website des *Planetary Data System* der NASA<sup>1</sup>. Die dort gehosteten Aufnahmen der jeweiligen Instrumente sind größtenteils unverarbeitet und befinden sich in einem speziell für diesen Zweck erstellten Dateiformat, sodass diese erst konvertiert und anschließend so verarbeitet werden müssen, dass für diesen Zweck besser geeignete Bilddateien entstehen. Diese Verarbeitung besteht bei den Aufnahmen der CTX der Konvertierung der Pixelfarbwerte, der Entfernung der Even/Odd Detector Stripes, und der Ausgabe in ein adäquates Bildformat. Dieses kann von dem eigentlichen Analyse-Algorithmus eingelesen und weiter verarbeitet werden.

Es existieren allerdings keine segmentierten Aufnahmen der Marsoberfläche. Aus diesem Grund wird zur eigentlichen Evaluation einer der größten Datensätze an Mars-Kratern, der *Mars Crater Catalog v1 Robbins* [RH12] genutzt, welcher Krater auf dem *THEMIS Daytime IR Global Mosaic* markiert. Insbesondere wird ein Teil der Aufnahme *thm\_dir\_N-30\_210* betrachtet, da auf dieser weniger nicht-fotografierte Stellen vorhanden sind als auf den meisten weiteren Aufnahmen. Des Weiteren wird zur Evaluierung nur ein Teil dieser Aufnahme betrachtet, da zur Segmentierung manuelle Arbeit erforderlich ist:

Da dieser allerdings nur Informationen über die Positionen der Krater auf der Marsoberfläche zur Verfügung stellt, wird die durch den hier vorgestellten Algorithmus

---

<sup>1</sup> [https://pds-imaging.jpl.nasa.gov/portal/mro\\_mission.html](https://pds-imaging.jpl.nasa.gov/portal/mro_mission.html)

erzeugte Segmentieren manuell in zwei Klassen eingeordnet: Krater und Nicht-Krater. Weitere Details zu diesem Vorgang sind in Abschnitt 5.4 zu finden.

### 5.1.2 Allgemein

Ein oft genutzter Datensatz zu Evaluierung von Segmentierungsalgorithmen ist der BSDS-500-Datensatz der UC Berkeley [AMFM11]. Zu diesem Datensatz existieren sowohl mehrere manuell erstellte Ground Truths pro Aufnahme, als auch mehrere auf ihm angewandte Algorithmen, mit denen die hier vorgestellte Methode verglichen werden kann (vgl. Unterabschnitt 5.6.2). Einige der Aufnahmen aus diesem Datensatz wurde bereits in vorherigen Kapitel zur Erläuterung einiger Algorithmen genutzt (vgl. bspw. Abbildung 3.1.2 oder 3.2.2).

## 5.2 MODIFIZIERUNGEN DER INITIALISIERUNG

Im Folgenden ist eine Anwendung der in Kapitel 4 genannten Modifizierungen zu finden.

### 5.2.1 Filterbänke

In Tabelle 5.2.1 ist die Anwendung der in Unterabschnitt 3.2.1 vorgestellten Filterbänke auf vier Beispielbilder (vgl. Abschnitt 2.1.1) sichtbar. Diese sind nach deren Bezeichnungen in dem genannten Abschnitt benannt (Krater, Vulkan, Vulkan mit strahlenförmigen Merkmalen, Gletscher), welche wiederum aus [Gre13, Kap. 7] stammen. Jedes Bild wurde in vier Cluster aufgeteilt und alle Optimierungen des Verfahrens (vgl. Unterabschnitt 3.2.1) wurden angewandt.

**KRATER** Neben dem eigentlichen Krater ist auf dieser Aufnahme der Ring aus größerem Gestein ein wichtiges Merkmal. Dieser wird von allen Filterbänken zuverlässig erkannt, wenn auch mit einer unterschiedlichen Dicke: Die LM- und S-Filterbank selektieren dieses Gestein eher großzügig, die MR-Filterbank hingegen zeigt sehr enge Markierungen dieser Region.

Alle Filterbänke formen einen Ring (oder dessen Ansatz) auf dem Kraterrand, die Maximum Response-Filterbank erzeugt zwei konzentrische Ringe

Der Krater selbst wird von den Filterkombinationen nach [JF91] und der LM-Filterbank leider nur in Hell- und Dunkel-Regionen aufgeteilt, die S-Filterbank zeigt innerhalb des Kraters kein brauchbares Ergebnis. Eine Ausnahme stellt die Maximum-

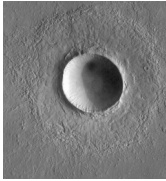
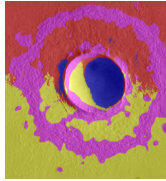
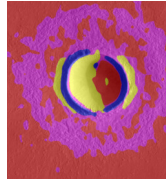
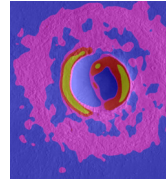
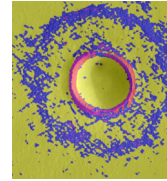

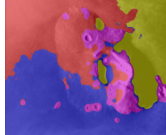
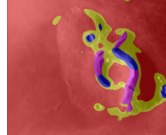
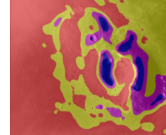
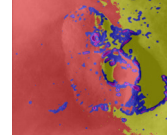
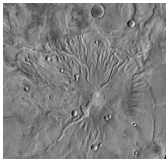
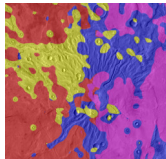
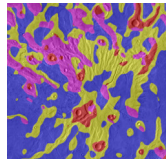
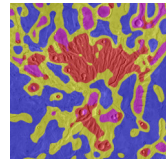
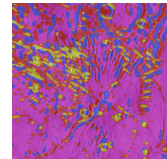
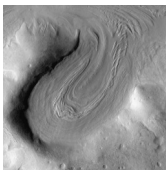
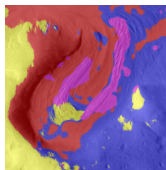
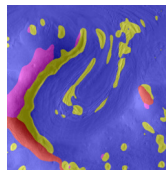
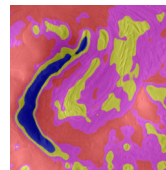
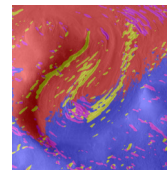
Bezeichnung	Eingabe, aus [Gre13]	Filterbank nach [JF91]	LM-Filterbank [LM01]	S-Filterbank [Scho1]	MR-Filterbank [Vis]
Krater					
Vulkan					
Vulkan mit strahlenförmigen Merkmalen					
Gletscher					

Tabelle 5.2.1: Vergleich verschiedener Filterbänke auf Bildern der Marsoberfläche. Die Farben der jeweiligen Cluster wurden zufällig gewählt und sagen nichts über deren Inhalt aus. Alle Bilder wurden in vier Cluster eingeteilt.

Response-Filterbank dar, welche neben den zwei erwähnten konzentrischen Clustern den Bereich innerhalb des Kraters als ein einziges Cluster erkennt. Dies zeugt von einer Invarianz gegenüber den Helligkeitswerten, da alle weiteren Filterbänke in diesem Bereich zwischen Licht- und Schattenregionen unterschieden haben. Die Maximum Response-Bank erzeugt in diesem Bereich das Resultat, welches sich wohl als bestes zur Weiterverarbeitung eignet, da sie unanfälliger für Helligkeitsveränderungen innerhalb eines Clusters ist und die Kraterränder am genauesten erfasst.

**VULKAN** Der Vulkanberg wird leider von keiner Filterbank optimal erkannt. Der äußere Rand wird nur von der S-Filterbank als ein Cluster erkannt, dies allerdings nicht sehr genau. Der rauere Bereich der Oberfläche, was kleinere Krater miteinbeschließt,

wird vom MR-Filter am ehesten und genauesten erkannt, auf dem zweiten Platz folgt die Filterbank nach [JF91], welche zwar alle Krater in ein Cluster fügt (violett), allerdings eher größer.

Die Vulkanmitte wird von keiner Filterbank erkannt, nur bei der MR-Bank lässt sich eine ringförmige, rauere Stelle um den Krater herum erahnen. Die anderen Filterbänke selektieren an dieser Stelle hauptsächlich nach den Helligkeitswerten. Außerdem ist interessant, dass der LM-Filter die Bergspitzen in vom Umfeld getrennte Cluster einteilt (blau und violett). Somit scheint es, als ließe sich diese Aufnahme über keine Filterbank gut clustern. Am ehesten eignet sich allerdings erneut die MR-Bank.

**VULKAN MIT STRAHLENFÖRMIGEN MERKMALEN** Das Ziel bei der Nutzung dieser Marsaufnahme zur Analyse besteht daraus zu analysieren, mit welcher Filterbank die konzentrischen Strahlen am genauesten erkannt werden. Des Weiteren existieren auf der Aufnahme noch Krater, welche es separat einzuordnen gilt.

Die Filterbank nach [JF91] erkennt die größeren Strukturen (Strahlen und Krater), fügt diese allerdings in dasselbe Cluster ein (gelb und blau).

Die LM-Filterbank scheint kein brauchbares Ergebnis zu produzieren, sie erkennt im Allgemeinen nur einige der Krater (rot) separat von deren Umfeld, die Strahlen sind nicht geeignet geclustert worden.

Die Schmid-Filterbank clustert einen Großteil der Strahlen gemeinsam in ein Cluster (rot). Die Krater hingegen sind nicht fest einer Clusterart zugewiesen, ein Großteil von ihnen befindet sich jedoch auch im roten Cluster.

Wie in den vorherigen Tests ist das Resultat der MR-Filterbank schon fast zu genau, die Cluster sind sehr fein gehalten. Im Gegensatz zu den Alternativen, werden hier die Krater relativ zuverlässig in gelbe Cluster eingeordnet, die Strahlen sind leider zu fein markiert, als dass man sie getrennt erkennen könnte. Dieses Phänomen könnte sich allerdings in der praktischen Anwendung zunutze gemacht werden, da dafür eine zu feine Initialisierung notwendig ist.

**GLETSCHER** Der Gletscher stellt im Bereich der Bildsegmentierung ein vergleichsweise schwieriges Problem dar, da er an vielen seiner Ränder keine feste Grenze zu benachbarten Regionen zeigt, nur links auf der Aufnahme ist er durch eine Hügelreihe strikt begrenzt. Diese Grenze wird von allen Filterbänken separat in ein oder mehrere getrennte Cluster unterteilt, mit Ausnahme der Filterbank nach [JF91]: Diese produziert kein geeignetes Ergebnis, nur die stärker erkennbaren „Streifen“ werden gut erkannt (violett). Die LM-Filterbank erkennt zwar die Abgrenzung auf der linken Seite, allerdings keine anderen Regionen korrekt. Sie liefert bei diesem Bild das wohl schlechteste Resultat. Die Maximum-Response Methode und die Filterbank nach

[Scho1] liefern zwar etwas bessere Ergebnisse, diese sind allerdings nicht ausreichend. Die MR-Bank erzeugt erneut sehr feine Cluster.

Zusammengefasst erzeugt aus dieser Auswahl an Filterbänden die Maximum Response-Methode von [Vis] das wohl am ehesten geeignete Resultat zur Weiterverarbeitung, da sie bei einem Großteil der Beispielbilder das beste Ergebnis produziert hat. Auf dem zweiten Platz folgt die Schmid-Filterbank.

### 5.2.2 Größe der Filter

Es wird nun versucht, die Ergebnisse der Maximum Response-Filterbank zu verbessern, da diese in vorherigen Versuchen die besten Ergebnisse erzielt hat.

Bis jetzt wurden die Marsaufnahmen aus [Gre13, Kap. 7] genutzt, da diese möglichst diverse Oberflächenstrukturen- und Merkmale abdecken. Von dieser Stelle an müssen allerdings Marsaufnahmen genutzt werden, bei denen die Auflösung bekannt ist, somit scheiden die genutzten leider aus. Stattdessen werden Ausschnitte aus der Aufnahme P03\_002147\_1865\_XI\_06N208W der Context Camera des Mars Reconnaissance Orbiters genutzt. Auch diese wurden so ausgewählt, dass sie möglichst diverse Strukturen und Texturen aufweisen. Ein weiterer Vorteil der Nutzung dieser Bilder besteht daraus, dass die Clusteringalgorithmen nun an realitätsnäheren Daten getestet werden können, statt an isolierten Aufnahmen einzelner Besonderheiten der Oberfläche. Die Ergebnisse sind in Tabelle 5.2.2 zu sehen. Als Startwert wurde ein Skalierungsfaktor  $s = 1$  betrachtet, angewandt auf die vorgestellten, quadratischen Filter mit einer originalen Kantenlänge von 49 px. Dieser wurde nach oben und unten hin in Schritten von  $\Delta s = 0.25$  verändert, und mit der skalierten MR-Filterbank ein neues Clustering der vier Ausschnitte erstellt. Die Eingabebilder sind quadratisch und besitzen eine Seitenlänge von 650 px bei einer Auflösung von  $6,17 \frac{\text{m}}{\text{px}}$ . Jedes der Eingabebilder wurde erneut in vier Bereiche eingeteilt, die Farben sind zur Veranschaulichung zufällig gewählt, da der Algorithmus zwar in Segmente unterteilen kann, allerdings nicht bestimmen kann, was diese Segmente enthalten.

Von hier an gilt in allen Vergleichen von Segmentierungen: Die erste Zeile der Tabelle entspricht der ersten Aufnahme, die zweite Zeile entspricht der zweiten Aufnahme, etc..

SKALIERUNGSFAKTOR  $s = 0,25$  Auf Aufnahme a) ist, vergleichbar mit den Beispielaufnahmen aus dem letzten Abschnitt, ein runder Krater zentriert zu sehen. Die geringe Skalierung mit dem Faktor  $s = 0.25$  führt dazu, dass die jeweiligen Filter zu

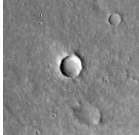
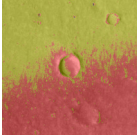
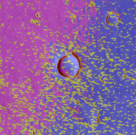
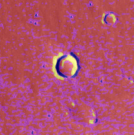
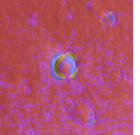
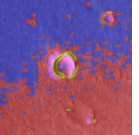
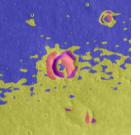
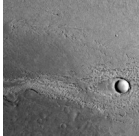
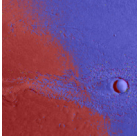
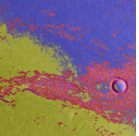
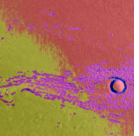
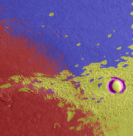
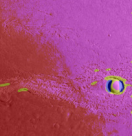
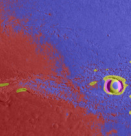
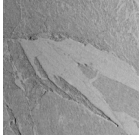
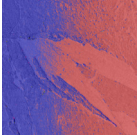
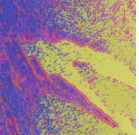
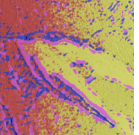
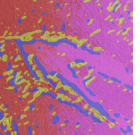
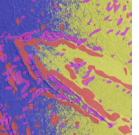
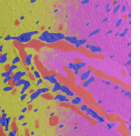
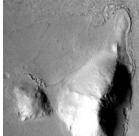
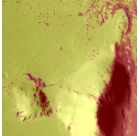
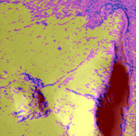
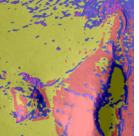
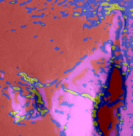
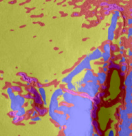
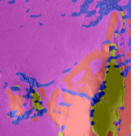
Bez.	Eingabe	$s = 0,25$	$s = 0,50$	$s = 0,75$	$s = 1,00$	$s = 1,25$	$s = 1,50$
a)							
b)							
c)							
d)							

Tabelle 5.2.2: Vergleich verschiedener Skalierungen der MR-Filterbank auf Bildern der Marsoberfläche. Die Farben der jeweiligen Cluster wurden zufällig gewählt und sagen nichts über deren Inhalt aus. Alle Bilder wurden in vier Cluster eingeteilt.

klein sind, um die Strukturen der Oberfläche zu erkennen, somit sieht diese Aufnahme aus, als wäre nur anhand ihrer Helligkeitswerten geclustert worden. Das gleiche gilt auch für die Aufnahmen b) und d).

