
# Objekterkennung mit Methoden der künstlichen Intelligenz.

## Detektion von Kiesflächen in Nordrhein-Westfalen auf Basis von Sentinel 2 Satellitendaten mit Convolutional Neural Networks

Bachelorarbeit

vorgelegt von:

**Eike Blomeier**

eblomeier@uos.de

Matrikelnummer: 965672

Studiengang: Geoinformatik (Bachelor of Science)

Betreut durch:

**Prof. Dr. N. de Lange**

und

**Dr. T. Fechner**

Osnabrück, den 09.10.2019

## Kurzfassung

Das Ziel der vorliegenden Bachelorarbeit ist es, das Konzept der Bildsegmentierung durch Convolutional Neural Networks auf Sentinel-2 Satellitenbilder anzuwenden. Dabei wird versucht, mittels Convolutional Neural Networks, aus den Satellitenbildern mit Wasser gefüllte Kiesabbauflächen in Nordrhein-Westfalen zu erkennen und deren Form und Größe zu extrahieren. Durch das Trainieren mehrerer Convolutional Neural Networks mit unterschiedlichen Bandkombination der Kanäle 2, 3, 4 und 8 des Sentinel-2 Systems wird zunächst festgestellt welche Bänder und Kanalkombinationen sich besonders gut eignen um Kiesflächen zu erkennen. Anschließend wird das Modell auf zwei Szenen angewendet. Dabei lässt sich feststellen, dass das CNN in der Lage ist, Kiesflächen zu detektieren, gleichzeitig aber einige Wasserflächen, nach dem Modell, eine hohe Wahrscheinlichkeit besitzen aktuelle Kiesabbauflächen zu sein. Zudem kann eine zuverlässige Extraktion der Form und Größe der Kiesabbauflächen nicht gewährleistet werden. Insgesamt ist beobachtet worden, dass das Konzept funktioniert, aber noch an einigen Stellen Verbesserungspotenzial aufweist.



## Abstract

In this Bachelor thesis, the convolutional neural networks concept of image segmentation has been used on Sentinel-2 satellite images. The aim was to use Convolutional Neural Networks to detect the shape and size of flooded gravel mines from satellite images in North Rhine-Westphalia. Several Networks has been trained, each using different channel combinations of the Sentinel-2 channels 2, 3, 4 and 8. Convolutional Neural Networks. The best model has been used to classify two different test scenes. It's been determined that Convolutional Neural Networks are capable to detect these gravel mines, but the trained models still classify other waters as gravel mines. Also, it couldn't be guaranteed to extract the shape and size of the gravel mines on a reliable base. Finally, the concept works, but offers lot’s room for improvement.

# Inhalt

# 1 Einleitung

## 1.1 Einführung

Die jährliche Wachstumsrate aller im Internet gespeicherten Daten beträgt 40% - 50% (IEEE 802.3 Ethernet Working Group, 2012). Dabei werden immer mehr Daten mit Sensoren, die im IoT (Internet of Things) miteinander vernetzt sind, gemessen und eingespeist (Andelfinger & Hänisch, 2015). Allein die Sentinel-2 Mission der ESA erzeugt innerhalb von fünf Tagen multispektrale Aufnahmen der gesamten Erdoberfläche (ESA EO Missions: Sentinel-2). Eine schnelle und genaue Analyse dieser Datenmenge durch den Menschen ist unmöglich. Machine Learning ermöglicht es, in solchen Daten Muster zu erkennen und daraus Modelle abzuleiten, welche wiederum auf neue Daten angewandt werden können um relevante Informationen zu extrahieren.

## 1.2 Ziel der Bachelorarbeit

Das Ziel dieser Arbeit ist es ein Convolutional Neural Network (CNN) zu programmieren, welches aus gegebenen Trainingsgebieten und den dazugehörigen Klassifikationsmasken, von mit Wasser gefüllten Kiesabbauflächen in NRW, ein Modell erzeugt, dass es erlaubt solche mit Wasser gefüllten Kiesabbauflächen, auf neuen, dem Netzwerk zuvor nicht bekannten Sentinel-2 Szenen, zu klassifizieren.

## 1.3 Aufbau und Gliederung der Bachelorarbeit

Zunächst wird in Kapitel 2 die benötigte Theorie erläutert. Dazu wird in 2.1 auf eine mögliche Definition von künstlicher Intelligenz eingegangen und wie das Gebiet der künstlichen Intelligenz in der Informatik zu verorten ist. Anschließend wird in 2.2 beschrieben, welches Problem von Computern durch Machine Learning gelöst werden soll, was genau ein von Machine Learning erzeugtes Modell enthält und wie dieses Modell durch die im Machine-Learning-Algorithmus verankerten Neuronen erzeugt wird. Außerdem wird der Unterschied zwischen überwachten und unüberwachten Machine Learning erklärt und was das Ziel des im weiteren Verlauf verwendeten überwachten Klassifikationsansatz ist.

Kapitel 2.2.1 stellt die Besonderheiten von Deep Learning heraus. Dabei wird auf den Aufbau von Deep-Learning-Networks (DNNs) eingegangen und dargestellt, wie die einzelnen Layer innerhalb eines DNNs miteinander verbunden sind. Anschließend wird der Begriff

Backpropagation erläutert und warum dieses Konzept wichtig für das Training neuronaler Netze ist. Abschnitt 2.2.2 erläutert das Prinzip der Convolutional Neural Networks (CNNs) und warum diese besonders in der Bildklassifizierung gut abschneiden. Dazu wird auf die wesentlichen Bestandteile, Convolution, Sparse Interaction und Pooling, eines CNNs eingegangen und deren Funktionsweise erklärt.

Kapitel 3 beschäftigt sich mit zwei möglichen Alternativen zur Bildklassifizierung neben CNNs. Dazu wird in 3.1 das Prinzip des Clusterings beschrieben. 3.2 erläutert die Funktionsweise von Support Vector Machines (SVMs). In 3.3 werden knapp die Vorteile eines CNNs gegenüber Clustering und SVM angesprochen und erklärt warum sich in dieser Arbeit für die Klassifizierung mittels eines CNNs entschieden wurde.

Im Anwendungskapitel 4 wird zuerst das später verwendete U-Net beschrieben. Das U-Net ist ein CNN, welches seinen Namen durch seine Struktur und die daraus resultierende U-Form erhalten hat (Abbildung 12). Dazu werden zunächst Beispiele genannt, in denen sich diese Netzwerkarchitektur bereits bewährt hat. Im Anschluss daran wird auf die Vorteile eingegangen für das in 1.2 beschriebene Ziel, ein Neuronales Netz von Anfang an zu trainieren und nicht ein bereits vortrainiertes Netzwerk zu benutzen. Des Weiteren werden auf Gründe eingegangen das U-Net mit dem Keras-Framework zu bauen. 4.1 geht auf die wichtigsten verwendeten Bibliotheken und Software für die Erzeugung geeigneter Trainingsdaten ein. 4.2 beschreibt die Datenvorverarbeitung um die geeignete Trainingsgebiete aus den verwendeten Sentinel-2 Satellitenbildern zu extrahieren und für das CNN vorzubereiten. Dabei wird zunächst beschrieben wie und auf welcher Grundlage die AOIs, also die Kiesabbauflächen, erzeugt wurden. In 4.2.1 geht es um den Download der Szenen und wie diese für die in 4.2.2 erklärte Datenprozessierung aufbereitet wurden. Die Erzeugung der für das überwachte Training wichtigen Binärmasken und das Ausschneiden der Trainingsgebiete aus der Gesamtszene wird in 4.2.2 verdeutlicht. Kapitel 4.3 erläutert den Aufbau des programmierten U-Nets. Dazu werden die Konfigurationen der einzelnen Parameter des U-Nets, der einzelnen Layer im U-Net, der verwendeten Callbacks, des Modells sowie die des Trainings des Modells genannt und erklärt. 4.3.1 beschreibt den allgemeinen Ablauf des Trainings und die gewählten Hyperparameter der Trainingsdaten. In 4.3.2 - 4.3.11 wird das U-Net mit unterschiedlichen Trainingsdaten trainiert und die jeweiligen Trainingsergebnisse genannt.