Aufnahme c) stellt eine relativ ebenmäßige Fläche in einem Umfeld von größerer Oberfläche dar. Dort scheint es, als gäbe es nicht genug Kontrast um nach den Helligkeitswerten zu clustern, der Algorithmus erstellt eine Filterung anhand der X- und Y-Koordinaten. Bei allen Aufnahmen werden nur zwei wirkliche Cluster erstellt, die letzten zwei Cluster die erstellt werden sollten bestehen nur aus jeweils wenigen Pixeln.

Dieser Skalierungsfaktor produziert keine brauchbaren Ergebnisse.

**SKALIERUNGSFAKTOR  $s = 0,50$**  Wie zu erwarten führt die genutzte, noch relativ geringe Skalierung der Filterbank mit  $s = 0,50$  zu vielen kleineren Clustern, in diesem Fall auf Aufnahme a) deutlich erkennbar anhand der gelb markierten Flächen. Obwohl es auf den ersten Blick scheint, als würden die Kraterränder erkannt werden, werden



durch die Filterbank bei dieser Skalierung nur die dunkleren Bereiche, in diesem Fall die Schatten, welche die erhöhten Kraterränder werfen, erkannt. Auf dieser Aufnahme führt dies zwar zu einer vergleichsweise guten Approximation der Krater, sobald anderweitige Schatten hinzukommen, produziert diese Skalierung allerdings keine geeigneten Resultate.

Das selbe Phänomen tritt bei der Aufnahme in der zweiten Zeile von oben auf, auch dort wird stark anhand der Helligkeitswerte geclustert, erkennbar insbesondere an den violetten Bereichen an den Kraterrändern. Bei diesem Bild wird allerdings auch trotz der kleineren Skalierung das raue Gebiet, welches sich horizontal durch das Bild erstreckt, in ein (rot gefärbtes) Cluster eingeteilt. Auch hier werden allerdings dunklere Bereiche (wie die Kraterränder) in separate (violett gefärbte) Cluster eingeteilt.

Auf Aufnahme c) wird bei der genutzten Skalierung die feinere Fläche größtenteils als ein (gelb gefärbtes) Cluster dargestellt, separat dazu sind in rot die Übergänge zwischen den Regionen gekennzeichnet. Die größeren Flächen in den äußeren Bereichen werden nicht zuverlässig in das selbe Cluster eingeordnet.

Die letzte Testaufnahme, bestehend aus zwei Hügeln bzw. Bergen, stellt durch die stark und wenig belichteten Flächen an den Hängen eine der größten Herausforderungen für den Clusteringalgorithmus dar. An diesen Stellen ist der Kontrast des Bildes sehr gering, sodass kaum noch Texturen erkannt werden, anhand derer geclustert werden könnte. Wie zu erwarten werden die dunklen Flächen in ein einzelnes (rotes) Cluster kombiniert. Kleine Hügelketten hingegen werden zuverlässig in zusammengehörige Cluster eingeteilt, welches auf diesen Aufnahmen blau markiert ist. Die beleuchteten Seiten der Hügel werden bei dieser Skalierung leider nicht von der allgemeinen Marsoberfläche unterschieden, höchstens der Fuß des Berges kann durch die violette Färbung erahnt werden.

Dieser Skalierungsfaktor führt zu vergleichsweise guten Clusteringergebnissen.

SKALIERUNGSFAKTOR  $s = 0,75$  Dieser Skalierungsfaktor führt beim Clustering des obersten Bildes, Aufnahme a), zu Resultaten, welche vergleichbar mit denen des Faktors  $s = 0,5$  sind. Die Ränder der Krater werden allerdings getrennt in helle und dunkle Regionen eingeteilt, die Cluster im äußeren Bereich sind größer und gröber.

Auf Aufnahme b) wird die horizontale rauere Region sehr gut in ein eigenes Cluster eingeteilt, selbiges gilt für die Kraterränder. Auf dieser Aufnahme erscheint der Skalierungsfaktor  $s = 0,75$  für diese Filterbank optimal.

Die vorletzte Aufnahme, Aufnahme c), unterscheidet sich bis auf die größeren, größeren Cluster in den äußeren Bereichen nicht wesentlich von der vorherigen Skalierung.

Bei der untersten Aufnahme führt die größere Skalierung zu einem uniformen Clustering des beleuchteten Berghangs, er wird als eine große Fläche erkannt. Die dunkle Seite hingegen wird in zwei Cluster unterteilt: Die eigentliche dunkle Seite des Berges, und die Stellen, welche zu unterbelichtet sind, um aus ihnen eine Textur zu erkennen.

Insgesamt scheint sich dieser Skalierungsfaktor auch zum texturbasierten Clustering zu eignen.

SKALIERUNGSFAKTOR  $s = 1,00$  Während bei diesem Skalierungsfaktor auf Aufnahme a) der größere, mittige Krater relativ gut durch ein gelbes kreisförmiges Cluster gekennzeichnet wird, wird der kleinere Krater nicht zuverlässig erkannt. Bei diesem gehen die Cluster in umlegende Gebiete über, als wäre dieser Krater nur eine etwas rauere Oberflächenstruktur.

Bei Aufnahme b) wird wie zuvor der Krater erfolgreich separat in ein Cluster eingeteilt, die Begrenzungen der gröberen Struktur, welche sich durch das Bild zieht, geht allerdings verloren. Diese Region wird leider im rechten Teil des Bildes sehr weitläufig erkannt.

In Testbild c) werden die gröberen Hügelketten am Rand des helleren Bereiches zwar erkannt (gelb markiert), und die Übergangsstellen zwischen zwei Regionen werden in ein blaues Cluster zusammengefügt, alle anderen Regionen sind allerdings nicht voneinander getrennt worden.

Bild d) wird bei dieser Skalierung fast identisch zu der Skalierung mit  $s = 0,75$  geclustert.

Zusammengefasst ist diese Skalierung schon zu grob um so gute Ergebnisse wie die vorherigen zu erreichen.

SKALIERUNGSFAKTOR  $s = 1,25$  UND  $s = 1,50$  Da diese beiden Skalierungsfaktoren zu fast identischen Clusterverteilungen führen, werden sie in einen Abschnitt zusammengefasst.

Bei diesen Skalierungen werden im oberen, weniger rauen Bereich der Aufnahme a) die vereinzelt Krater wie gewollt erkannt, sie versagen allerdings im unteren, gröberen Bereich, wo sie nur ein großes Cluster erstellen. Der Krater wird nur anhand von starken Helligkeitsdifferenzen erkannt, welche in der gesamten Ausgabe gelb bei  $s = 1,25$ , bzw. violett bei  $s = 1,5$  markiert wurden.

Selbiges gilt im Bezug auf die starken Helligkeitsdifferenzen in Aufnahme b).

Bei Aufnahme c) wird von beiden Skalierungsfaktoren kein brauchbares Resultat produziert, man kann höchstens erahnen, dass die gelb bzw. violett gefärbten Regionen besonders kontrastarme, also glatte Flächen darstellen sollen.

Beim Clusteringergebnis der letzten Aufnahme unterschieden sich die beiden Skalierungsfaktoren erstmals stärker: Bei  $s = 1,25$  gleicht das Ergebnis dem des vorherigen Clusterings: Kleinere Bergregionen und rauere Oberflächenstrukturen werden getrennt voneinander erkannt, Flächen mit weniger oder (aufgrund von Unterbelichtung) keiner werden auch in ein großes Cluster hinzugefügt. Der Faktor  $s = 1,5$  scheint selbst bei diesem Bild zu groß zu sein, bis auf die Hügelketten mit starker Helligkeitsdifferenz wird die Aufnahme nur anhand ihrer Helligkeitswerte geclustert.

Diese Skalierungsfaktoren führen also zu Filtern, welche eine zu große Kantenlänge besitzen um brauchbare Resultate zu produzieren, da die Muster innerhalb einer Oberflächentextur eine wesentlich geringere Entfernung besitzen, nach welcher sie sich wiederholen.

Aus diesen Ergebnissen lässt sich schlussfolgern, dass sich die Skalierungsfaktoren  $s = 0,5$  und  $s = 0,75$  bei diesen Eingabedaten die besten Resultate liefert. Da durch die nachfolgende Weiterverarbeitung durch den Algorithmus nach Kanazaki eine hohe Anzahl an Clustern benötigt wird, diese aber nicht zu klein und verstreut ausfallen sollten, wird von nun an ein Skalierungsfaktor von  $s = 0,75$  genutzt.

### 5.2.3 Gewichtung der Parameter

Wie in Unterabschnitt 4.2.3 erläutert, spielt die Gewichtung der Komponenten des Datenwürfels, welcher zum Clustering durch k-Means genutzt wird eine wichtige Rolle. In den Abbildungen 5.2.3 und 5.2.4 die Resultate des Clusterings mit unterschiedlichen Werten für diese jeweiligen Gewichtungen zu sehen. Die Skalierung der Filter beträgt  $s = 0,5$ , da im vorherigen Unterabschnitt festgestellt wurde, dass dieser Wert relativ gute Ergebnisse liefert.

**GEWICHTUNG DER KOORDINATEN** Das Hinzufügen der X- und Y-Koordinaten zu dem Datenwürfel ist ursprünglich aus dem Grund geschehen, dass ein gegebener räumlicher Bezug zwischen unterschiedlichen Pixeln beim Clustering berücksichtigt werden kann (vgl. Unterabschnitt 3.2.1; [JF91]). Aus den Clusteringergebnissen in Tabelle 5.2.3 allerdings lässt sich erkennen, dass dieser räumliche Bezug in diesem Anwendungsfall nicht außerordentlich hilfreich ist: Bei einer Gewichtung von  $w_p \geq 1,33$  lässt sich eindeutig erkennen, dass der allgemeine Bodenbereich auf allen Aufnahmen zu stark anhand seiner Positionen eingeteilt wird, es entstehen Cluster in den Ecken der Aufnahmen, welche sich ins Zentrum erstrecken. Unter diesem starken Einfluss nimmt zusätzlich die relative Wichtigkeit der ähnlichen Textur ab.

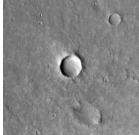
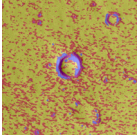
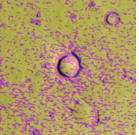
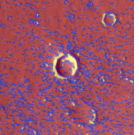
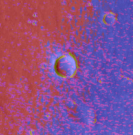
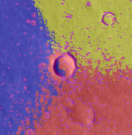
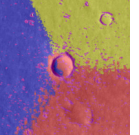
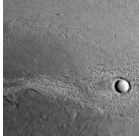
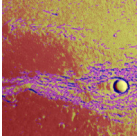
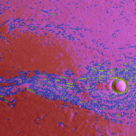
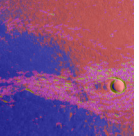
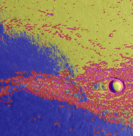
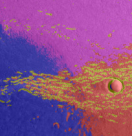
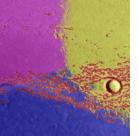
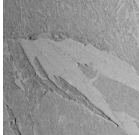
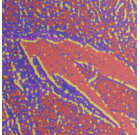
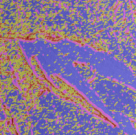
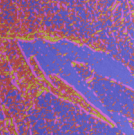
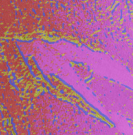
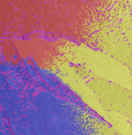
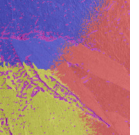
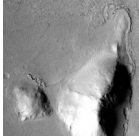
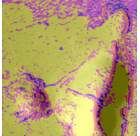
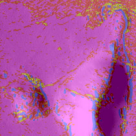
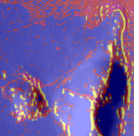
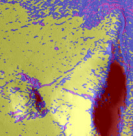
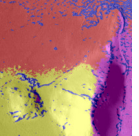
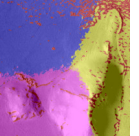
Bez.	Eingabe	$w_p = 0,00$	$w_p = 0,33$	$w_p = 0,66$	$w_p = 1,00$	$w_p = 1,33$	$w_p = 1,66$
a)							
b)							
c)							
d)							

Tabelle 5.2.3: Vergleich verschiedener Gewichtungen der Koordinaten beim Clustering der jeweiligen Eingabedateien. Die Farben der jeweiligen Cluster wurden zufällig gewählt. Alle Bilder wurden in vier Cluster eingeteilt.

Wird nun ein geringerer Wert für  $w_p$  betrachtet, so erkennt man für Werte mit  $w_p \leq 0,66$  keinen starken Unterschied bei wechselnden Gewichtungen. Dies ist insbesondere bei  $w_p = 0$  signifikant, da dort der räumliche Bezug der einzelnen Pixel zueinander bei der Berechnung des Clusterings zwar komplett ignoriert wird, das eigentliche Ergebnis sich, bis auf etwas kleinere Cluster in den merkmalsarmen Bereichen, aber kaum verändert. Dies lässt sich darauf zurückführen, dass das Hinzufügen des räumlichen Bezuges ursprünglich den Grund hatte, dass auf „normalen“ Fotografien die vorkommenden Cluster meistens nur ein einziges mal an einem räumlich begrenzten Ort vorhanden sind, so sind z. B. bei der Aufnahme aus Abbildung 3.2.3 aus Unterabschnitt 3.2.1 die kleineren Korallen im linken Hintergrund nur an dieser Stelle vorhanden und ließen sich durch ein weit ausgedehntes Cluster zusammenfassen. Bei den aktuell genutzten Aufnahmen hingegen ist dies nicht der Fall, eine auftretende Oberflächenstruktur, wie z. B. ein Krater kann an verschiedenen Stellen ohne räumlichen Bezug zueinander auftreten. Die Nutzung der Positionswerte hat allerdings einen Vorteil: Sie sorgt dafür die Clustergrößen nicht zu klein und fragmentiert wer-

den zu lassen, indem sie eine Relation mit benachbarten Pixeln erzeugt, welche der k-Means Algorithmus berücksichtigt.

Aus diesem Grund macht es Sinn, die Koordinatenbeträge der Pixel relativ gering zu gewichten, um dafür zu sorgen, dass das texturbasierte Clustering möglichst positionsinvariant operiert, aber auch nicht so gering, dass viele fragmentierte Cluster entstehen. Anhand von Tabelle 5.2.3 ist erkennbar, dass ein Wert von  $w_p = 0,66$  sich in diesem Fall sehr gut eignet, ein brauchbares Clusterergebnis zu erstellen.

**GEWICHTUNGEN DER FARB-/HELLIGKEITSWERTE** Die hier folgenden Clustering-ergebnisse wurden mit den zuvor bestimmten optimalen Parametern erstellt.

In der originalen Ausarbeitung des texturbasierten Clusterings [JF91] werden nur zwei Ebenen für die jeweiligen X- und Y-Koordinaten hinzugefügt. Es steht also an dieser Stelle offen, ob das Hinzufügen der Farb- bzw. Helligkeitsdimension zum Datenwürfel zu besseren Ergebnissen führt. Dazu wurde der vorgestellte Algorithmus mit unterschiedlichen Werten für die Gewichtung der Helligkeitswerte  $w_c$  auf den bekannten Beispielbildern ausgeführt. Die Resultate dieses Experimentes sind in Tabelle 5.2.4 dargestellt.

Auf diesen Clusterings wird schnell ersichtlich, dass eine Veränderung der Gewichtung relativ geringe Auswirkungen hat. Eine stärkere Gewichtung von  $w_c = 1,66$  besitzt auf der vierten Aufnahme den wohl größten Einfluss, da dort anhand der Über- und Unterbelichtung die beiden Bergseiten getrennt voneinander erkannt werden. Obwohl dieses Phänomen hier gute Auswirkungen hat, ist es im Allgemeinen ungewollt: Mit diesem starken Einfluss der Helligkeitswerte lernt das neuronale Netzwerk stärker mithilfe der Helligkeit statt über die Textur dieser Cluster zu lernen, was eines der größten Probleme beim Trainieren dieses Netzes ist.

Da sich durch die zusätzlichen Helligkeitswerte keine Verbesserungen der Clustering-ergebnisse feststellen lassen wird dieser Schritt von nun an entfernt, es gilt also  $w_c = 0$ .

#### 5.2.4 Anzahl der Cluster

Im vorherigen Kapitel wurde erklärt, warum sich der bis an diese Stelle genutzte Wert von vier Clustern nicht für die erfolgreiche Anwendung des vorgestellten Algorithmus eignet. Aus diesem Grund folgt nun eine Analyse, welche einen geeigneten Wert bestimmen soll.

In Tabelle 5.2.5 ist zu erkennen, dass eine geringe Anzahl an initialen Clustern zu sehr groben Resultaten führt. Dies ist darauf zurückzuführen, dass diese wenigen,

Bez.	Eingabe	$w_c = 0,00$	$w_c = 0,33$	$w_c = 0,66$	$w_c = 1,00$	$w_c = 1,33$	$w_c = 1,66$
a)							
b)							
c)							
d)							

Tabelle 5.2.4: Vergleich verschiedener Gewichtungen der Farb- bzw. Helligkeitswerte beim Clustering der jeweiligen Eingabedateien. Die Farben der jeweiligen Cluster wurden zufällig gewählt. Alle Bilder wurden in vier Cluster eingeteilt.

großen Cluster unterschiedliche Merkmale enthalten (können), welche zwar vom neuronalen Netz erkannt werden, anschließend allerdings mit dem Clusterlabel des am häufigsten erkannten Wertes überschrieben werden. Folglich gehen sehr schnell wichtige Oberflächenmerkmale verloren, was im Extremfall (vgl. Aufnahme 3) dazu führen kann, dass die gesamte Aufnahme durch nur ein großes Segment markiert wird. An dieser Stelle hilft das Abbruchkriterium der Anzahl der Cluster auch kaum, da in diesen Aufnahmen noch zwei weitere, kleinere Cluster mit einer Größe von wenigen Pixeln enthalten sind (hier nicht sichtbar).

Ein zu großer Wert für diese Initialisierung hingegen führt, wie oben beschrieben, zu Segmentzuweisungen, welche auf einer zu geringen Menge an Informationen basiert. Dieser Effekt ist auf den genutzten Aufnahmen nicht sichtbar, da diese meist relativ feine Texturen enthalten. Er wäre erst erkennbar, wenn die Länge eines Clusters geringer wäre als die Distanz, in welcher sich die zu analysierende Textur wiederholt. Außerdem führt eine höherer Anzahl von Clustern zu einer erhöhten Laufzeit der

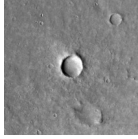
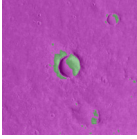
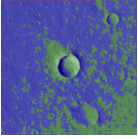
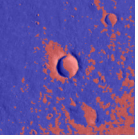
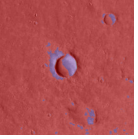
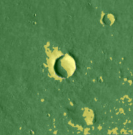
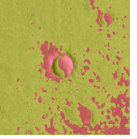
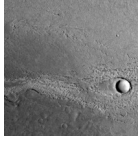
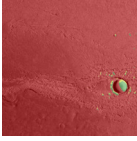
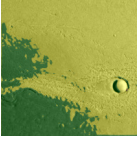
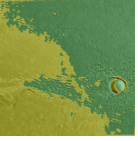
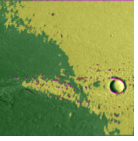
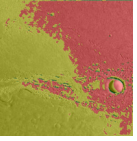
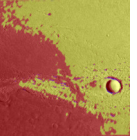
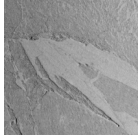
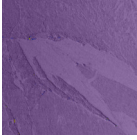
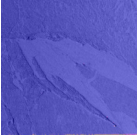
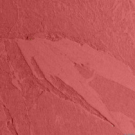
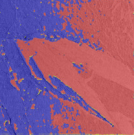
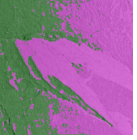
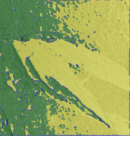
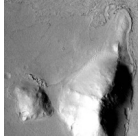
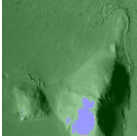
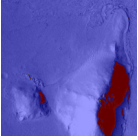
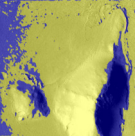
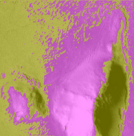
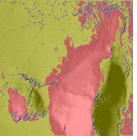
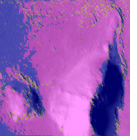
Bez.	Eingabe	n = 5	n = 10	n = 20	n = 50	n = 75	n = 100
a)							
b)							
c)							
d)							

Tabelle 5.2.5: Vergleich verschiedener Werte für die Anzahl der Cluster bei der Initialisierung. Die Farben der jeweiligen Cluster wurden zufällig gewählt.

Initialisierung, da der dort genutzte k-Means-Algorithmus eine Laufzeit besitzt, welche proportional zur Anzahl der zu erstellenden Cluster ist.

Anhand der Grafik lässt sich feststellen, dass ein Wert von  $n = 50$  für eine erfolgreiche Segmentierung ausreichend ist, da größere Werte keine sichtbare Verbesserung der Ergebnisse hervorbringen.

### 5.3 MODIFIZIERUNGEN DER NETZWERKARCHITEKTUR

#### 5.3.1 Abbruchkriterium

Eine graphische Darstellung der Änderungsrate in Abhängigkeit der Epochen des Netzwerkes ist für die vier verschiedenen Aufnahmen des letzten Abschnittes in Tabelle 5.3.1 sichtbar.

Die in [Kan18] genutzte Anzahl der Segmente eignet sich zwar als grundlegendes Abbruchkriterium, es ist allerdings sichtbar, dass dieses pro Epoche stark schwankt

und gegen unterschiedliche Werte konvergiert. Ein ähnliches Verhalten ist auch bei der Betrachtung des Loss pro Epoche sichtbar.

Es wird auch deutlich, dass sich die vorgeschlagene Methode der Nutzung der letzten 20 Verlustwerte nicht als Abbruchkriterium eignet, da diese stark zwischen unterschiedlichen Aufnahmen variiert und trotz ihres relativ großen Umfangs noch starke lokale Extremstellen beinhaltet.

Um eine Ableitung der Verlustfunktion einfach bilden zu können, muss diese durch eine stetige Funktion approximiert werden. Da die Verlustfunktion von der Stelle  $x = 0$  an immer weiter sinkt und somit gegen einen Wert von  $y = 0$  konvergiert, wird diese Approximierungsfunktion durch  $f(x) = a * \frac{1}{x} + b$  beschrieben, welche die Ableitung  $f'(x) = -a \frac{1}{x^2}$  besitzt. Die konkreten Werte für  $a$  und  $b$  werden in jeder Iteration des Netzwerkes neu berechnet, da durch jede weitere Epoche neue Messwerte hinzukommen, welche zu einer besseren Approximation führen. Außerdem wird mit dieser Verfahren erst ab der zwanzigsten Epoche begonnen, da die Verlustfunktion davor zu instabile Werte annimmt, als dass eine gute Ableitung gebildet werden kann. Zur eigentlichen Erstellung der abzuleitenden Funktion wird die Methode der kleinsten Quadrate genutzt.

Die Nutzung der Ableitung der Lossfunktion hingegen scheint zu vergleichsweise guten Ergebnissen zu führen, da deren Graphen ab etwa 100 Epochen kaum schwanken und alle gegen den selben Wert zu konvergieren scheinen. Weitere Experimente haben gezeigt dass eine Schwelle von  $f'(x) \geq -0,0025$  sich gut als Abbruchkriterium eignet. Es sollte allerdings noch ein weiteres Abbruchkriterium hinzugefügt werden, welches die Anzahl der Segmente nach unten hin beschränkt. Ansonsten kann es vorkommen, dass das gesamte Bild als nur ein einziges Segment erkennt, was zwar den Wert der Verlustfunktion senkt, aber keine nutzbaren Ergebnisse erzeugt.

Es ist auch ersichtlich, dass die Verlustfunktion alleine nicht als Abbruchkriterium funktionieren würde, da diese für verschiedene Eingabedateien sich an unterschiedliche Werte annähert – es ist also nicht möglich einen gemeinsamen Wert festzulegen, ab welchem das neuronale Netzwerk abgebrochen werden sollte.

Diese beiden Probleme bedeuten, dass wenn eins der beiden, der Wert der Verlustfunktion oder deren Änderung, als Abbruchkriterium genutzt werden, dies dazu führen würde, dass das Netzwerk zu viele oder zu wenige Segmente erzeugt, da das Verhältnis zwischen Verlust und Anzahl der Segmente für jedes Eingabebild unterschiedlich ist. Folglich wird das Abbruchkriterium bei der Anzahl der generierten Segmente belassen.



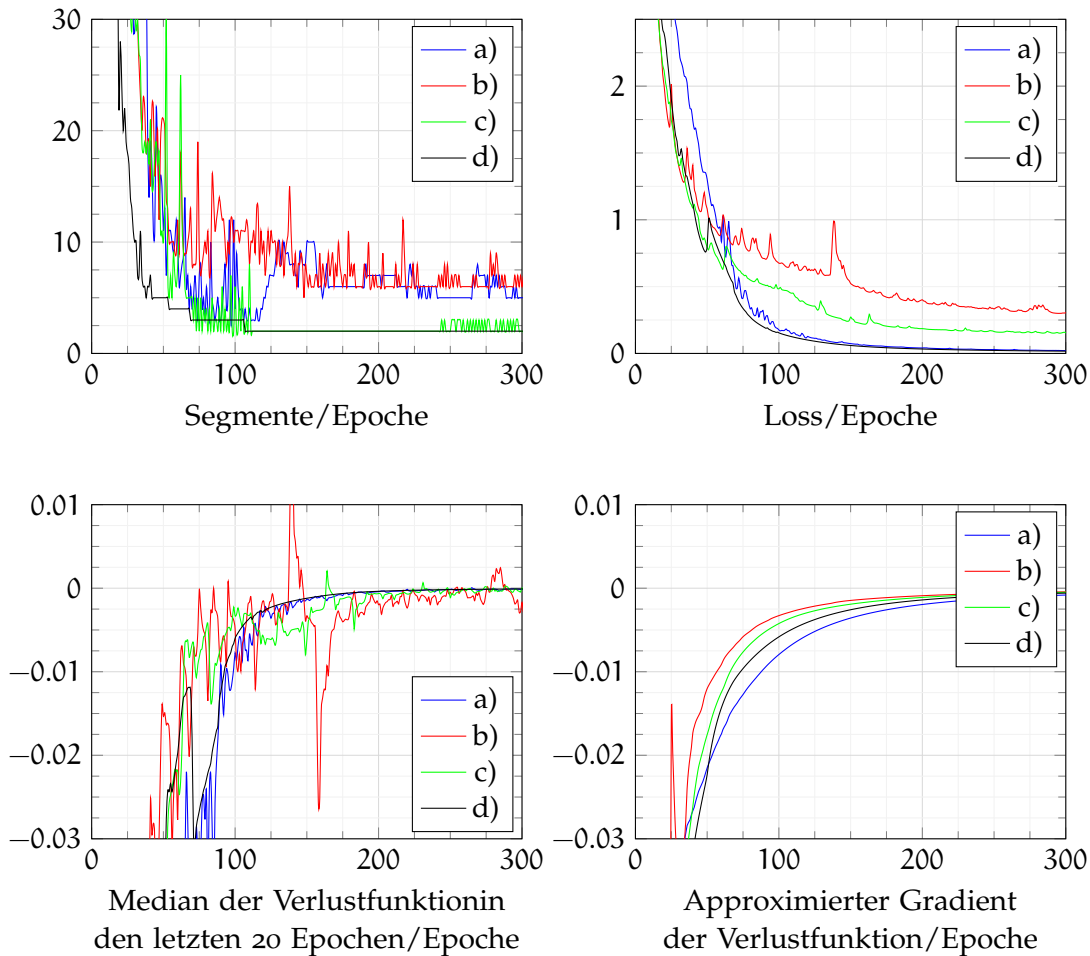


Abbildung 5.3.1: Vergleich unterschiedlicher möglicher Abbruchkriterien, jeweils bis 300 Epochen. Die Graphen sind in höherer Auflösung in Appendix A.1.1 zu finden. Die Legenden geben die zugehörige Marsaufnahme aus den vorherigen Abschnitten an.

### 5.3.2 Aktivierungsfunktionen

Im Folgenden wird überprüft, welche Aktivierungsfunktion sich am ehesten dazu eignet, mithilfe des vorgestellten Algorithmus eine gute Segmentierung zu erstellen. Dazu werden sowohl die ReLU-Funktion, die Sigmoid-Funktion, die tanh-Funktion, als auch das Entfernen jeglicher Aktivierungsfunktionen untersucht. Genutzt werden

die bereits verbesserten weiteren Hyperparameter. Die Ergebnisse befinden sich in Tabelle 5.3.1.