In Kapitel 5 wird das beste Modell aus 4.3.2 - 4.3.11 anhand zweier Beispielklassifikationen bewertet. Dabei wird darauf eingegangen wie viele Flächen fälschlicherweise als

Kiesabbaufläche klassifiziert worden sind und wie genau die Kiesflächen bezüglich ihrer Form und Größe extrahiert werden.
Im Fazit werden die wichtigsten Erkenntnisse dieser Arbeit wiedergegeben. Es werden die in Kapitel 5 erkannten Probleme aufgegriffen und Möglichkeiten vorgestellt, wie diese Probleme in Zukunft vermieden werden könnten, um genauere Resultate zu erzielen.

# 2 Theorie

## 2.1 Künstliche Intelligenz

Künstliche Intelligenz (KI, engl. Artificial Intelligence) ist ein Teilbereich der Informatik. KI untersucht durch algorithmische Modelle die kognitive Intelligenz des Menschen (Russell & Norvig, 2009, S. 5) und versucht dabei *„normalerweise von Menschen erledigte Aufgaben automatisiert zu lösen“* (Euen, 2015, S. 29). Dabei nehmen die Akteure (Agenten oder Agents) ihre Umgebung über (verschiedene) Sensor(en) wahr und reagieren, basierend auf ihrem Wissen, auf den Input über Aktuator(en). Dieser Prozess ist in Abbildung 1 schematisch dargestellt. Die hier verwendete Agentenmetapher ist sehr allgemein definiert. Nicht jede KI verfügt über Sensoren und Aktuatoren, welche die für den Menschen wahrnehmbare Umgebung wahrnehmen und verändern. Allerdings ist diese Metapher generisch und lässt sich gut für die Erläuterung von KI verwenden.

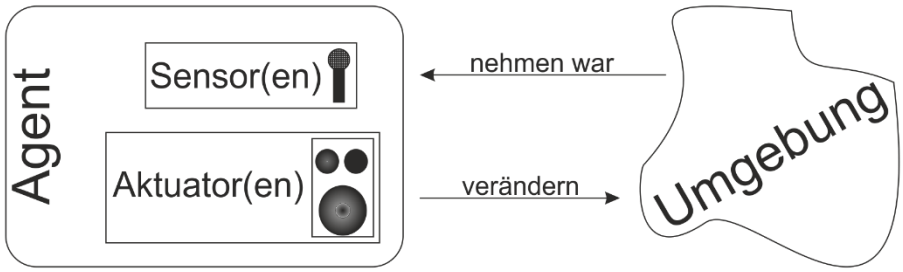


*Abbildung 1 - KI-Umgebungs-Interaktion (Hertzberg, 2018)*

Das KI ein höchst aktuelles Thema ist, lässt sich unter anderem darin erkennen, dass u.a. im aktuellen deutschen Bundeskoalitionsvertrag von 2018 die Errichtung eines „Zentrums für Künstliche Intelligenz“ festgehalten worden ist (Z. 1488 - 1498). Allerdings ist die Idee der künstlichen Intelligenz nicht besonders neu. Bereits in den 1830er bis 1840er Jahren wurde die Analytical Engine von Charles Babbage entworfen, welche ihrer Zeit weit voraus war. Diese war darauf ausgelegt Berechnungen in der mathematischen Analyse durch mechanische

Abläufe zu automatisieren. Im Jahr 1956 wurde dann von J. McCarthy, M. Minsky, N.Rochester, C. Shannon ein Finanzierungsantrag zur KI-Forschung auf der Dartmouth Conference eingereicht.

## 2.2 Machine Learning

Im Gegensatz zum Menschen hat ein Computer kein Problem mit dem Speichern von Informationen, sondern beim Transfer von Daten oder Erfahrungen auf neue bzw. andere Aufgaben oder Daten. An dieser Stelle setzt Machine Learning (ML) an, welches ein Teilgebiet der KI ist. Dabei versucht Machine Learning, durch das Betrachten von Trainingsdaten und im Gegensatz zur klassischen Programmierung, Regeln selbst zu erlernen (Nguyen & Zeigermann, 2018, S. 14). *„Learning denotes changes in the system that are adaptive in the sense that they enable the system to do the task or tasks drawn from the same population more efficiently and more effectively the next time"* (Simon, 2011). Diese Regeln werden in einem so genannten Modell gespeichert, um sie auf neue Daten anwenden zu können. In Abbildung 2 werden nochmal die Unterschiede der klassischen Programmierung und Machine Learning visualisiert. Machine Learning lässt sich wiederum in zwei Hauptgebiete unterteilen: Überwachtes und unüberwachtes Lernen (Sarle, 2002)). Beim überwachten Lernen, dieses wird im weiteren Verlauf der Arbeit behandelt und angewandt, sind Ein- und Ausgabedaten bereits vorher bekannt. Der Machine-Learning-Algorithmus lernt lediglich ein Modell an Überführungsregeln, um von der Ein- auf die Ausgabe zu kommen (Raschka & Mirjalili, 2018, S. 26). Unüberwachtes Lernen lernt nicht anhand von vorher gelabelten Daten. Beim unüberwachten Lernen wird versucht mittels Machine Learning Strukturen in den Daten zu erkennen und Informationen zu extrahieren (Raschka & Mirjalili, 2018, S. 31). Anhand dieser automatisch gelernten Strukturen und Informationen werden anschließend die Daten, basierend auf einem Ähnlichkeitsindex, zu Klassen zusammengefasst. Der Unterschied der Vorgehensweise von überwachten und unüberwachten Verfahren wird nochmal in Abbildung 3 verdeutlicht. Beide Verfahren gehören zu Klassifikationsverfahren, da Inputdaten einem bestimmten Output zugeordnet werden sollen.

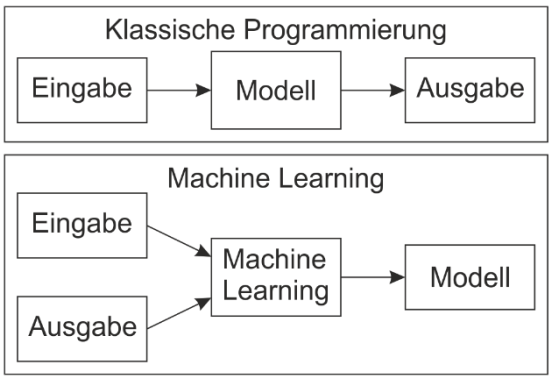


*Abbildung 2 - Paradigmen der klassischen Programmierung und des Machine Learning (Butzmann, 2019)*

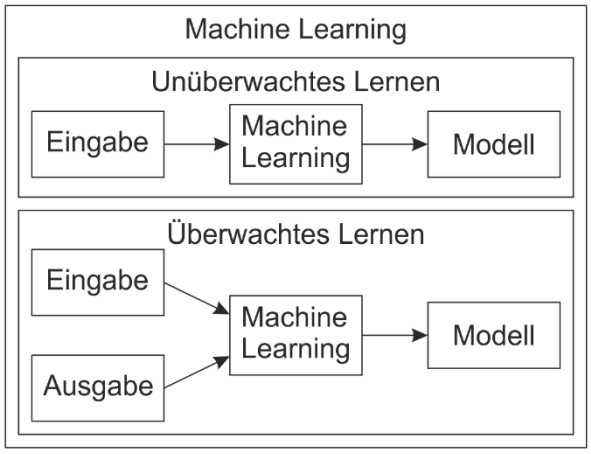


*Abbildung 3 - Unüberwachtes und überwachtes Lernenv (Butzmann, 2019)*

Für die Erzeugung eines Modells durch überwachtes Machine-Learning sind innerhalb des Machine-Learning-Algorithmus sogenannte Neuronen zuständig. Dabei hat jedes Neuron mindestens einen und bis zu n Inputvektoren. Diese Inputvektoren werden innerhalb des Neurons mathematisch verarbeitet und als Outputvektor wieder ausgegeben. Dabei ist die Dimensionalität des Outputvektors variable und reicht von zwei möglichen Outputs bei einer Binärklassifikation bis zu n Outputvektoren. Abhängig von den vom Anwendungsfall geforderten Funktionalitäten. Diese Verarbeitung wird Aktivierungsfunktion genannt. Innerhalb der Aktivierungsfunktion besitzt jeder Inputvektor eine eigene Gewichtung. Dies ist schematisch in Abbildung 4 abgebildet. Die Gewichte der einzelnen Neuronen bilden wiederum am Ende das vom Machine-Learning-Algorithmus erlernte Modell.