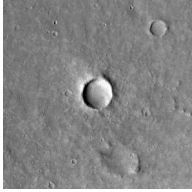
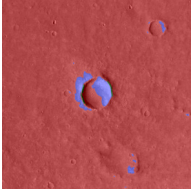
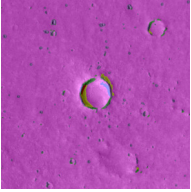
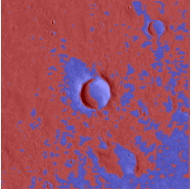
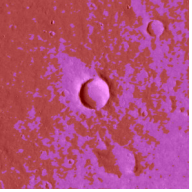
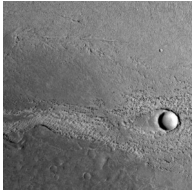
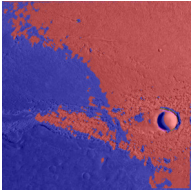
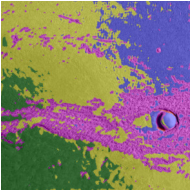
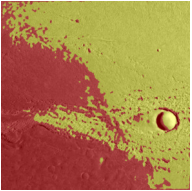
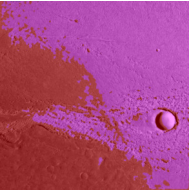
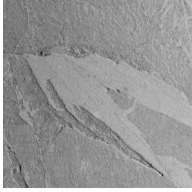
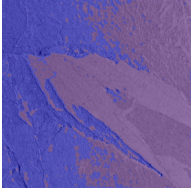
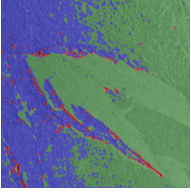
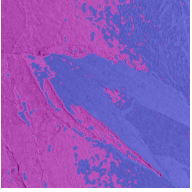
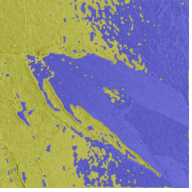
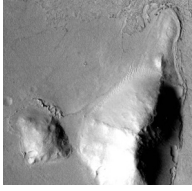
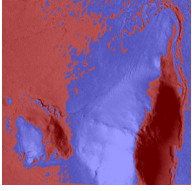
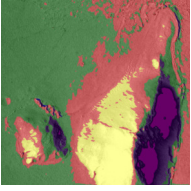
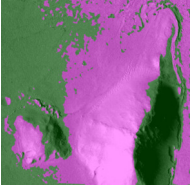
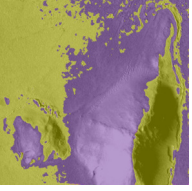
Bez.	Eingabe	Ohne Aktivierungsfunktion	ReLU-Aktivierung	Sigmoid-Aktivierung	tanh-Aktivierung
a)					
b)					
c)					
d)					

Tabelle 5.3.1: Vergleich der Auswirkungen der Nutzung verschiedener Aktivierungsfunktionen. Die Farben der jeweiligen Cluster wurden zufällig gewählt. Hinweis: In der dritten Aufnahme von oben ist das helle Segment in der Mitte mit dem gleichen Label markiert wie der jeweils rechte Rand. Durch die Darstellung ist dies nicht eindeutig zu erkennen.

Aus diesen lässt sich relativ schnell erkennen, dass sowohl die Sigmoid-, als auch die tanh-Funktion als Aktivierungsfunktion keinen merklichen Einfluss auf die entstehende Segmentierung haben. Lediglich bei der ReLU-Aktivierung werden deutliche Verbesserungen gegenüber der Segmentierung ohne Aktivierungsfunktion sichtbar.

So ist direkt in Aufnahme a) erkennbar, dass das neuronale Netzwerk ein Segment entlang der einzelnen, kleineren Kraterränder erstellt, bei den anderen Aktivierungsfunktionen gehen diese vollständig verloren, da dort hauptsächlich der Helligkeitswert zur Segmentierung genutzt wird. Auch in Aufnahme b) zeigen sich deutliche Verbesserungen, der raue Bereich, welcher sich horizontal durch die Aufnahme zieht wird mithilfe der ReLU-Funktion auf der vollen Länge erkannt, statt nur in den rechten zwei Dritteln. Des Weiteren wird dort der kleinere Bereich mit einer ähnlichen Textur im oberen Bereich des Bildes sehr gut erkannt und in das selbe Segment eingeordnet wie die größere Struktur. Auch hier werden die Kraterränder gut in ein im Bild einzigartiges Cluster eingeteilt. In Aufnahme c) ist die Nutzung der ReLU-Funktion vergleichbar mit den anderen Aktivierungen, allerdings werden hier die Begrenzungen der feineren, helleren Struktur in ein rotes Segment eingeteilt. Dieses ist sonst nicht vorhanden. Auch auf der letzten Aufnahme, Aufnahme d), ist sichtbar, dass das Netzwerk dank der ReLU-Aktivierung nicht nur nach den Helligkeitswerten segmentiert, sondern wie gewünscht nach der Oberflächenstruktur: Der Bereich um den Berg herum wird in ein rotes Segment eingeteilt, außerdem werden die Berghänge trotz ihrer starken Licht-/Schattenregionen getrennt zu ihrem Umfeld erkannt.

An dieser Stelle existieren nun zwei unterschiedliche Theorien, warum die Nutzung der ReLU-Aktivierung wesentlich bessere Ergebnisse produziert als sämtliche Alternativen:

1. Die Verbesserung ist darauf zurückzuführen, dass in dem hier genutzten Algorithmus die Anzahl der Iterationen im Vergleich zu den meisten anderen neuronalen Netzen relativ gering ist, da sie hier nur etwa 100 bis 300 beträgt. ReLU ist auf dieser geringen Epochenanzahl effizienter darin, unbrauchbare Merkmale auszufiltern.
2. Die Verbesserung ist darauf zurückzuführen, dass die ReLU-Aktivierung negative Eingaben als Nullwert weitergibt. Lernt also bspw. eine konvolutionelle Schicht statt den gewünschten Parametern zur texturbasierten Segmentierung Parameter, welche die Textur ignorieren (also bspw. nur die Helligkeitsinformationen betrachten), so werden diese nicht negativ gewichtet. Stattdessen würden die Konvolutionsfilter, welche zu diesem Punkt führen mit 0 multipliziert, und somit im Bezug auf das Endergebnis verworfen, statt mit einem negativen Faktor darauf einzuwirken. Dieses Phänomen wird im Allgemeinen als *dying ReLU* oder *dead ReLU* bezeichnet.

Um nun herauszufinden, welche Erklärung die Ursache für die auftretenden Ergebnisse liefert, wird der Algorithmus mit der Leaky ReLU-Aktivierungsfunktion

getestet. Diese Aktivierungsfunktion verhält sich wie die ReLU-Aktivierung, hat im Bereich  $x < 0$  allerdings eine sehr geringe Steigung (hier 0.01), eben um das *dying ReLU*-Problem zu bekämpfen: [HZRS15]

$$f(x) = \begin{cases} x & \text{wenn } x > 0 \\ 0.01x & \text{andernfalls} \end{cases} \quad (5.3.1)$$

Diese Aktivierungsfunktion ist in Abbildung 5.3.2 dargestellt.

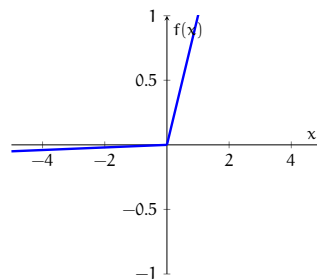


Abbildung 5.3.2: Leaky ReLU

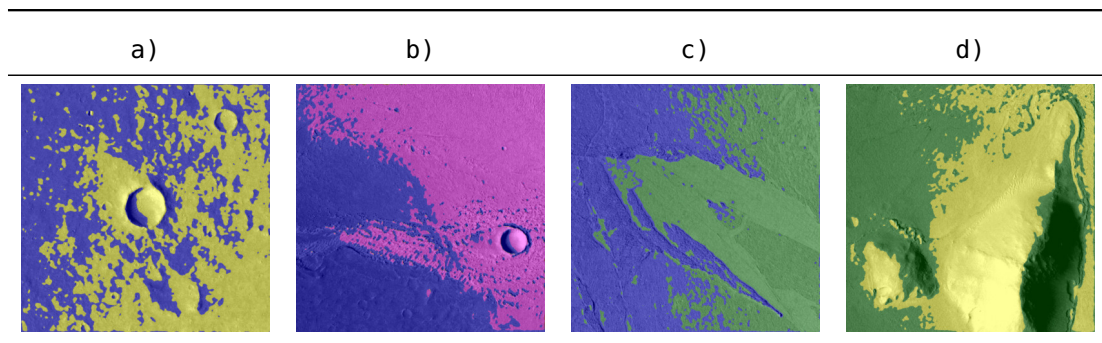


Tabelle 5.3.2: Nutzung der Leaky ReLU-Aktivierungsfunktion. Die Farben der jeweiligen Cluster wurden zufällig gewählt.

Anhand der Ergebnisse der Anwendung des Algorithmus mithilfe der Leaky ReLU-Funktion in Tabelle 5.3.2 lässt sich erkennen, dass sich die Segmentierung praktisch nicht von der Segmentierung mithilfe der Sigmoid- oder der tanh-Funktion unterscheidet. Daraus lässt sich nun schließen, dass die erste Theorie für die Ursache der

besseren Ergebnisse falsch ist, da auch die nun genutzte Aktivierung in etwa die selbe Geschwindigkeit besitzt wie die normale ReLU-Aktivierung. Selbst wenn diese Aktivierung langsamer sein sollte, so müssen zumindest Ergebnisse produziert werden, welche unterschiedlich zu den bisher erstellten Segmentierungen sind. Somit ist die zweite Theorie für die Verbesserung durch ReLU korrekt, da diese darauf basiert, dass sich das *dying ReLU*-Problem zunutze gemacht wird.

### 5.3.3 *Pooling Layer*

Die Resultate nach der Implementierung von nur einer einzigen Max-Pooling Layer mit den Kernel-Größen  $F = (2, 2)$  und  $F = (4, 4)$  sind in der folgenden Tabelle 5.3.3 aufgeführt. Das Hinzufügen von weiteren Pooling-Schichten und die Nutzung von größeren Kernen führt zu Ergebnissen, auf welchen keine sinnvolle Segmentierung erkennbar war und ist aus diesem Grund nicht an dieser Stelle aufgeführt.

Auf Aufnahme a) zeigt sich der größte Nachteil der Pooling-Schichten. Die kleinen Krater in den äußeren Bereichen werden bei  $F = (2, 2)$  nicht so zuverlässig wie zuvor erkannt, bei einer Kernelgröße von  $F = (4, 4)$  werden nur noch sehr wenige kleinere Krater in separate Segmente eingeteilt. Auch auf den weiteren Eingabedateien ist keine Optimierung der Resultate durch die Nutzung der Max-Pooling-Schichten sichtbar. Nach mehreren Testdurchläufen stellt sich heraus dass die meisten der dortigen Unterschiede zwischen den einzelnen Kernelgrößen auf die zufälligen Initialisierungen des Clusterings und des Neuronalen Netzes zurückzuführen sind. Ansonsten fällt erneut auf, dass kleinere Regionen nicht mehr sicher erkannt werden. Diese Nachteile sind auch durch den Vorteil von Pooling-Schichten, die erhöhte Performance des Netzes, nicht auszugleichen. Ein weiteres Argument gegen die Nutzung von Pooling Layern ist, dass diese die Auflösung verringern. Sie werden meistens zur Objekterkennung genutzt, bei welcher es meistens irrelevant ist, bei welchen Koordinaten genau ein Gegenstand beginnt oder endet. Hier allerdings ist die Problemstellung ein Segmentierungsproblem, die Marsoberfläche soll möglichst genau segmentiert werden.

Wie zu erkennen ist, bringt die Nutzung einer Pooling-Schicht bei diesem Anwendungsfall keine sichtbaren Vorteile mit sich und ist folglich ungeeignet.

### 5.3.4 *Fully Connected Layers*

Anhand der Segmentierungen in Tabelle 5.3.4 ist erkennbar, dass die Nutzung von  $n_{fc}$  Fully Connected Layer bei diesem Anwendungsfall keine Vorteile mit sich bringt:

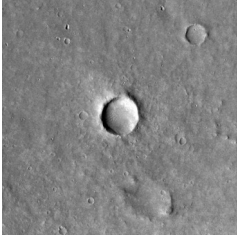
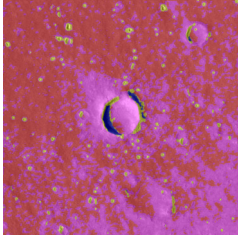
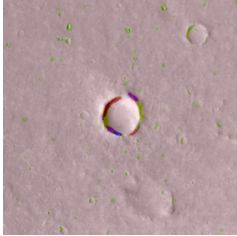
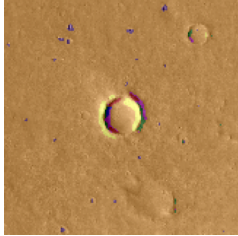
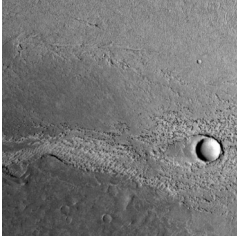
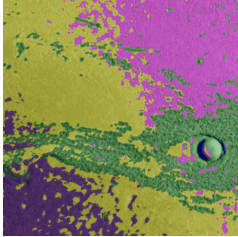
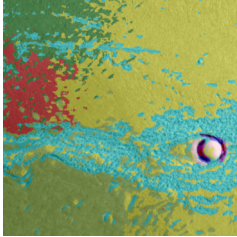
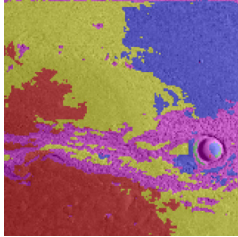
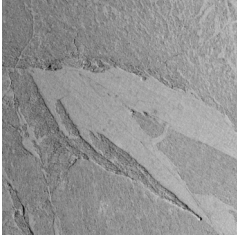
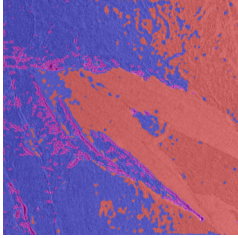
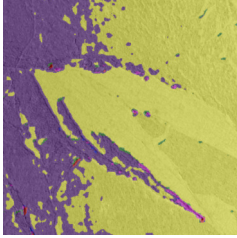
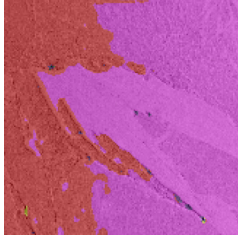
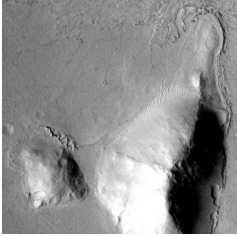
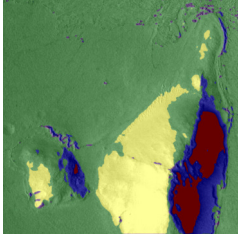
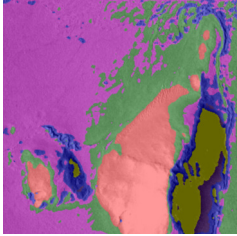
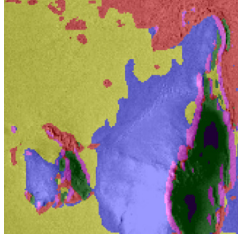
Bez.	Eingabe	$F = (1, 1)$	$F = (2, 2)$	$F = (4, 4)$
a)				
b)				
c)				
d)				

Tabelle 5.3.3: Vergleich der Auswirkungen der Nutzung von Pooling-Schichten. Die Farben der jeweiligen Cluster wurden zufällig gewählt.

Sie führt eher dazu, dass die Eingabedateien nach ihrer Helligkeit segmentiert werden, da innerhalb dieser Fully Connected Layers kein Konzept der Oberflächenstruktur entsteht.

Dies ist insbesondere auf Aufnahme d) offensichtlich, da dort die vorherige Segmentierung in die zwei Berghänge ersetzt wird durch eine Segmentierung, welche die stark

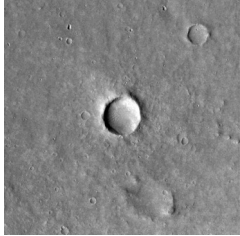
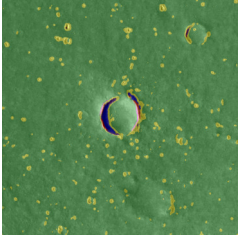
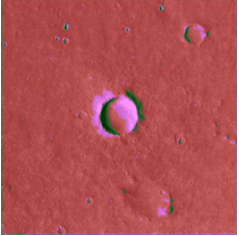
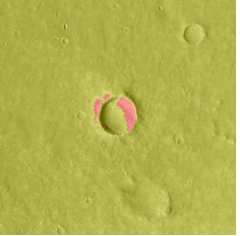
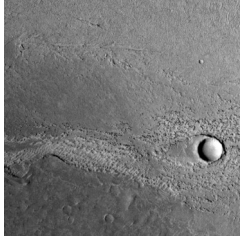
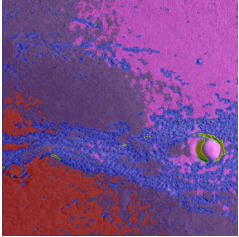
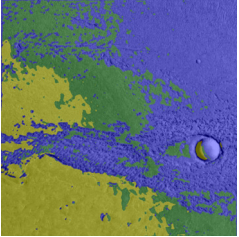
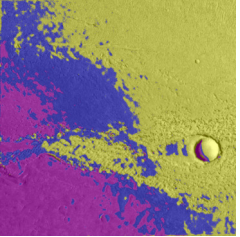
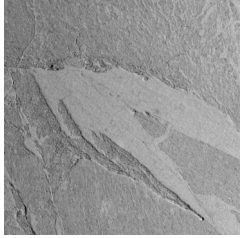
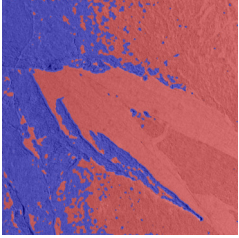
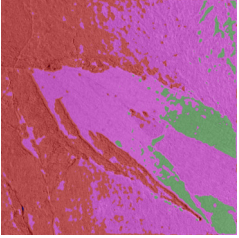
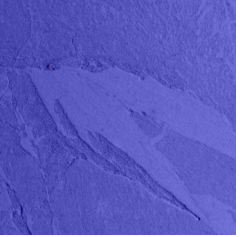
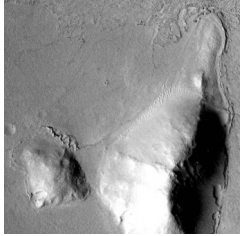
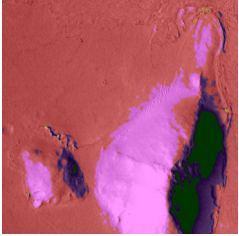
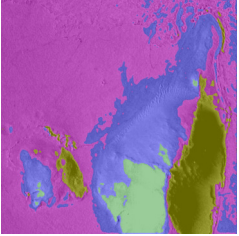
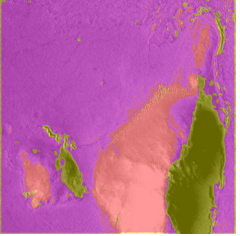
Bez.	Eingabe	$n_{fc} = 0$	$n_{fc} = 1$	$n_{fc} = 2$
a)				
b)				
c)				
d)				

Tabelle 5.3.4: Vergleich der Auswirkungen der Nutzung von Fully-Connected-Layers. Die Farben der jeweiligen Cluster wurden zufällig gewählt.

belichteten und wenig belichteten Regionen separat von ihrem Umfeld einordnet. Ein weiteres Beispiel für diese Funktionsweise ist auch in Abbildung a) sichtbar. Während die Segmentierung ohne die Nutzung von Fully Connected Layers die Ränder der beiden größeren Krater in ein helligkeitsunabhängiges Segment einteilt, wird diese bereits bei der Nutzung von nur einer FC-Schicht durch zwei Cluster ersetzt: Eins,

welches helle Regionen, und eins, welches dunkle Regionen kennzeichnet. Des Weiteren gehen durch die Nutzung dieser Fully Connected Layers Informationen über einen Großteil der kleineren Krater in dieser Aufnahme verloren.

Diese Phänomene rühren daher, dass Fully Connected Layer meistens nur als Klassifikator am Ende eines ansonsten vollständigen neuronalen Netzes genutzt werden. Dort werden sie dazu genutzt, anhand der ihnen von den konvolutionellen Schichten gelieferten Merkmalsräume zu entscheiden, wie ein gegebener Punkt klassifiziert werden soll. In der hier beschriebenen Anwendung allerdings, reicht es aus, die unterschiedlichen Merkmale voneinander zu unterscheiden, diese müssen folglich nicht weiter klassifiziert werden.

Aus diesen Umständen lässt sich schließen, dass die Nutzung dieser Fully Connected Layers, welche in anderen Anwendungsgebieten viele Vorteile hat, hier eher ungeeignet ist.

### 5.3.5 Anzahl der Konvolutionsschichten

In der originalen Implementierung war die Anzahl der Konvolutionsschichten (gefolgt von den Aktivierungs- und Batch-Normalisierungs-Schichten) dynamisch, mit einem Standardwert von 1+2 Schichten. Nun wird über die Resultate in Tabelle 5.3.5 überprüft, welchen Einfluss eine Veränderung dieses Parameters auf die entstehende Segmentierung hat.

Insgesamt fällt hier kein signifikanter Unterschied zwischen den einzelnen Aufnahmen auf: Ein Großteil der Verbesserungen und Verschlechterungen bei der Modifikation der Anzahl der Konvolutionsschichten ist schlichtweg auf die unterschiedliche Initialisierung des texturbasierten Clusterings und des Netzwerkes selbst zurückzuführen. Erkennbar ist dies daran, dass in der zweiten Aufnahme, bei fünf Konvolutionsschichten eine vergleichbar schlechte Segmentierung generiert wird, diese wird allerdings wieder besser, egal ob eine Konvolutionsschicht hinzugefügt oder entfernt wird.

In Aufnahme c) verändert sich die Segmentierung zwischen  $n_{\text{conv}} = 2$  und  $n_{\text{conv}} = 5$  kaum, lediglich bei  $n_{\text{conv}} = 6$  wird der rechte Bereich als ein weiteres Segment erkannt. Dieser Effekt erschien über mehrere Ausführungen des Experimentes hinweg auch nicht zuverlässig, es ist daher auch nur mit einer unterschiedlichen Initialisierung begründet.



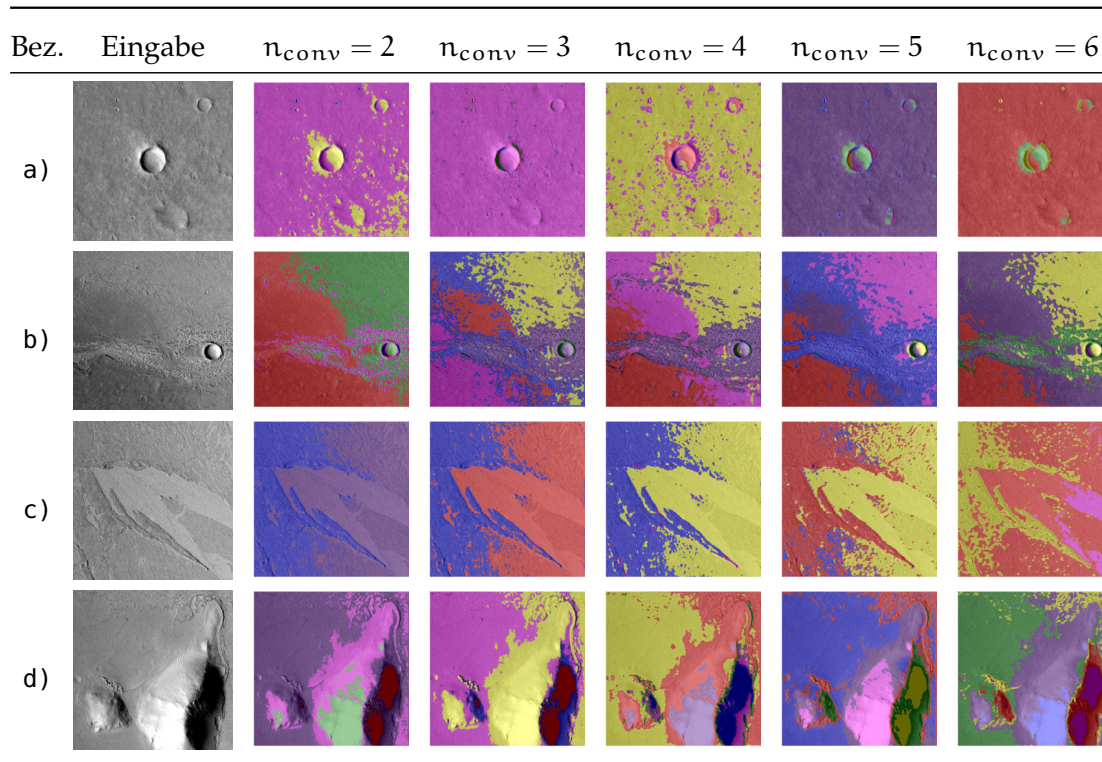


Tabelle 5.3.5: Vergleich der Auswirkungen der Änderung der Anzahl an Konvolutionsschichten. Die Farben der jeweiligen Cluster wurden zufällig gewählt.

### 5.3.6 Anpassungen zur Segmentierung von mehrfarbigen Fotografien

Wie in Kapitel 4 erläutert, können nicht alle der bisher bestimmten Hyperparameter dazu genutzt werden, auch Aufnahmen aus dem BSDS500-Datensatz zu segmentieren. Daher wurde der gesamte Prozess des Abschnittes 5.2 für Aufnahmen dieses Datensatzes wiederholt. Ein großer Vorteil bei der Anpassung der Parameter für diesen Datensatz besteht daraus, dass für den BSDS500-Datensatz Ground Truths existieren. Die Parameter wurden also nicht nach einer subjektiven Ansicht der jeweiligen Ergebnisse angepasst, sondern danach, welche Eingaben die besten Werte für die Variation of Interest und den Probabilistic Rand Index erzielte. Der Algorithmus wurde mit den aus diesen Experimenten besten Hyperparametern angewandt. Die Ergebnisse sind in Unterabschnitt 5.5.2 dargestellt.

Die folgende Kombination an Parametern hat die Resultate mit dem besten PRI (vgl. Unterabschnitt 5.4.2) erzielt:

- Gewichtung der Positionswerte:  $w_p = 1,33$
- Gewichtung der Farbwerte:  $w_c = 1,33$
- Anzahl der Cluster:  $n = 35$
- Abbruchkriterium:  $f'(x) \geq -0,0018$

Alle nicht aufgeführten Parameter entsprechen den bisher bestimmten Optimalwerten bei der Anwendung des Algorithmus auf die Marsoberfläche.

Auf eine genaue Analyse der Auswirkungen der Anpassung der einzelnen Parameter wird an dieser Stelle bewusst verzichtet, da dies außerhalb des Rahmens dieser Arbeit liegt.

#### 5.4 METRIKEN

Im Folgenden wird beschrieben, wie der vorgestellte Algorithmus evaluiert werden kann. Dabei kommt es darauf an, dass Metriken genutzt werden, welche auch von anderen, ähnlichen Algorithmen genutzt werden, damit ein entsprechender Vergleich möglich ist. Dies beinhaltet, dass die Resultate der Segmentierung der Marsoberfläche von Hand weiter verarbeitet werden müssen, dies wird im Paragraph „Post-Processing der Resultate“ genauer erläutert.

Die Evaluation (wie die folgenden Resultate, der Vergleich und die Diskussion) ist aufgeteilt in jeweils zwei Unterabschnitt: Zum einen wird auf das ursprüngliche, domänenspezifische Ziel eingegangen, die Segmentierung von Aufnahmen der Marsoberfläche. Anschließend folgt je ein Unterabschnitt, in welchem der Algorithmus bei der Anwendung auf einen allgemeineren Datensatz genauer betrachtet wird.

##### 5.4.1 Domänenspezifisch

**POST-PROCESSING DER RESULTATE** Wie in Unterabschnitt 5.1.1 beschrieben, tritt an der Stelle der Evaluation der domänenspezifischen Resultate das Problem der fehlenden Ground Truth auf. Der genutzte Mars Krater Katalog [RH12] enthält lediglich Informationen, ob ein gewisser Pixel der jeweiligen Aufnahme einem Krater angehört oder nicht, während der vorgestellte Algorithmus das Eingabebild in unterschiedliche Segmente aufteilt. Da dieser Algorithmus unüberwacht lernt, existieren zu den jeweiligen Segmenten keine Klassenzuweisungen. Aus diesem Grund müssen die produzierten Ergebnisse manuell konvertiert werden, um mit der vorhandenen Ground Truth verglichen werden zu können.

Um möglichst neutrale Konvertierungsergebnisse zu erhalten, wird zu diesem Zweck lediglich die Ausgabe des Algorithmus betrachtet. Andernfalls könnte es passieren, dass die Person, welche die Segmentierungen weiterverarbeitet, unterbewusst nach Stellen Ausschau hält, an welcher in der originalen Aufnahme oder in der Ground Truth Krater vorhanden sind. Zur eigentlichen Konvertierung wird die erzeugte Segmentierung an jenen Stellen mit kreisförmige Labeln überlagert, an welchen sie auf das Vorhandensein eines Kraters hindeutet. Die Tatsache, dass in (fast) allen Aufnahmen die allgemeine Marsoberfläche separat von den speziellen Merkmalen erkannt wird, vereinfacht diesen Prozess.

Damit ein Segment auf einem Krater hindeutet und als solcher markiert wird, gelten folgende Regeln:

1. Das Segment muss kreisförmig oder ellipsenförmig sein.
2. Es ist irrelevant, ob das Segment gefüllt ist, oder lediglich die Ränder eines Kreises markiert wurden, solange diese zu mehr als 50% abgebildet sind.
3. Wird ein Kraterrand erkannt, so darf dieser aus mehreren unterschiedlichen Segmenten zusammengesetzt sein. Dies ist damit begründet, dass ein gewisser Teil der Krater an unterschiedlichen Teilen der Ränder unterschiedliche Strukturen aufweist. Mit dieser Regel wird dem entgegengewirkt.
4. Alle manuell erzeugten Segmente sind perfekt kreisförmig, es ist irrelevant ob das unterliegende Segment nicht perfekt rund ist. Dies ist eine Limitierung der Ground Truth.
5. Es werden keine konkaven Segmente markiert.
6. Krater mit einem Durchmesser von unter 20 px werden ignoriert, da diese in der Ground Truth auch nicht erscheinen. (Die Ground Truth enthält nur Krater mit einem Durchmesser von  $\geq 1$  km.)
7. Krater dürfen sich teilweise überlappen.
8. Krater müssen nicht vollständig im Bildbereich liegen. Dies hat die Ursache, dass die Eingabedatei aufgrund technischer Limitationen in kleinere Aufnahmen eingeteilt werden musste, bevor sie den vorgestellten Prozess durchlaufen konnte.
9. Krater, welche aufgrund der Aufteilung des Eingabebildes abgeschnitten sind werden vervollständigt

10. Ein kreisförmiges Segment wird nicht markiert, wenn es offensichtlich ist, dass dies ein Teil einer größeren Struktur wie einer Hügelkette ist.

Diese Regeln führen zu einer möglichst neutralen Segmentierung, da sie unabhängig von der originalen Eingabedatei operieren und auch maschinell umsetzbar sind, d. h. sie beruhen nicht auf menschlicher Subjektivität, sie stellen allerdings eben durch diese menschliche Interaktion eine weitere Fehlerquelle dar.

**F1-SCORE** Der F<sub>1</sub>-Score ist eine Metrik zur Evaluation der Genauigkeit eines binären Klassifikators gegenüber einer Ground Truth. Er ist definiert als: [Chi92]

$$F_1 = 2 \frac{P \cdot R}{P + R} \quad (5.4.1)$$

Dabei sind P die Genauigkeit und R die Trefferquote:

$$P = \frac{tp}{tp + fp} \quad (5.4.2)$$

$$R = \frac{tp}{tp + fn} \quad (5.4.3)$$

Es sind in diesem Anwendungsfall:

- True Positives tp: Die Anzahl der Pixel, welche als einem Krater zugehörig erkannt wurden, welche auch in der Ground Truth einem Krater angehören.
- False Positives fp: Die Anzahl der Pixel, welche als einem Krater zugehörig erkannt wurden, welche allerdings in der Ground Truth keinem Krater angehören.
- True Negatives tn: Die Anzahl der Pixel, welche als keinem Krater zugehörig erkannt wurden, welche auch in der Ground Truth keinem Krater angehören.
- False Negatives fn: Die Anzahl der Pixel, welche als keinem Krater zugehörig erkannt wurden, welche allerdings in der Ground Truth einem Krater angehören.