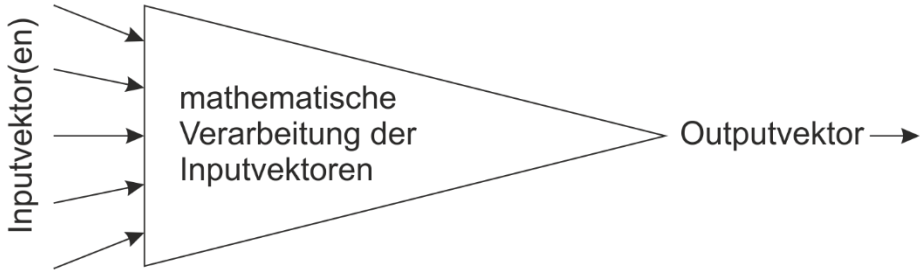


*Abbildung 4 - Funktion eines Neurons (stark vereinfacht)*

Das Ziel des überwachten Lernens ist es, ein Modell zu entwickeln, bei dem die Modellausgabe möglichst gut zu den Input-Daten passt. Dafür muss zunächst durch Training eine Gewichtsverteilung gefunden werden, welche die geforderte Ausgabe optimal wiederspiegelt. Dieser Prozess wird Backpropagation genannt und in Deep Learning genauer erläutert.
So richtig etabliert hat sich das Konzept des Machine Learnings allerdings erst in den 1990er Jahren. Zuvor mangelte es insbesondere an Leistungsfähiger Hardware und Massendaten zum Trainieren. Dann ist es allerdings sehr schnell zum erfolgreichsten Teilgebiet der KI aufgestiegen (Chollet, 2018, S. 24).

### 2.2.1 Deep Learning

Deep Learning ist ein Teilbereich des Machine Learnings. Deep-Learning-Networks (DNNs) sind üblicherweise aus Inputlayer, mehreren Hiddenlayern und Outputlayer bestehende neuronale Netze (Abbildung 5).

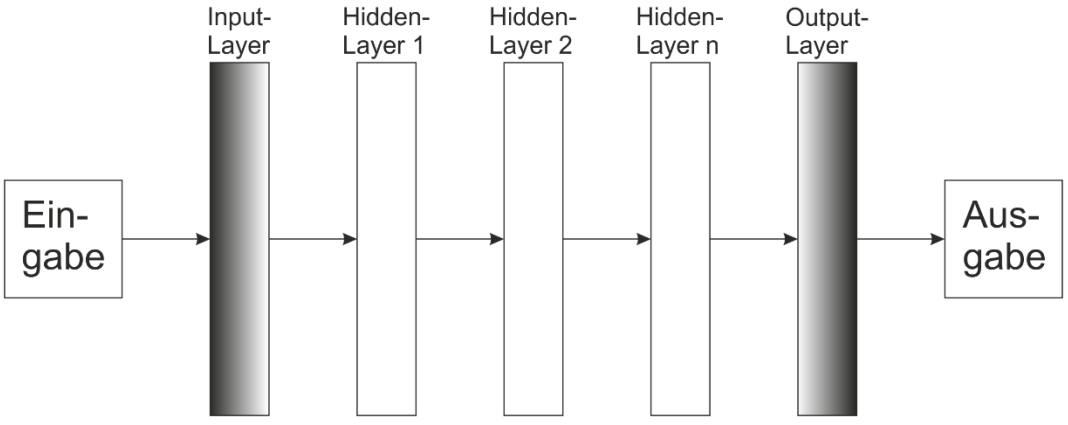


*Abbildung 5 -Schematischer Aufbau eines Deep Learning Networks*

Die einzelnen Layer enthalten Neuronen, welche mit den Neuronen in der nächsten Layerschicht verbunden sind (Abbildung 6). Die einzelnen Neuronen in den Layern verarbeiten

dabei ihre(n) Inputvektor(en) in einen Outputvektor. Der Outputvektor fungiert wiederum als Inputvektor für Neuronen im darauffolgenden Layer.

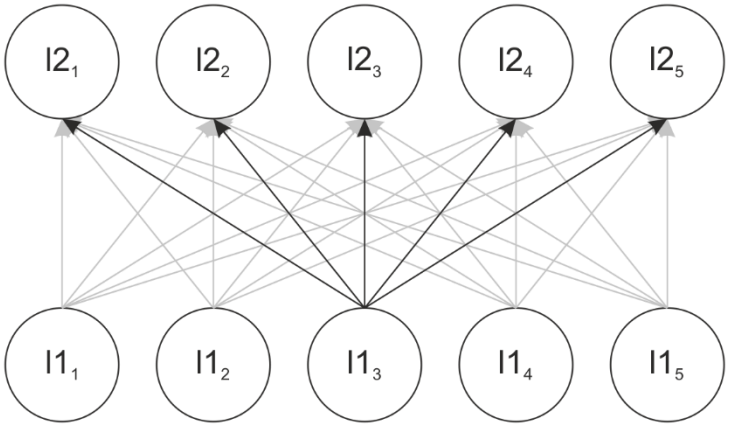


*Abbildung 6 - Zusammenspiel zweier Fully-Connected-Layer*

Bei der Initialisierung eines DNNs werden zunächst die Gewichtungen der einzelnen Aktivierungsfunktionen der Neuronen zufällig initialisiert. Da solche Zufallsgewichtung in der Regel alles andere als ideal sind, wurde bereits in den 1960er Jahren ein Verfahren Backpropagation genannt, entwickelt, um die Gewichtungen während des Lernprozesses immer weiter zu optimieren. Bei Backpropagation wird durch Rückkopplung die Verlustfunktion dahingehend optimiert, dass diese einen möglichst geringen Verlustscore aufweist. Der Verlustscore errechnet sich aus den Werten der Vorhersage Y‘ und den Werten des Zielwertes Y. Um diesen Verlustscore zu minimieren wird mittels einer Optimierungsfunktion eine neue Gewichtung der einzelnen Gewichte innerhalb der Neuronen errechnet (Rumelhart, Hinton, & Williams, 1986). Diesen Prozess nennt man Backpropagation. Abbildung 7 visualisiert diesen Ablauf.

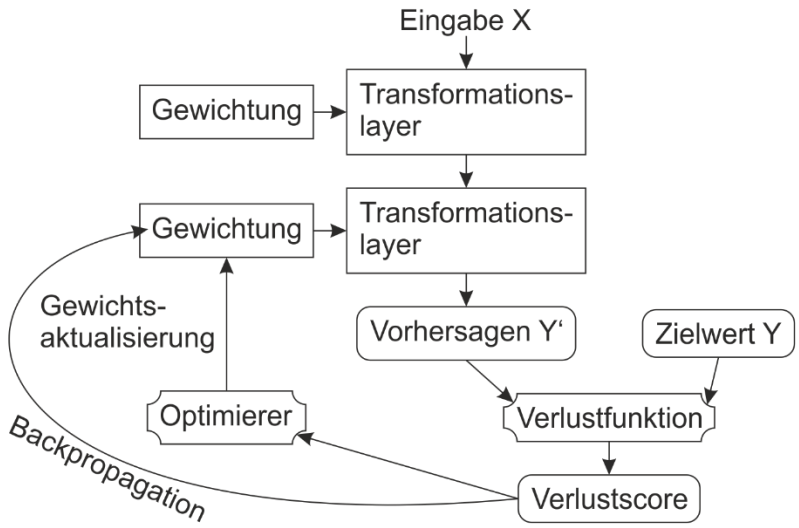


*Abbildung 7 – Backpropagation (Chollet, 2018, S. 31)*

Innerhalb des Deep Learnings sind Convolutional Neural Networks ein bekannter Teilbereich. Das folgende Kapitel 2.2.2 sich mit CNNs und erläutert dieses Thema.

### 2.2.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) gehören zum Bereich der DNNs. Mit Convolution ist an dieser Stelle die mathematische Faltung zweier Funktionen gemeint. Für eine genauere Erläuterung siehe (Godfellow, Bengio, & Courville, 2016, S. 327ff).