#### 5.4.2 Allgemein

Zur Evaluation der hier erarbeiteten Methode zur Bildsegmentierung werden zwei unterschiedliche Metriken benutzt. Diese wurden ausgewählt, da sie zur Evaluation anderer Segmentierungsalgorithmen auf dem BSDS-500-Datensatz [AMFM11] genutzt wurden und somit bereits Referenzwerte existieren, die dabei helfen, die hier gewonnenen Ergebnisse im weiteren Kontext besser einordnen zu können. [AMFM10]

**PROBABLISTIC RAND INDEX** Der Probablistic Rand Index (PRI) ist ein Maß, welches die Ähnlichkeit zwischen einer unüberwachten Segmentierung  $S_{\text{test}}$  und einer Menge von  $K$  Referenzsegmentierungen  $S_k$ , den Ground Truths, angibt. Dabei handelt es sich jeweils um die Aufteilung der selben Eingabedatei  $X = \{x_1, \dots, x_N\}$  bestehend aus  $N$  Pixeln. Das Segment, in welches ein Pixel  $p_i$  in Segmentierung  $S$  eingeordnet wurde ist dargestellt durch  $l_i^S$ . Der Probabilistic Rand Index ist definiert als: [UPHo7]

$$\text{PRI}(S_{\text{test}}, \{S_k\}) = \frac{1}{\binom{N}{2}} \sum_{\substack{i,j \\ i < j}} (c_{ij} p_{ij} + (1 - c_{ij}) (1 - p_{ij})) \quad (5.4.4)$$

Es gilt:

$$c_{ij} = \begin{cases} 1 & \text{wenn } l_i^{S_{\text{test}}} = l_j^{S_{\text{test}}} \\ 0 & \text{andernfalls} \end{cases} \quad (5.4.5)$$

Außerdem gibt  $p_{ij}$  die Bernoulli-Verteilung an, nach welcher die Pixel  $x_i$  und  $x_j$  in den unterschiedlichen Ground Truths in das selbe Segment eingeordnet wurden. Umgeschrieben bedeutet dies:

$$p_{ij} = \frac{1}{K} \sum_{k \in K} d_{k,ij} \quad (5.4.6)$$

Mit

$$d_{k,ij} = \begin{cases} 1 & \text{wenn } l_i^{S_k} = l_j^{S_k} \\ 0 & \text{andernfalls} \end{cases} \quad (5.4.7)$$

Da  $c_{ij} \in \{0, 1\}$ , lässt sich Gleichung 5.4.4 auch schreiben als: [UPHo7]

$$\text{PRI}(S_{\text{test}}, \{S_k\}) = \frac{1}{\binom{N}{2}} \sum_{\substack{i,j \\ i < j}} (p_{ij}^{c_{ij}} (1 - p_{ij})^{1-c_{ij}}) \quad (5.4.8)$$

Ersetzt man nun in dieser Gleichung  $p_{ij}$  durch den Term aus Gleichung 5.4.7 und formt um so erhält man:

$$\text{PRI}(S_{\text{test}}, \{S_k\}) = \frac{1}{K} \sum_{k \in K} \left[ \sum_{\substack{i,j \\ i < j}} \frac{(d_{k,ij}^{c_{ij}} (1 - d_{k,ij})^{1-c_{ij}})}{\binom{N}{2}} \right] \quad (5.4.9)$$

Der gewöhnliche Rand Index kann beschrieben werden durch:

$$\text{RI}(S_1, S_2) = \frac{(d_{1,ij}^{d_{2,ij}} (1 - d_{1,ij})^{1-d_{2,ij}})}{\binom{N}{2}} \quad (5.4.10)$$

Es fällt auf, dass dies gleich dem Term in den eckigen Klammern in Gleichung 5.4.9 ist. Somit gilt für den Probabilistic Rand Index:

$$\text{PRI}(S_{\text{test}}, \{S_k\}) = \frac{1}{K} \sum_{k \in K} \text{RI}(S_{\text{test}}, S_k) \quad (5.4.11)$$

Der Probabilistic Rand Index ist also gleich dem Median aller gewöhnlichen Rand Indexe zwischen der zu testenden Segmentierung  $S_{\text{test}}$  und der Menge der Ground Truths  $S_k$ .

Wie der Rand Index nimmt der Probabilistic Rand Index Werte aus dem Wertebereich  $[0, 1]$  an. Dabei indiziert 0 gegenteilige Segmentierungen, 1 bedeutet alle Segmentierungen gleichen sich. [UPHo7]

**VARIATION OF INFORMATION** Die Variation of Information, aus [Meio3] stammend, wurde speziell dafür geschaffen, mehrere Clusterings miteinander vergleichen zu können. Sie ist für zwei Partitionen  $X$  und  $Y$  der selben Menge  $A$  definiert durch:

$$\text{VI}(X, Y) = - \sum_{ij} r_{ij} \left( \log \left( \frac{r_{ij}}{p_i} \right) + \log \left( \frac{r_{ij}}{q_j} \right) \right) \quad (5.4.12)$$

Es gelten:

$$n = \sum_i |X_i| = \sum_j |Y_j| = |A| \quad (5.4.13)$$

$$p_i = |X_i| / n \quad (5.4.14)$$

$$q_j = |Y_j| / n \quad (5.4.15)$$

$$r_{ij} = |X_i \cap Y_j| / n \quad (5.4.16)$$

$$(5.4.17)$$

Da bei dem genutzten Datensatz mehrere Ground Truths vorhanden sind, wird der Durchschnitt der jeweiligen VOIs als Resultat genutzt.

## 5.5 EVALUATION

### 5.5.1 Domänenspezifisch

Nach der Anwendung des in Unterabschnitt 5.4.1 beschriebenen Verfahrens ergibt sich ein  $F_1$ -Score von  $F_1 = 0,6374$ , ein Teil der dazugehörigen Aufnahmen und deren Segmentierungen sind in Tabelle 5.5.1 zu sehen. Dort ist zu erkennen, dass die Kraterränder auf den segmentierten Aufnahmen leicht zu erkennen sind, da der allgemeine Hintergrund erfolgreich als solcher erkannt wird und wie erwünscht einfarbig markiert wird. Außerdem fällt auf, dass sich die Kriterien eines Krater der Ground Truth von denen in Unterabschnitt 5.4.1 unterscheiden: Insbesondere auf Aufnahme c) fällt auf, dass bspw. die kurvige, in der erstellten Segmentierung als größere blau markierte Fläche sichtbare Region im rechten Teil es Bildes zwar in der Ground Truth als Krater gekennzeichnet ist, dies allerdings den aufgestellten Regeln widerspricht, nach welchen mindestens 50% eines Kraterrandes ersichtlich sein müssen.

### 5.5.2 Allgemein

Durch die in Unterabschnitt 5.4.2 beschriebenen Methoden wurden als resultierende Metriken für die Anwendung des erarbeiteten Algorithmus auf den gesamten Validierungsdatensatz des BSDS-500 ein Probabilistic Rand Index von  $PRI = 0,73$  und eine Variation of Interest von  $VOI = 2,88$  erreicht. Diese Werte geben das durchschnittliche Resultat aus fünf Testläufen an, da diese Ergebnisse abhängig von der zufälligen Initialisierung des k-Means-Algorithmus und des neuronalen Netzes sind.

Eine Auswahl der resultierten Segmentierungen befindet sich in Tabelle 5.5.2.

## 5.6 VERGLEICH

### 5.6.1 Domänenspezifisch

Der in dieser Arbeit erzielte  $F_1$ -Score von  $F_1 = 0,6374$  lässt sich nur schwer mit alternativen Algorithmen zur Kratererkennung vergleichen. Dies liegt einerseits daran, dass nicht alle Algorithmen auf dem hier genutzten Robbins-Datensatz [RH12] angewandt

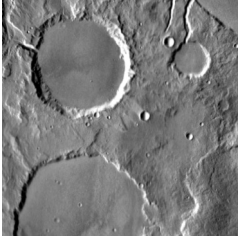
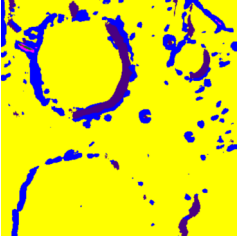
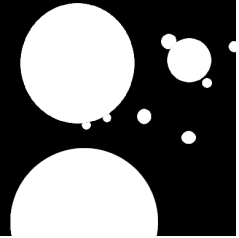
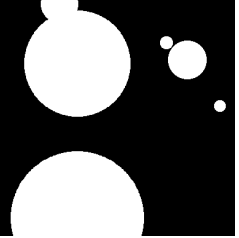

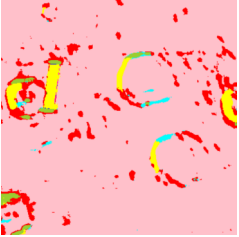
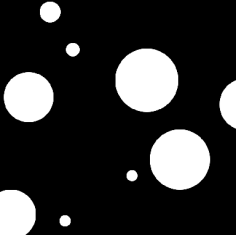
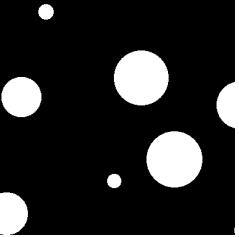
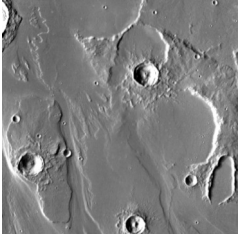
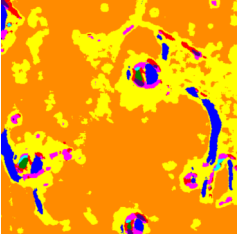
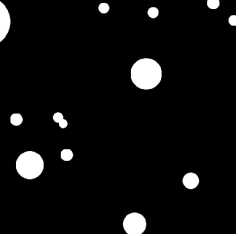
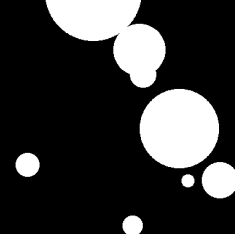
Bez.	Eingabe	Segmentierung durch unüberwachtes tiefes Clustering	Menschlich annotierte Segmentierung	Ground Truth
a)				
b)				
c)				

Tabelle 5.5.1: Eine Auswahl der Segmentierungsergebnisse. Die Farben der jeweiligen Cluster wurden zufällig gewählt. Die „menschlich annotierte Segmentierung“ wurde nach den Regeln aus Unterkapitel 5.4.1 erzeugt, die Ground Truth stammt aus [RH12].

wurden, sondern auch auf weiteren Datensätzen wie [BDS10]. Auf diesem Datensatz wurden bereits die in Tabelle 5.6.1 aufgezeigten  $F_1$ -Scores erreicht.

Eine weitere Ursache für den vergleichsweise niedrigen  $F_1$ -Score ist, dass die direkte Kratererkennung nicht das Ziel des vorgestellten Algorithmus ist. Dieser ist eher auf die Erkennung und Segmentierung von unterschiedlichen Oberflächenmerkmalen spezialisiert. Dies hat, wie bereits erläutert, zur Folge, dass durch die menschliche Analyse eine weitere Fehlerquelle eingeführt wird.








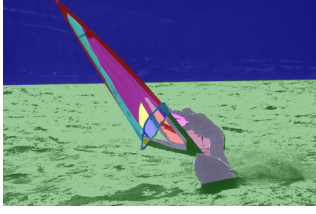

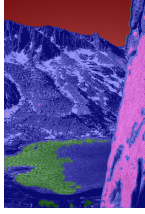
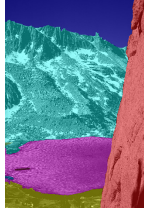




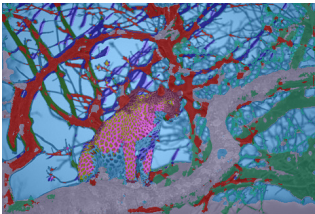

Bez.	Eingabe	Segmentierung	Ground Truth
a)			
b)			
c)			
d)			
e)			

Tabelle 5.5.2: Eine Auswahl der Segmentierungsergebnisse. Die Farben der jeweiligen Cluster wurden zufällig gewählt.

### 5.6.2 Allgemein

Die Ergebnisse der Anwendung des in dieser Arbeit vorgestellten Algorithmus sind in Tabelle 5.6.2 zu finden. Alle weiteren Ergebnisse wurden aus [AMFM10] übernommen.

Algorithmus	F <sub>1</sub> -Score
Urbach '09 [US09] (Unüberwacht)	0,7243
Bandeira '10 [BDS10] (Unüberwacht)	0,8359
Ding '11 [DSM <sup>+</sup> 11] (Unüberwacht)	0,8550
Cohen '16 [CLLD16] (Überwacht)	0,8929
<b>Unüberwachtes tiefes Clustering</b>	0,6374

Tabelle 5.6.1: Vergleich der durchschnittlichen F<sub>1</sub>-Scores unterschiedlicher Algorithmen zur Kratererkennung (vgl. Abschnitt 3.3). Die Daten wurden aus [CLLD16] übernommen.

Nur die Werte, welche über die dort beschriebene „optimal dataset scale“-Methode generiert wurden, wurden übernommen, da der vorgestellte Algorithmus ebenfalls die selben Parameter für alle Eingabebilder nutzt. Die Erweiterung des Algorithmus um eine Methode zur Generierung einer „optimal image scale“ wäre in der Theorie machbar (bspw. über eine jeweilige binäre Suche über alle zu optimierenden Parameter). Aufgrund des dazu benötigten enormen Rechen- und Zeitaufwandes wird diese Metrik allerdings an dieser Stelle nicht beachtet.

Segmentierungsmethode	Probabilistic Rand Index	Variation of Information
Mensch [AMFM11]	0,88	1,17
gPb-pwt-ucm [AMFM10]	0,83	1,69
Mean Shift [CM02]	0,79	1,85
Felz-Hutt [FH04]	0,80	2,21
Canny-owt-ucm [AMFM10]	0,79	2,19
NCuts [CBS05]	0,78	2,23
Quad-Tree	0,73	2,46
<b>Unüberwachtes tiefes Clustering</b>	0,73	2,88

Tabelle 5.6.2: Metriken der Anwendung mehrerer Algorithmen auf den BSDS-500 Datensatz [AMFM11]. Die Werte wurden aus [AMFM10] übernommen und sind anhand des PRIs absteigend geordnet.

Aus dem Vergleich mit weiteren Segmentierungsmethoden wird deutlich, dass sich der hier vorgestellte Algorithmus durchaus zur Segmentierung von alltäglichen Aufnahmen eignet. Die Metriken deuten darauf hin, dass die Qualität der Ergebnisse zwar nicht ganz auf dem Niveau der führenden Alternativen sind, allerdings wurde gezeigt, dass der vorgestellte Algorithmus auf eine Großzahl an unterschiedlichen Eingaben angewandt werden kann.

## 5.7 DISKUSSION

### 5.7.1 Domänenspezifisch

Insgesamt scheint der erarbeitete Algorithmus gute Ergebnisse zur Kratererkennung zu liefern, unter der Voraussetzung, dass die entstandene Segmentierung manuell zu einem Kraterdatensatz weiter verarbeitet wird. Alternativ würde sich der Algorithmus auch zur Erstellung einer Datenbank von Kraterkandidaten eignen, welche anschließend maschinell oder manuell weiter verfeinert werden kann (vgl. Abschnitt 3.3). Dieser Schluss folgt aus der genaueren Betrachtung der Aufnahmen und dazugehörigen Segmentierungen in Tabelle 5.5.1: Dort ist erkenntlich, dass ein Großteil der Krater erkannt werden. Dies schließt auch jene Krater mit ein, deren Durchmesser zu gering waren, um in *Robbins Crater Dataset* aufgelistet zu werden, diese sind an den kleineren verstreuten, kreisförmigen Segmenten zu erkennen.

Zu dem eigentliche Ziel dieser Arbeit, der Segmentierung der Marsoberfläche, wurden allerdings keine adäquaten Ergebnisse erreicht. In den genannten Aufnahmen sind zwar viele Krater deutlich gekennzeichnet, andere Oberflächenstrukturen wurden vom Algorithmus jedoch nicht erfolgreich erkannt. Dies ist insbesondere verwunderlich, da diese in Abschnitt 5.2 und 5.3 vergleichsweise zuverlässig klassifiziert wurden, und alle Parameter so angepasst wurden, dass diese auf eine komplette Segmentierung hinarbeiten statt auf eine Kratererkennung.

### 5.7.2 Allgemein

Obwohl dieser Ansatz ursprünglich nicht zur Segmentierungen von allgemeinen Fotografien erstellt wurde, liefert dieser unerwartet gute Ergebnisse. Dies ist insbesondere auffällig, da nur sehr wenige Parameter angepasst werden mussten (vgl. Unterabschnitt 5.3.6) um diese Resultate zu erreichen.

Eines der größten Probleme bei der Anwendung auf den BSDS-500-Datensatz besteht allerdings daraus, dass die Aufnahmen zu sehr anhand ihrer Textur analysiert

werden: So fällt in Tabelle 5.5.2 auf, dass bspw. auf den Abbildungen d) und e) fast ausschließlich anhand der Textur segmentiert wird. Auf Abbildung d) unterscheidet das Netzwerk zwischen den horizontalen und vertikalen Streifen auf dem Kleidungsstück, während auf Abbildung e) zwischen den größeren und kleineren Punkten des Leoparden differenziert wird. Ein geänderter Skalierungsfaktor für die Filterbank, welche in der Initialisierung genutzt wird, führt auf diesen wenigen Aufnahmen zwar zu einer besseren Segmentierung, der  $F_1$ -Score, welcher über den kompletten Datensatz berechnet wird, wird dadurch allerdings gesenkt.

Dieses Phänomen ließe sich über unterschiedliche Skalierungsgrößen innerhalb der Initialisierung lösen (vgl. Unterabschnitt 5.2.2), dafür wäre allerdings ein Algorithmus von Nöten, welcher geeignete Skalierungen berechnet. Diese Tatsache wird auch in [AMFM10] erwähnt: Dort wird zwischen einer „optimal dataset scale“ und einer „optimal image scale“ unterschieden, letzteres wird durch ein Orakel bestimmt.

### 5.7.3 Weitere Anwendungsgebiete

Durch die vorgestellten Resultate wurde gezeigt, dass tiefes unüberwachtes Clustering sich sehr gut dazu eignet, eine Aufnahme anhand ihrer Textur in Segmente einzuteilen. Obwohl dies auf Fotografien nicht immer zu wünschenswerten Ergebnissen führt, lässt sich dieser Ansatz in weiteren Anwendungsgebieten einsetzen: So besteht eine mögliche Einsatzdomäne aus dem medizinischen Bereich, in welchem auch oft zwischen einzelnen Strukturen in einer Aufnahme unterschieden werden muss.

Insbesondere die Tatsache, dass keine Ground Truth benötigt wird, und der Algorithmus mithilfe der vielen Parameter (vgl. Abschnitt 5.2 und 5.3) angepasst werden kann, eignet er sich in vielen Anwendungsbereichen zur Klassifizierung von Aufnahmen anhand ihrer Textur.

Zur Nutzung im Anwendungsgebiet des autonomen Fahrens wäre tiefes unüberwachtes Clustering zwar auch anwendbar, da es bspw. die Textur einer Straße gut von anderen Objekten wie Straßenmarkierungen unterschieden könnte.

## FAZIT

## 6.1 ZUSAMMENFASSUNG

In dieser Arbeit wurde auf einen Algorithmus zur Bildsegmentierung aus [Kan18] aufgebaut, um diesen so zu optimieren, dass dieser ein domänenspezifisches Problem löst. Dieses Problemstellung besteht daraus, Satellitenaufnahmen der Marsoberfläche anhand ihrer Merkmale in unterschiedliche Segmente zu klassifizieren.

Wie bei vielen aktuellen Segmentierungsproblemen wird zur Lösung ein konvolutionelles neuronales Netzwerk zur Hilfe gezogen. Diese zeigten zwar in vielen Anwendungsbereichen großen Erfolg, ihre Anwendung benötigt aber meistens einen Trainingsdatensatz, dieser ist für die Marsoberfläche jedoch nicht vorhanden. Es existieren zwar einige Datensätze und neuronale Netze zur Analyse der Marsoberfläche [CLLD16], diese konzentrieren sich allerdings fast ausschließlich auf die Kratererkennung. Stattdessen soll das Netzwerk anhand einer weiteren Clusteringmethode lernen, nach welchen Kriterien es das Eingabebild segmentieren soll.

**INITIALISIERUNG** Diese Clusteringmethode war im originalen Algorithmus nach [Kan18] noch der SLIC-Algorithmus [ASS<sup>+</sup>10], dieser eignet sich nicht zur Anwendung in der hier beschriebenen Domäne, da er ausschließlich anhand der Koordinaten und Farbwerte eines Pixels entscheidet, mit welchen weiteren Pixeln dieser in ein Cluster zusammengefügt wird (vgl. Unterabschnitt 4.2). In dieser Arbeit wurde die Nutzung eines alternativen Clusterings zur Initialisierung untersucht, welches sich besser zur Erstellung von texturbasierten Clusterings eignet. Dabei wurde auf das texturbasierte Clustering mithilfe von Gabor-Filtern [JF91] genauer eingegangen, und für dieses eine geeignete Filterbank und die restlichen Parameter ausgewählt.

**NETZWERKARCHITEKTUR** Obwohl die Netzwerkarchitektur aus der ursprünglichen Ausarbeitung [Kan18] bereits gute Ergebnisse erzielt hat, wurde versucht, diese weiter zu optimieren. Aus diesem Grund wurden Pooling-Schichten, Fully-Connected-Layers, alternative Abbruchkriterien und weitere Aktivierungsfunktionen daraufhin überprüft, ob sie diese Ergebnisse noch weiter verbessern können. Schlussendlich wurde neben kleineren Hyperparameter-Änderungen nur das Abbruchkriterium ersetzt. Ein

Problem bestand darin, dass das originale Abbruchkriterium eine feste Anzahl an Segmenten benötigt hat. Da sämtliche Eingabebilder aber jeweils eine unterschiedliche Anzahl an Merkmalen aufweisen, war diese Lösung suboptimal. Für alternative Abbruchkriterien kamen unterschiedlichste Metriken in Frage, zu den besten Resultaten führt allerdings das Unterschreiten eines Grenzwertes für die Ableitung einer Annäherung an die Verlustfunktion.

## 6.2 FAZIT

In dieser Arbeit wurde ein Algorithmus entwickelt, welcher Aufnahmen der Marsoberfläche anhand ihrer Oberflächenmerkmale, welche durch die jeweilige Textur erkannt werden, segmentiert. Dabei gilt es allerdings zu beachten, dass der Algorithmus in seiner derzeitigen Form nicht direkt auf allen Eingabeaufnahmen optimale Ergebnisse liefert: Während auf dem Datensatz der Context Camera des Mars Reconnaissance Orbiters (vgl. Unterabschnitt 2.1.2) viele gute Segmentierungen produziert wurden, verlief die Analyse der Oberfläche auf den Referenzaufnahmen des *Robbins Crater Dataset* [RH12] weniger erfolgreich, dort wurden hauptsächlich Krater erkannt (vgl. Unterabschnitt 5.7.1). Das gewünschte Ergebnis ist in den Unterabschnitten sichtbar, in welchen die entsprechenden Hyperparameter optimiert wurden. So sieht man bspw. in Tabelle 5.3.5, dass auf anderen Aufnahmen erfolgreich zwischen den unterschiedlichen Oberflächenstrukturen unterschieden werden konnte.

Obwohl der Algorithmus ursprünglich zur Anwendung auf Marsaufnahmen gedacht war, lässt dieser sich auch in weiteren Anwendungsgebieten einsetzen: So wurden bspw. unerwartet gute Ergebnisse bei der Anwendung auf den BSDS-500-Datensatz [AMFM11] erzielt, welcher alltägliche Fotografien beinhaltet. Ein weiterer Pluspunkt besteht daraus, dass durch die Tatsache, dass der Algorithmus unüberwacht operiert, er auch bei Segmentierungsproblemen eingesetzt werden kann, bei denen keine Ground Truth existiert. Ein Nachteil gegenüber bereits vorhandener, klassischer Segmentierungsalgorithmen besteht allerdings in der Performance: Durch eine durchschnittliche Rechenzeit von etwa einer Minute pro Aufnahme eignet er sich nicht in Anwendungsgebieten, in denen Echtzeitergebnisse von Nöten sind.

**PROBLEME** Die erste Hürde bei der Entwicklung dieser Methode zur Bildsegmentierung trat, bevor ein neuer Initialisierungsalgorithmus eingeführt wurde. Das Hauptproblem bestand bis an dieser Stelle daraus, wie dem Netzwerk am besten beigebracht werden kann, anhand der Textur zu segmentieren. Einige mögliche Lösungsansätze, wie den Mean-Shift-Algorithmus auf die Eingabedaten anzuwenden

oder die Aktivierungen der jeweiligen Filter der Filterbänke direkt an das neuronale Netzwerk weiterzureichen führten allerdings zu keinen brauchbaren Resultaten. Die schlussendlich vorgestellte, unerwartet einfache Lösung, den k-Means Algorithmus auf die Filterbankresultate anzuwenden wurde erst nach vielen anderen Lösungsansätzen erforscht.

Aber auch mit dieser Initialisierung bestand das Problem weiterhin, dass das neuronale Netzwerk eine Segmentierung erstellt, welche sich zu stark nach den Helligkeitswerten der Eingabe richtet. Ein wichtiger Bestandteil zur Lösung, die Anzahl der in der Initialisierung zu generierenden Cluster, wurde in Unterabschnitt 4.2.4 aufgeführt. Bis dieser Parameter verändert wurde entsprach die Anzahl der Initialisierungscluster wie in der originalen Arbeit [Kan18] gleich der Anzahl der Merkmalsebenen im neuronalen Netz.

Das letzte größere Problem bei der eigentlichen Entwicklung des Algorithmus bestand, wie bereits beschrieben, aus der Suche nach einem geeigneten Abbruchkriterium. Es wurde mit Grenzwerten für einige Parameter, wie die Anzahl der generierten Segmente oder der direkten Verlustfunktion, ist aufgefallen, experimentiert. Diese nähern sich zwar jeweils an einen Wert an, bloß ist dieser von Aufnahme zu Aufnahme unterschiedlich. Des Weiteren verläuft bspw. die Verlustfunktion sehr instabil, wie in Unterabschnitt 5.3.1 aufgezeigt wird.

Da diese zu instabil (und diskret statt stetig) verläuft, war es im Hinblick auf die Programmierung nicht einfach möglich, diese Funktion abzuleiten, um schlussendlich das Überschreiten eines Grenzwertes dieser Ableitung als Abbruchkriterium zu nutzen. Abhilfe schaffte an dieser Stelle eine Approximation der Verlustfunktion durch die Funktion  $f(x) = a\frac{1}{x} + b$ . Mithilfe von dessen Ableitung wurde ein geeignetes Abbruchkriterium geschaffen.

Eine Herausforderung bei der Evaluation des Algorithmus stellte der Mangel an verwandten Arbeiten und Datensätzen dar: Für alltägliche Anwendungsbereiche sind selbstverständlich viele Datensätze verfügbar, wie das genutzt BSDS-500-Datensatz, oder auch *Common Objects in Context* [LMB<sup>+</sup>14]. Für den Mars allerdings existiert kein auch nur teilweise segmentierter Datensatz, und auch jene zur Kratererkennung sind selten und schwer zu finden. Daher wurde in dieser Arbeit eine Methode zum Post-Processing vorgestellt. Mithilfe dieser konnten aus der Segmentierung einzelne Krater extrahiert werden (vgl. Unterabschnitt 5.4.1). Diese konnten anschließend mit Datensätzen verglichen werden, welche nur Informationen über Krater katalogisiert haben, auch wenn so keine optimalen Ergebnisse produziert werden konnten, da der Algorithmus nicht explizit auf diese Aufgabenstellung hin angepasst wurde.

### 6.3 ZUKÜNFTIGE ARBEITEN

Diese Arbeit soll als Grundlage für weitere Forschung dienen. So wurden einige in der Arbeit beschriebene Probleme bzw. deren Lösungsansätze noch nicht vollständig evaluiert. Ein Beispiel dafür liefert die Nutzung von dynamischen Filtergrößen in der Initialisierung (vgl. Unterabschnitt 5.7.2). Eine weitere Verbesserungsmöglichkeit besteht aus der Nutzung bereits erprobter Netzwerkarchitekturen zur Bildsegmentierung, wie bspw. [RFB15].

In Unterabschnitt 5.7.2 wird des Weiteren die Nutzung eines Orakels zur Generierung optimaler Parameter (insbesondere die Filtergröße) für die Initialisierung eines jeden Eingabebildes erläutert, dieser Ansatz wurde sich bereits in [AMFM10] zu Nutze gemacht und lässt sich eventuell auch auf den hier erarbeiteten Algorithmus übertragen. Um dieses Orakel entwickeln zu können, müsste allerdings zuerst bestimmt werden, von welchen Eigenschaften des Eingabebildes die optimale Filtergröße abhängt. Zusätzlich könnte die Initialisierung durch die Nutzung von alternativen Filterbanken oder (analog zu SLIC [ASS<sup>+</sup>10]) die Nutzung des CIELAB statt des RGB-Farbraums verbessert werden.