Durch ihre Funktionsweise eignen CNNs sich besonders gut zur Merkmalsextraktion aus räumlich korrelierten Daten (z.B. aus den einzelnen Pixelwerten von Bildern). Um Merkmale aus den Daten zu extrahieren wandert eine kleine Maske (Kernel) über die Inputvektoren (Abbildung 8). Ein Kernel ist ein Filter, welcher die Inputvektoren bündelt und dabei Eigenschaften der Inputvektoren herausstellt. Das Bündeln der Inputvekotren nennt sich Aktivierungsfunktion. Die Aktivierungsfunktion erzeugt einen Outputvektor, der an den nächsten Layer übergeben wird.

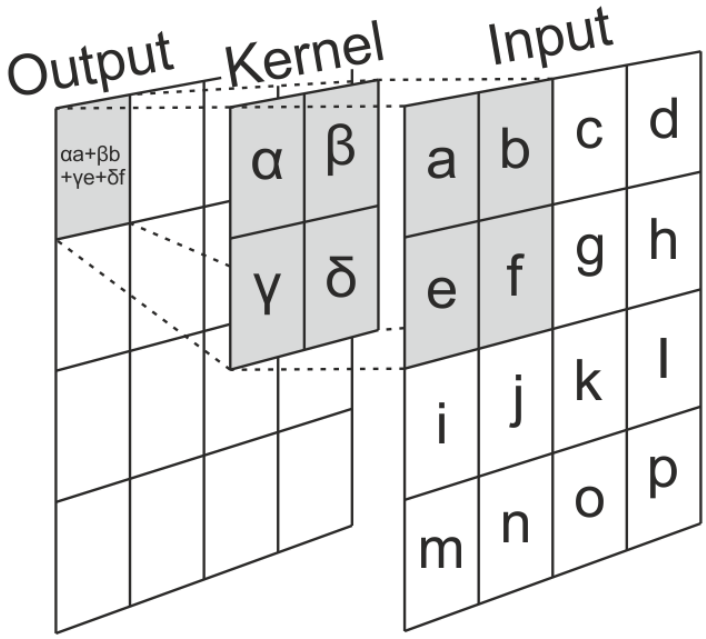


*Abbildung 8 - Funktionsweise Convolution*

Ein großes Problem für CNN ist allerdings, dass durch ihr primäres Anwendungsgebiet in der Bildklassifikation bereits der Inputlayer über einen sehr großen Merkmalsraum verfügt. Beispielsweise muss ein CNN, welches 128x128 Pixel große Bilder klassifizieren bereits einen Inputlayer mit 16384 Eingängen aufweisen. Da in jedem Layer jedoch nicht nur ein Kernel verwendet wird, sondern in der Regel mehrere, werden logischerweise auch mehrere Outputs erzeugt. Nämlich einen pro Kernel. Bei der für CNNs typischen tiefen Konstruktionsweise mit vielen aufeinanderfolgenden Hiddenlayern wird dieser Merkmalsraum sehr schnell sehr groß.
Für ein CNN, welches z.B. Merkmale aus einem 128x128 Pixel großen Input extrahieren soll und dabei aus einem Input- Hidden- und Outputlayer besteht, wobei zwischen Input- und Hiddenlayer, sowie Hidden- und Outputlayer jeweils ein Kernel der Größe 3x2x2 arbeitet um Merkmale zu extrahieren. Dann kommt beim Outputlayer ein 9x128x128 großer Vektorraum an. Dies entspricht 147456 Parametern.
Es muss also eine sehr große Anzahl an Parametern eingestellt werden, obwohl bereits eine kleinere Anzahl an Neuronen ausreichen würde, um wichtige Merkmale zu erkennen. Dies würde ein Training eines solch konstruierten Netzes verlangsamen. Damit dies vermieden wird, werden in CNNs verschiedene, unten genauer erläuterte Techniken angewandt um die Trainingseffizienz zu verbessern.

Sparse Interaction reduziert den Inputraum, indem nicht jedes Neuron des aktuellen Layer mit jedem Neuron des folgenden Layer verknüpft ist (Abbildung 9). Sparse Interaction führt dazu, dass zum einen der benötigte Speicher reduziert wird (weniger Gewichtungen in den Neuronen). Dies führt wiederum dazu, dass weniger Berechnungen durchgeführt werden müssen und somit die Rechenzeit verkürzt wird (Godfellow, Bengio, & Courville, 2016, S. 330f).

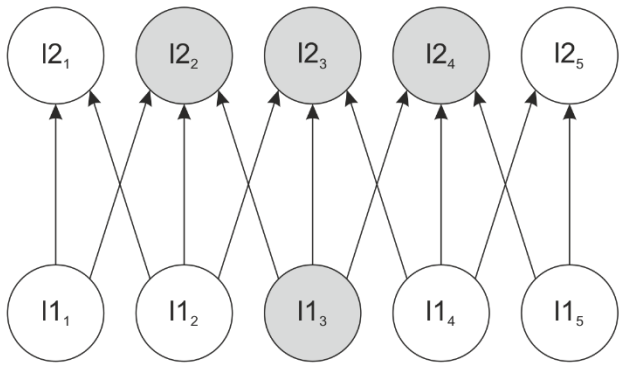


*Abbildung 9 - Sparse Interaction (Godfellow, Bengio, & Courville, 2016, S. 331)*

Parameter Sharing heißt, dass für jeder Convolution-Operation innerhalb eines Layer der gleiche Gewichtsvektor benutzt wird. So besitzt jedes Layer nur einen Gewichtungskernel. Dies hat keinen Effekt auf die Laufzeit, spart aber Speicherplatz ein (Godfellow, Bengio, & Courville, 2016, S. 331ff).

Dies führt direkt zu einer weiteren wichtigen Fähigkeit der einzelnen CNN-Layer, die sogenannten Equivariant Operations. Das heißt, dass wenn sich der Input für eine Funktion (Kerneloperation) ändert, ändert sich der Output dieser Kerneloperation auf dieselbe Art. Bei der Klassifizierung von Bilddaten würde das beispielsweise bedeuten, dass ein CNN nicht nur in der Lage wäre eine Aussage darüber zu treffen, ob ein Objekt in dem Bild vorhanden ist, sondern auch bestimmen kann wo das Objekt sich befindet (Godfellow, Bengio, & Courville, 2016, S. 334f).

Eine weitere wichtige Funktion von CNNs ist das Pooling. Beim Pooling wird der Output verkleinert, in dem nur bestimmte Outputs in den nächsten Layer überführt werden (Abbildung 10). Dies bewirkt eine gewisse Outputtoleranz gegenüber dem Input. Wird der Input also um einen geringen Wert verändert, bleiben die meisten Outputvektoren davon unberührt (Godfellow, Bengio, & Courville, 2016, S. 336). Somit kann Pooling gut dafür genutzt werden um zum einen den Merkmalsraum zu reduzieren und zum anderen wichtige Eigenschaften trotzdem zu behalten.

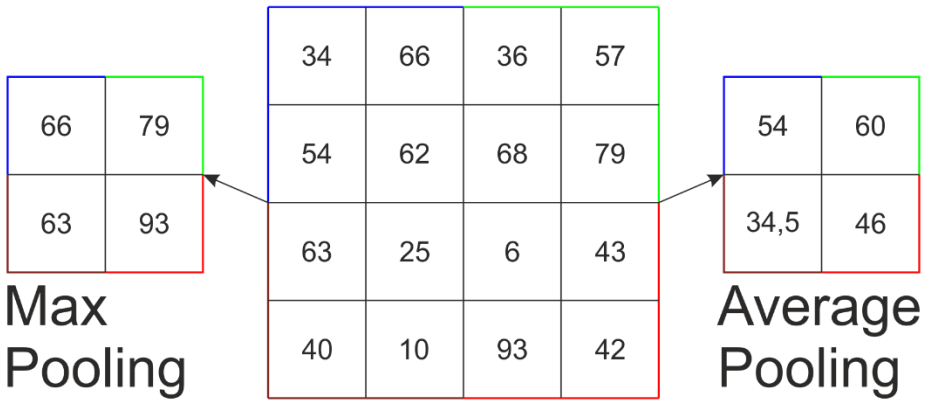


*Abbildung 10 – 2x2 Pooling, Max Pooling bewirkt, dass nur der größte Wert weitergegeben wird. Average Pooling bewirkt berechnet hingegen den Durschnitt der Werte und gibt diesen weiter*

Zusammenfassend ist festzustellen, dass ein CNN aus mehreren Convolutional Layern besteht. Die Convolutional Layer bestehen wiederum aus drei Abschnitten (Abbildung 11):

1. Convolution: Berechnung der Aktivierung α durch Faltung
2. Detection: Anwendung einer nichtlinearen Aktivierungsfunktion
3. Pooling: Ersetzung des Outputs vom Detection-Abschnitt durch Statistiken

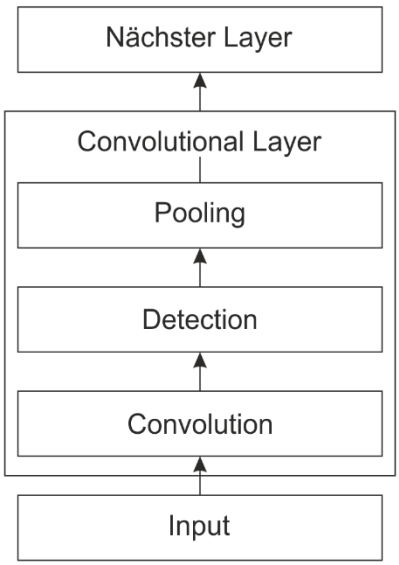


*Abbildung 11 - CNN Komponenten (Godfellow, Bengio, & Courville, 2016, S. 336)*

# 3 Beispiele alternativer Verfahren zur Detektion von Kiesflächen

## 3.1 Clustering

Clustering ist ein Verfahren aus dem Bereich des unüberwachten Lernens mit dem Ziel, die Eingangsdaten anhand ihrer Eigenschaften zu kategorisieren (Bacher, Pöge, & Wenzig, 2011, S. 15). Dabei sind die Daten innerhalb eines Clusters homogener zueinander und heterogener gegenüber Daten anderer Cluster (Hudson, 1982, S. 6). Durch dieses Homogenitätsgebot kann es passieren, dass Clusterklassen entstehen, die für den Menschen, aufgrund einer anderen Wahrnehmung, nicht logisch erscheinen.

## 3.2 Support Vector Machines

Support Vector Machines (SVMs) gehören überwachten Konzepten des Machine Learnings. SVMs projizieren Daten über Stützvektoren in einen Vektorraum. Dieser Vektorraum kann nach unterschiedlichen Parametern definiert werden. Beispielsweise können Pixelwerte eines RGB-Bildes in einen dreidimensionalen Vektorraum projiziert werden, wobei die RGB-Kanalwerte eines Pixels dabei als Stützvektoren genommen werden können. Dabei versucht die SVM ihre Eingangsdaten durch eine Hyperebene zu separieren (Awad & Khanna, 2015, S. 39f). Um Daten innerhalb eines Vektorraums besser separieren zu können, wird u.a. von SVMs der Kernel-Trick angewendet. Dabei werden die Stützvektoren in einen höheren Vektorraum projiziert, welcher eine bessere Separierbarkeit der Daten erlaubt (Frochte, 2018, S. 293ff). Neben dem Kernel-Trick zur besseren Separierbarkeit der Daten müssen zunächst geeignete Features zur Klassifizierung extrahiert werden, um später gute Klassifizierungsergebnisse zu erhalten. Das Problem der Featureselektion wird von (Weston, et al., 2001) genauer erläutert. In (Frochte, 2018) sind SVMs und ihre Funktionsweise theoretisch erklärt und anhand von praktischen Beispielen verständlicher gemacht.

## 3.3 Vorteile von CNN gegenüber Clustering und SVMs zur Detektion von Kiesflächen

Ein einfaches Clustering von Pixeln nach dem in 3.1 erläuterten Verfahren fällt in dieser Anwendungsfrage raus, da es sich nicht um eine unüberwachte Klassifikation handelt.

Im Gegensatz zu SVMs haben CNNs keine Probleme damit, dass ein Featurespace festgelegt werden muss, in den die Daten projiziert werden. Durch ihre Funktionsweise werden wichtige Features innerhalb der Convolutional-Layer extrahiert und Zusammenhänge erkannt. Somit wird sich durch den Einsatz von CNN zur Erkennung von Kiesflächen erhofft, dass das CNN in der Lage ist, Features, welche ein Kiesabbaugebiet kennzeichnen, automatisch aus den Satellitenbildern zu extrahieren und diese Eigenschaften auf neue, unbekannt Szenen anzuwenden, um Kiesflächen zu detektieren.

Bereits 2014 wurde dieser Vorteil der automatischen Extraktion von Eigenschachten deutlich, als das von Simonyan und Zisserman entwickelte VGG16-CNN eine Klassifikationsgenauigkeit von bis zu 92,7% erreichte (Simonyan & Zisserman, 2014, S. 12), während eine von Lin et al. entwickelte SVM auf eine Klassifikationsgenauigkeit von 52,9% kam (Lin, et al., 2011, S. 1695). Beide Ansätze hatten zum Ziel die 1000 Klassen der ImageNet-Datensammlung zu Klassifizieren. Dabei wurden als Trainingsgrundlage die in der ImageNet-Datensammlung enthaltenen Bilder verwendet. Entsprechend dieser Beurteilung hat sich der Autor dazu entschieden das modernere Verfahren der Bildsegmentierung durch CNNs zu verwenden.

# 4 Anwendung

Für den Prozess der Kiesflächendetektion mittels CNN wurde das von Ronneberger, Fischer und Brox entwickelt U-Net gewählt und mit Hilfe des Keras-Frameworks nachgebaut (Abbildung 12)[1].

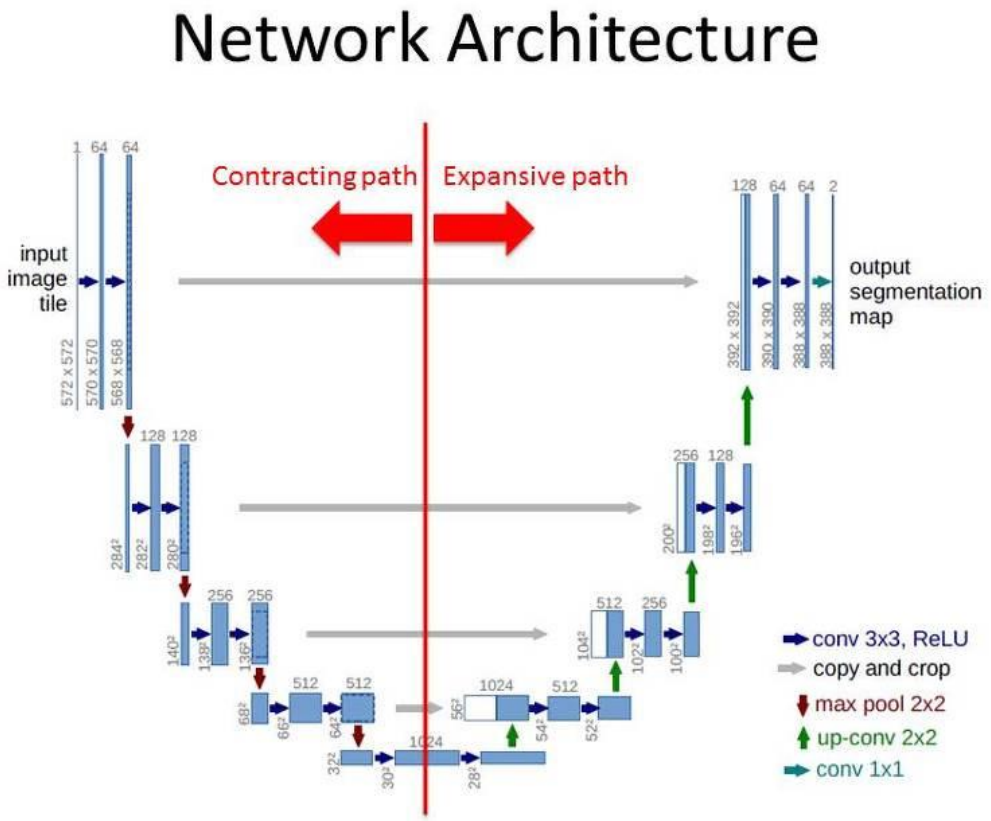


*Abbildung 12 - U-Net (Ronneberger, Fischer, & Brox, 2015, S. 235)*

Das U-Net ist eine Netzwerkstruktur, welches eine Segmentation-Map als Output liefert. Es wurde ursprünglich zur Segmentierung von biologischen Zellen, sowie sehr kleine Trainingsdatensätze entwickelt und erprobt (vgl. 2015, U-Net: Convolutional Networks for Biomedical Segmentation).