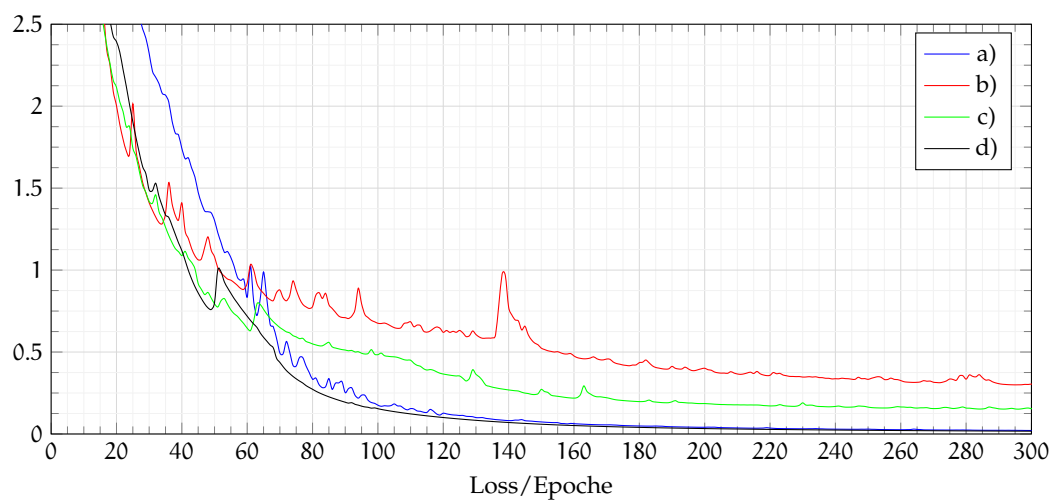
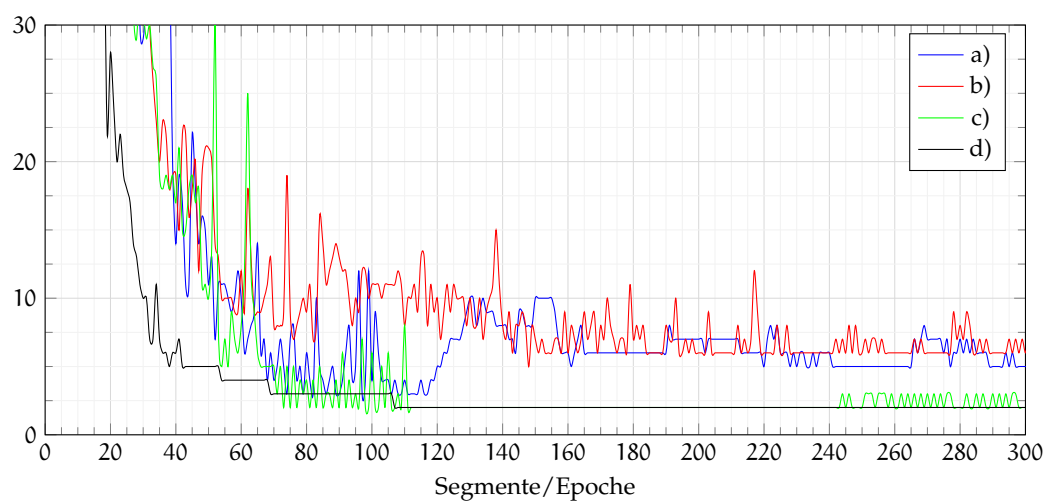
Zukünftige Arbeiten könnten den vorgestellten Ansatz auch so adaptieren, als dass dieser in einer komplett anderen Domäne Anwendung finden kann. In Unterabschnitt 5.7.2 wurde beschrieben, dass sich diese Methode zur Bildsegmentierung hervorragend im medizinischen Bereich einsetzen lassen müsste: Auch dort gilt es häufig, Aufnahmen anhand ihrer Textur zu segmentieren, und auch dort sind oftmals keine Ground Truths vorhanden, mit denen ein überwachtes neuronales Netzwerk diesen Prozess erlernen könnte.

Ein letzter Punkt, der in dieser Arbeit übersprungen wurde, besteht daraus, eine große Aufnahme, wie die in Unterabschnitt 2.1.2 beschriebenen CTX Streifen, auf System mit begrenzten Ressourcen erfolgreich zu segmentieren. Die derzeitige Lösung besteht daraus, die Eingabedatei in mehrere Unteraufnahmen einzuteilen, und diese komplett unabhängig voneinander zu analysieren. Dieser Ansatz lässt allerdings zu wünschen übrig, da so Flächen, welche diese arbiträr gesetzten Grenzen überschreiten, nicht als eine zusammengehörige Region gekennzeichnet werden können. An dieser Stelle könnte ein angepasstes Sliding-Window-Verfahren hilfreich sein.



## GRAPHEN

## A.1 GRAPHEN ZU UNTERABSCHNITT 5.3.1 „ABBRUCHKRITERIUM“



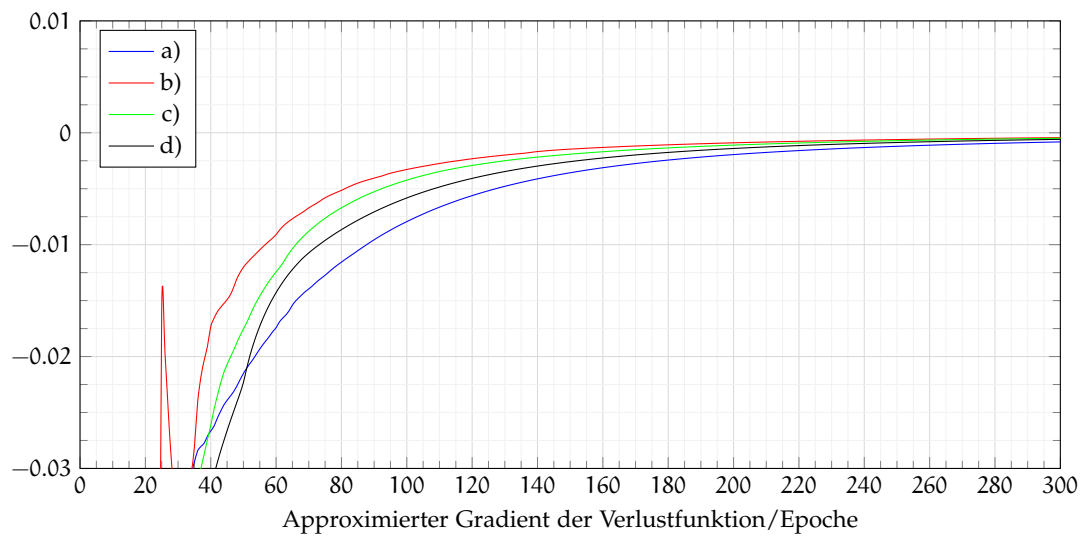
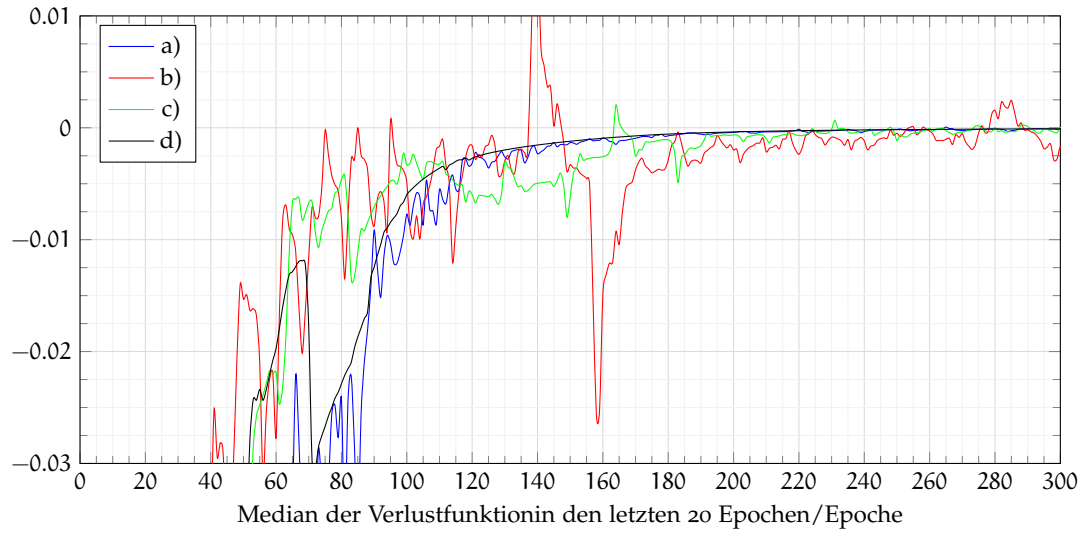


Abbildung A.1.1: Vergleich unterschiedlicher möglicher Abbruchkriterien, jeweils bis 300 Epochen. Die Legenden geben die zugehörige Marsaufnahme aus Abschnitt 5.2 an.

## LITERATURVERZEICHNIS

---

- [AMFM<sub>10</sub>] Pablo Arbelaez, Michael Maire, Charless Fowlkes, and Jitendra Malik. Contour detection and hierarchical image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 33(5):898–916, 2010.
- [AMFM<sub>11</sub>] Pablo Arbelaez, Michael Maire, Charless Fowlkes, and Jitendra Malik. Contour detection and hierarchical image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(5):898–916, May 2011.
- [ASS<sup>+</sup><sub>10</sub>] Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurélien Lucchi, Pascal Fua, and Sabine Süsstrunk. Slic superpixels. *Technical report, EPFL*, page 15, 06 2010.
- [BDS<sub>10</sub>] L. Bandeira, W. Ding, and T. F. Stepinski. Automatic Detection of Sub-km Craters Using Shape and Texture Information. In *Lunar and Planetary Science Conference*, Lunar and Planetary Science Conference, page 1144, Mar 2010.
- [BDS<sub>12</sub>] Lourenço Bandeira, Wei Ding, and Tomasz F. Stepinski. Detection of sub-kilometer craters in high resolution planetary images using shape and texture features. *Advances in Space Research*, 49(1):64 – 74, 2012.
- [BK<sub>15</sub>] Max Berniker and Konrad Kording. Deep networks for motor control functions. *Frontiers in computational neuroscience*, 9:32, 03 2015.
- [Bot<sub>12</sub>] Léon Bottou. *Stochastic Gradient Descent Tricks*, pages 421–436. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.
- [BSP<sub>07</sub>] L. Bandeira, J. Saraiva, and P. Pina. Impact crater recognition on mars based on a probability volume created by template matching. *IEEE Transactions on Geoscience and Remote Sensing*, 45(12):4008–4015, Dec 2007.
- [CBS<sub>05</sub>] Timothee Cour, Florence Benezit, and Jianbo Shi. Spectral segmentation with multiscale graph decomposition. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 2, pages 1124–1131. IEEE, 2005.

- [Chi92] Nancy Chinchor. Muc-4 evaluation metrics. In *Proceedings of the 4th Conference on Message Understanding, MUC4 '92*, page 22–29, USA, 1992. Association for Computational Linguistics.
- [CLLD16] Joseph Paul Cohen, Henry Z Lo, Tingting Lu, and Wei Ding. Crater detection via convolutional neural networks. *arXiv preprint arXiv:1601.00978*, 2016.
- [CM02] Dorin Comaniciu and Peter Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on pattern analysis and machine intelligence*, 24(5):603–619, 2002.
- [DSM<sup>+</sup>11] Wei Ding, Tomasz F. Stepinski, Yang Mu, Lourenco Bandeira, Ricardo Ricardo, Youxi Wu, Zhenyu Lu, Tianyu Cao, and Xindong Wu. Subkilometer crater discovery with boosting and transfer learning. *ACM Trans. Intell. Syst. Technol.*, 2(4), July 2011.
- [FH04] Pedro F Felzenszwalb and Daniel P Huttenlocher. Efficient graph-based image segmentation. *International journal of computer vision*, 59(2):167–181, 2004.
- [Fra18] Chollet François. *Deep learning with Python*. Manning Publications Co., 2018.
- [GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.
- [GP17] Adam Gibson and Josh Patterson. *Deep learning: a practitioners approach*. O'Reilly Media, Inc., 2017.
- [Gre13] Ronald Greeley. *Introduction to Planetary Geomorphology*. Cambridge University Press, 2013.
- [Har17] Larry Hardesty. *Explained: Neural networks*, Apr 2017.
- [HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.
- [IS15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

- [JF91] Anil K. Jain and Farshid Farrokhnia. Unsupervised texture segmentation using gabor filters. *Pattern Recognition*, 24(12):1167 – 1186, 1991.
- [Kan18] Asako Kanezaki. Unsupervised image segmentation by backpropagation. In *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2018.
- [KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. Imagenet classification with deep convolutional neural networks. *Neural Information Processing Systems*, 25, 01 2012.
- [KU19] Çağrı Kaymak and Ayşegül Uçar. *A Brief Survey and an Application of Semantic Image Segmentation for Autonomous Driving*, pages 161–200. Springer International Publishing, Cham, 2019.
- [LJY19] Fei-Fei Li, Justin Johnson, and Serena Yeung. Stanford lecture notes to: Cs231n convolutional neural networks for visual recognition, 2019.
- [LMo1] Thomas Leung and Jitendra Malik. Representing and recognizing the visual appearance of materials using three-dimensional textons. *International Journal of Computer Vision*, 43:29–44, 06 2001.
- [LMB<sup>+</sup>14] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.
- [mat15] Texture segmentation using gabor filters, 2015.
- [MBIC<sup>+</sup>07] Michael C. Malin, James F. Bell III, Bruce A. Cantor, Michael A. Caplinger, Wendy M. Calvin, R. Todd Clancy, Kenneth S. Edgett, Lawrence Edwards, Robert M. Haberle, Philip B. James, Steven W. Lee, Michael A. Ravine, Peter C. Thomas, and Michael J. Wolff. Context camera investigation on board the mars reconnaissance orbiter. *Journal of Geophysical Research: Planets*, 112(E5), 2007.
- [Meio3] Marina Meilă. Comparing clusterings by the variation of information. In Bernhard Schölkopf and Manfred K. Warmuth, editors, *Learning Theory and Kernel Machines*, pages 173–187, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.
- [NASoo] NASA. Mars topography (mola dataset) hires, October 2000.

- [Nie15] Michael A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015.
- [NIGM18] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*, 2018.
- [RFB15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.
- [RH12] Stuart J. Robbins and Brian M. Hynek. A new global database of mars impact craters  $\geq 1$  km: 1. database creation, properties, and parameters. *Journal of Geophysical Research: Planets*, 117(E5), 2012.
- [SAB<sup>+</sup>04] R.S. Saunders, Raymond Arvidson, G.D. Badhwar, William Boynton, P.R. Christensen, Francis Cucinotta, W.C. Feldman, R.G. Gibbs, C. Kloss, M.R. Landano, R.A. Mase, G.W. McSmith, Michael Meyer, I.G. Mitrofanov, G.D. Pace, J.J. Plaut, W.P. Sidney, David Spencer, Thomas Thompson, and Cary Zeitlin. 2001 mars odyssey mission summary. *Space Science Reviews*, 110:1–36, 01 2004.
- [Scho1] Cordelia Schmid. Constructing models for content-based image retrieval. volume 2, pages II–39, 02 2001.
- [Sid18] Asif A. Siddiqi. *Beyond Earth: a chronicle of deep space exploration, 1958-2016*. National Aeronautics and Space Administration, Office of Communications, NASA History Division, 2018.
- [SL10] G. Salamuniccar and S. Loncaric. Method for crater detection from martian digital topography data using gradient value/orientation, morphometry, vote analysis, slip tuning, and calibration. *IEEE Transactions on Geoscience and Remote Sensing*, 48(5):2317–2329, May 2010.
- [SSBD14] Shai Shalev-Shwartz and Shai Ben-David. *Stochastic Gradient Descent*, page 150–166. Cambridge University Press, 2014.
- [STIM18] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? In *Advances in Neural Information Processing Systems*, pages 2483–2493, 2018.

- [UPHo7] Ranjith Unnikrishnan, Caroline Pantofaru, and Martial Hebert. Toward objective evaluation of image segmentation algorithms. *IEEE transactions on pattern analysis and machine intelligence*, 29:929–44, 07 2007.
- [USo9] Erik R Urbach and Tomasz F Stepinski. Automatic detection of sub-km craters in high resolution planetary images. *Planetary and Space Science*, 57(7):880–887, 2009.
- [Vis] Visual Geometry Group - University of Oxford. Texture classification - filters.
- [VLL<sup>+</sup>10] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, 11(Dec):3371–3408, 2010.
- [XGF16] Junyuan Xie, Ross Girshick, and Ali Farhadi. Unsupervised deep embedding for clustering analysis. In *International conference on machine learning*, pages 478–487, 2016.