Neben seinem ursprünglichen Anwendungsgebiet in der Biomedizin, wo das U-Net von Ronneberger et al. 2015 die ISBI cell tracking challenge gewinnen konnte, ist die U-Net-Struktur auch bereits in weiteren Anwendungsgebieten erfolgreich verwendet worden. So hat diese Netzwerkstruktur die TGS Salt Identification Challenge 2018 gewinnen können und

[1] http://deeplearning.net/tutorial/_images/unet.jpg, [23.09.2019]

wurde im Bereich der Geoinformatik erfolgreich zur Klassifizierung von Tulpenfeldern eingesetzt (Marthi, 2018).

In allen genannten Anwendungsgebieten ist der große Vorteil, welcher der menschliche Betrachter durch das U-Net erlangt, der, dass für ihn nur schwer bis überhaupt nicht erkennbare Unterschiede sichtbar gemacht werden.

Alternativ zum Entwickeln und Trainieren eines eigenen Netzwerkes, hätte auch auf ein vortrainiertes Netzwerk, z.B. VGG16 (Simonyan & Zisserman, 2014) zurückgegriffen werden können. Das Verwenden eines bereits trainierten Modells wäre mit Vor- und Nachteilen einhergegangen. Auf der pro Seite lässt sich eindeutig feststellen, dass eine bereits bestehende Architektur mit Gewichten vorhanden gewesen wäre. Jedoch muss an der Stelle die Frage gestellt werden, ob diese Architektur zielführend für das eigene Problem ist. An Beispiel des VGG16 lässt sich dieses Problem sehr gut erkennen. Das Netzwerk wurde basierend auf der ImageNet-Datensammlung trainiert und klassifiziert auf 1000 verschiedene Klassen. Allerdings wird keine Segmentation-Map erzeugt, sondern lediglich erkannt ob ein Objekt auf dem Input-Bild ist oder nicht. Um eine Segmentation-Map zu erzeugen hätten die letzten Layer dieser Netzwerkarchitektur abgeschnitten werden müssen und durch eine eigene Struktur ersetzt werden. Dies ist technisch möglich, würde aber ein komplettes Trainieren der neuen Layer erfordern. Aus diesem Grund und dass solche vortrainierten Netzwerke für die Klassifizierung eher allgemeiner Bilder, z.B. erkennen ob ein Elefant, Auto oder Haus auf dem Bild vorhanden ist, ausgelegt sind und nicht für fachspezifischer Fragestellungen wie dem erkennen von Kiesflächen auf Satellitendaten, ist es für den Autor an dieses Stelle sinnvoller gewesen ein komplett neues Netzwerk aufzusetzen und dieses von Grund auf für das gegebene Problem zu trainieren.

Keras ist ein Framework für Machine Learning für Python. Durch seine gute Dokumentation und eine simple API ist dieses Framework besonders nutzerfreundlich und erlaubt es mit nur wenigen Zeilen Code bereits einfache Neuronale Netze zu konzipieren. Des Weiteren bietet Keras die Möglichkeit die Berechnungen wahlweise auf einer CPU oder GPU auszuführen. Insbesondere die Möglichkeit Neuronale Netze auf GPUs zu trainieren bringt unheimliche Performancevorteile mit (Schlegel, 2015). Daneben unterstützt Keras mit TensorFlow, CNTK und Theano gleich drei Machine-Learning-Bibliotheken und ermöglicht es dem Nutzer zwischen diesen drei Backend-Bibliotheken, ohne Änderungen an den mit Keras erzeugten Neuronalen Netzen hin und her zu wechseln. Dies macht Keras aktuell (Stand 2018) zu einem der beliebtesten Framework für Neuronale Netze (Hale, 2018) und wird nach offiziellen

Angaben im Webauftritt von Keras (Keras) auf der einen Seite von großen wissenschaftlichen Institutionen wie das CERN oder die NASA und auf der anderen Seite auch von Marktführern in der IT-Branche, wie Google, Microsoft, NVIDIA und Amazon genutzt und unterstützt. Diese Punkte haben den Autor schlussendlich dazu veranlasst das Neuronale Netz mit dem Keras-Framework aufzusetzen.

Die in Datenvorverarbeitung durchgeführte Datenvorverarbeitung ist mit Python 3.7.3 durchgeführt worden. Die verwendeten Bibliotheken sind im

Versionsverzeichnis zu finden. Die genutzte Hardware ist im Hardwareverzeichnis zu finden. Das in Kapitel 4.3 dargelegte Aufsetzen und Trainieren des CNNs ist mit Python 3.7.4 durchgeführt worden.

## 4.1 Verwendete Software

Um geeignete Trainingsflächen auszuwählen, zu standardisieren und für das CNN zugänglich zu machen, wurden verschiedene Python-Bibliotheken verwendet. Insbesondere wurden die NumPy-, Matplotlib-, sentinelsat- und osgeo-Bibliothek verwendet. Sentinelsat wurde dabei für den automatisierten Download der Rohdaten aus dem Copernicus-Hub der ESA benutzt.
NumPy und Matplotlib sind beides Bibliotheken aus dem Data-Science Umfeld. NumPy eignet sich hierbei besonders gut um Vektoren, Matrizen und große Arrays schnell, leicht und effizient zu verwalten. Matplotlib ist dagegen besonders gut für die Visualisierung mathematischer Informationen geeignet. Die osgeo-Bibliothek ist dafür konzeptziert um geographisch verortete Information schnell zu verarbeiten und ist in der Lage einen großen Teil, der aus dem GIS-Bereich bekannten Datenformate zu lesen und schreiben. Außerdem bietet die Bibliothek viele, der aus dem GIS-Bereich bekannten geographischen Datenoperationen und Datentransformationen durchzuführen.
Diese Bibliotheken wurden in den eigens programmierten Bibliotheken und Jupyter-Notebooks verwendet um die Daten herunterzuladen, aufzubereiten und das CNN zu bauen.
Neben diesen Bibliotheken wurde die Datenverarbeitungssoftware FME verwendet um Informationen aus PDFs herauszufiltern und zu speichern.

## 4.2 Datenvorverarbeitung

Zunächst wurde, um die Satellitendaten herunterzuladen, das Gebiet definiert, welches die Aufnahme schneiden muss um heruntergeladen zu werden. Anschließend wurde vom Internetauftritte des Geologischen Dienstes NRW[2] die Abgrabungsmonitoringsberichte von 2018 für die Bereiche Detmold und Düsseldorf im PDF-Format heruntergeladen. In diesen Berichten befinden sich Karten mit der ungefähren Verortung der aktuellen Kiesabbauflächen. Um diese Informationen zu extrahieren, wurde in FME ein Prozess modelliert (Abbildung 13), welcher die Koordinaten der Abbauregion und den Abbaugebieten ausliest und als .csv-Datei abspeichert. Da die so ausgelesenen Koordinaten nicht die Position in einem geographischen

[2] https://www.gd.nrw.de/ro_am.htm [10.06.2019]

Koordinatensystem beschreiben, sondern lediglich die absolute Position auf der PDF-Seite, sind diese Blattkoordinaten mit Hilfe der Methoden in ‚pdf2coords.py' in geographische Koordinaten überführt worden. Die vollzogene Verschiebung der Koordinaten funktioniert nur unter der Annahme, dass es sich bei dem Input- und Referenz-Koordinatensystem jeweils um Koordinatensysteme handelt, welche den durch das Koordinatensystem beschriebenen Teil der Erde als planare Ebene annehmen. Auf Nachfrage vom 01.August.2019 beim Geologischen Dienst NRW wurde bestätigt, dass das zugrundeliegende Referenzsystem der Karten in den Abgrabungsberichten ETRS89 / UTM Zone 32N ist. Während dem Referenzsystem zur Verschiebung der Koordinaten in ETRS89 / UTM Zone 32N vorliegen. Da die sowohl Input- als auch Referenzsystem Koordinatensysteme sind, die die Erde lokal als planare Oberfläche annehmen, konnte diese Überführung der Blattkoordinaten in geographische Koordinaten so vorgenommen werden. An dieser Stelle ist auch darauf hinzuweisen, dass es nicht das Ziel des Autors war exakte Positionen von Abbaugebieten zu errechnen. Dies wäre schon durch den Maßstab der Karten in den Monitoringsberichten nicht möglich gewesen. Es sollte vielmehr die ungefähre Position von Kiesabbauflächen bestimmt werden um im nächsten Schritt die Abbauflächen genauer zu definieren.

Im weiteren Verlauf wurden die so erzeugten Koordinaten der ungefähren Position der Kiesabbauflächen in QGIS eingeladen. Außerdem wurden mit dem OpenLayers-Plugin die aktuellen Satellitenkarten der Anbieter Google Maps und Bing Maps als Grundkarten eingeladen. Mit Hilfe dieser Grundkarten sind die Kiesabbauflächen noch einmal visuell geprüft und der Umriss der jeweiligen Kiesflächen eingezeichnet.

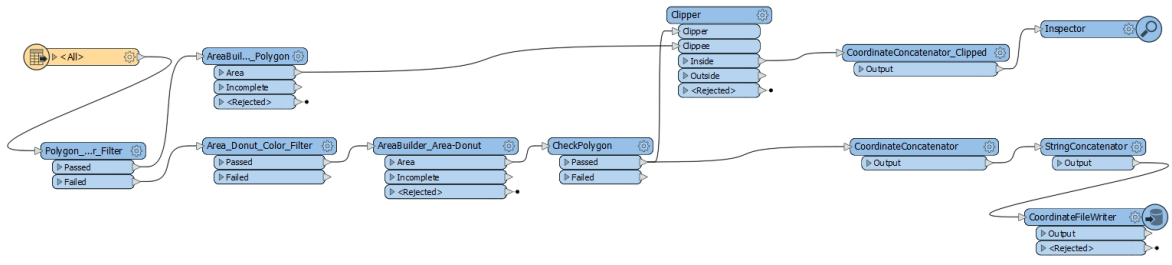


*Abbildung 13 - FME-Prozess zur Koordinatenextraktion*

### 4.2.1 Datendownload

In dem Jupyter Notebook, Jupyter ist eine Open-Source Entwicklungsumgebung, starter.ipynb ist der Prozess zum automatisierten Download der Daten verskriptet. Zunächst wird eine Anfrage mit der AOI und einem Zeitraum an die Downloadserver der ESA abgeschickt um den Download zu starten (Schritt 0, ‚Download of the desired area'). Im nächsten Schritt (3, ‚Check valid areas') wird überprüft, ob in der Scene überhaupt Kiesflächen zu finden sind. Dies geschieht durch das Verschneiden der Abbaugewässer mit der jeweiligen Ausdehnung der Szene. Anschließend werden die Kanäle 2 (blauer Bereich des elektromagnetischen Spektrums), 3 (grüner Bereich des elektromagnetischen Spektrums), 4 (roter Bereich des elektromagnetischen Spektrums) und 8 (naher Infrarotbereich des elektromagnetischen Spektrums) der Sentinel-2 Satelliten in der 10m Auflösung zu einem Geotiff zusammengeführt und abgespeichert (siehe Abbildung 14).

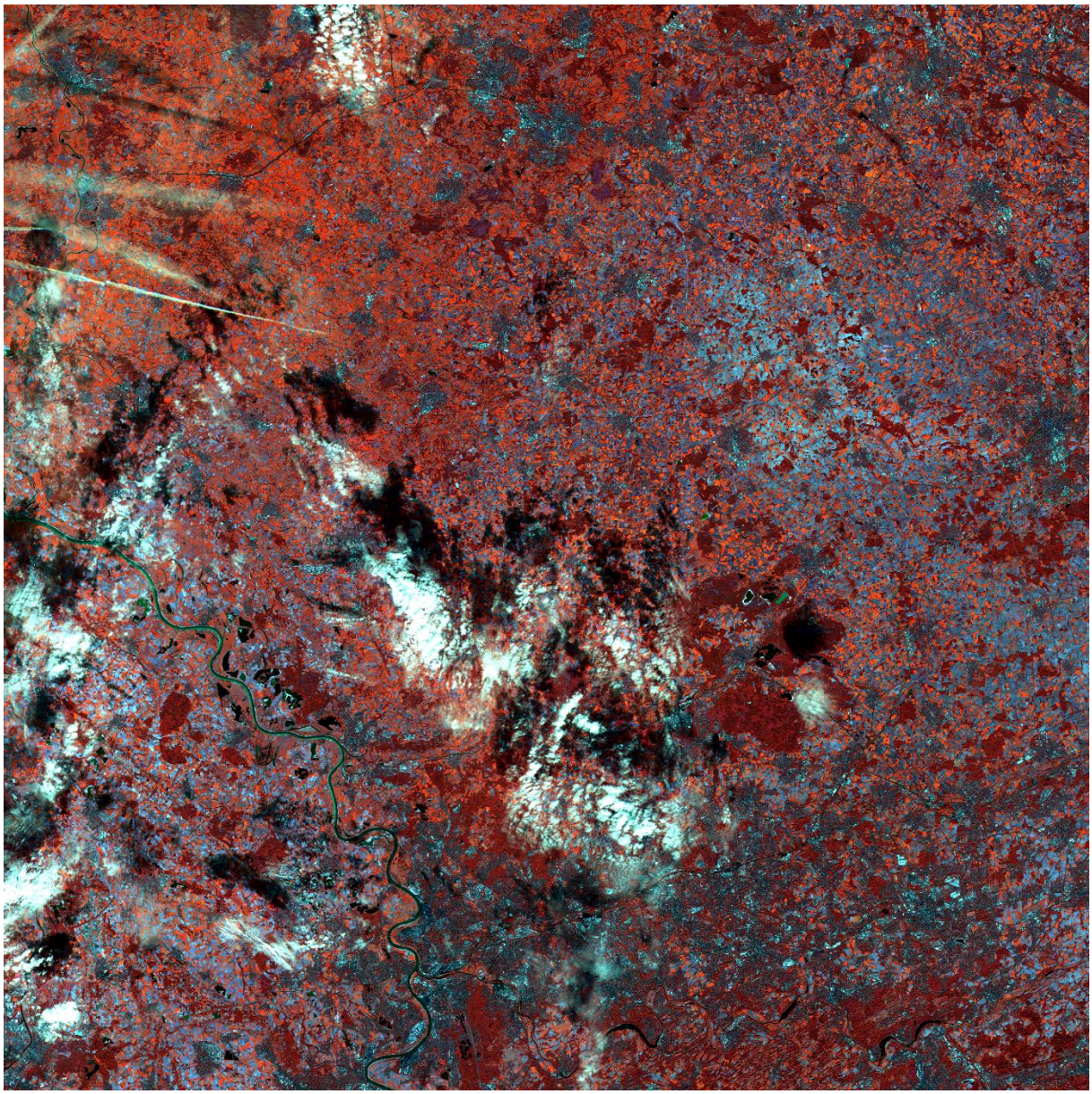

*Abbildung 14 - Beispielergebnis eines GeoTiffs in der Falschfarbendarstellung NIR, G, B*

Außerdem werden die Wolkenmasken und Metadatendateien separat abgespeichert. Um Speicherplatz zu sparen, wurden nicht mehr benötigte Daten gelöscht. Damit auf die einzelnen Kanäle, die Projektion und Transformation der Geotiffs in späteren Verarbeitungsschritten zugegriffen werden kann, sind die Szenen in Form eines Dictionary, im Python eigenen Pickleformat, abgespeichert worden. Diese sind Folgendermaßen aufgebaut:

$$dict = \{\text{"b"}: b, \text{"g"}: g, \text{"r"}: r, \text{"nir"}: nir, \text{"clouds"}: cloud,$$
$$\text{"geotransform"}: transform, \text{"projection"}: projection, \text{"bin"}: bin$$

Dabei werden in ‚b' die Daten aus Kanal 2, in ‚g' die Daten aus Kanal 3, in ‚r' die Daten aus Kanal 4 und in ‚nir' die Daten aus Kanal 8 hinterlegt. ‚clouds' enthält das mitgelieferte Wolkenraster, welches von der Ursprünglichen 20 x 20m Auflösung auf eine Auflösung von

10 x 10m pro Rasterzelle gesampelt wird. Dies geschieht, um später die Bildinformationen besser mit den Wolkendaten verrechnen zu können. In ‚bin' wird später die Binärmaske zu den Trainingsgebieten hinterlegt, während ‚geotransform' und ‚projection' Informationen zu der Georeferenzierung der Rasterdaten enthalten, um diese jederzeit in ein georeferenzierte Rasterdatei überführen zu können. Gleichzeitig wurden in dem Verarbeitungsschritt die einzelnen Pixelwerte der Kanäle 2, 3, 4 und 8 auf den Wertebereich 0 - 255 standardisiert.

### 4.2.2 Datenprozessierung

In dem Jupyter Notebook masks.ipynb ist der Prozess zum Erstellen der Klassifizierungsmasken programmiert. Hierzu wird zunächst der NDWI, skaliert auf einen Wertebereich von -1 bis 1, einer Szene berechnet und anschließend mit den Abbaugewässern bildlich dargestellt (Abbildung 15).

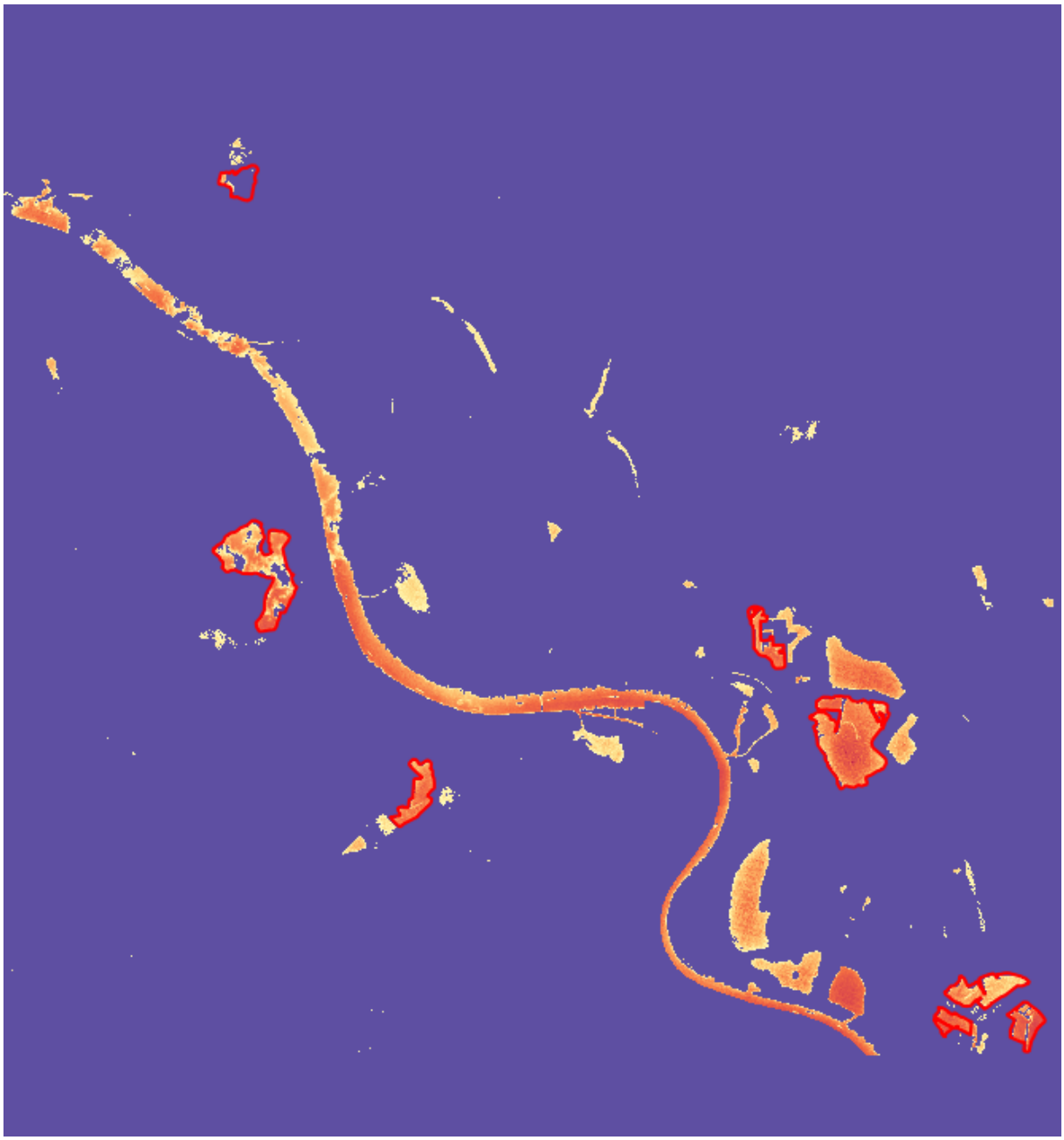

*Abbildung 15 - Ausschnitt aus dem Auswahlprozess (bekannte Kiesflächen Rot umrandet)*

Die einzelnen Abbaugebiete werden zunächst händisch ausgewählt und anschließend per Floodfillalgorithmus in eine Binärmaske umgewandelt. Als Schwellwert für den Floodfillalgorithmus wurden die Werte 0.0 (untere Grenze) und 1.0 (obere Grenze) gewählt. Diese Schwellwerte wurden, um auch sedimentreiche Wasserflächen zu extrahieren, basierend auf dem Paper von (Gao, 1996) gewählt. Der so erzeugte Binärlayer (Abbildung 16) ist anschließend dem zugehörigen Dictionary hinzugefügt worden.

*Abbildung 16 – Binärmaske zu Abbildung 15*

Im nächsten Schritt wurden die Szenen mit der Ausdehnung der Abbauregion Detmold und Düsseldorf verschnitten.

Anschließend sind in dem Jupyter Notebook subsetting.ipynb 512 x 512 Pixel große Ausschnitte um die Kiesabbauflächen eingezeichnet und diese anschließend ausgeschnitten. So wurden insgesamt 61 Szenen erzeugt, welche Kiesabbauflächen enthalten.

## 4.3 Das Convolutional Neural Network

Für den Trainingsprozess und zum Erzeugen des CNNs müssen zunächst die in Tabelle 1 erläuterten Parameter in den Jupyter Notebooks im Ordner ‚ml_notebooks' als Hyperparamter gesetzt werden.

| Hyperparameter | Funktion |
|---|---|
| network_name | Speichern und Zuordnen von Trainingsergebnissen |
| w, im_width, h, im_height | Definiert die Ausdehnung des Inputlayer |
| c, layer | Definiert die Tiefe des Inputlayer |
| base_pos | Dateipfad zu den Trainingsdaten |
| augs_per_img | Wie oft soll jede Trainingsszene augmentiert werden |
| ts | Prozentualer Anteil der Validierungsdaten an den Trainingsdaten |
| model_name | Dateiname des erzeugten Models |
| his_name | Dateiname des vom Training erzeugten history-Objekts |

*Tabelle 1 - Hyperparameter*

Im Anschluss an das Setzen der globalen Parameter werden zunächst pickle-Dateien aus ‚base_pos' in das Jupyter Notebook geladen, für den Input verarbeitet und augmentiert. Hierbei wird der Ursprüngliche Datensatz um weitere Daten erweitert. Dies geschieht, um mehr Varianz in dem Trainingsdatensatz zu erzeugen und eine Überanpassung des CNNs auf die Trainingsdaten zu vermeiden (Chollet, 2018, S. 184). Für die Augmentierung der Szenen ist die imgaug-Bibliothek verwendet worden.

Das U-Net selber setzt sich aus mehreren Layern nach dem Vorbild des von Ronneberger et al. entwickelten U-Nets zusammen (Abbildung 12). Dabei besteht der „contracting path" aus fünf Blöcken, welche jeweils zwei Conv2D-, Activation- und BatchNormalization-Layer bestehen. Zusätzlich enthalten die Blöcke 1 - 4 jeweils noch einen MaxPooling2D- und Dropout-Layer. Der „expansive path" besteht aus vier Blöcken. Diese setzen sich wiederum aus jeweils fünf Layern zusammen. Dem Conv2DTranspose-, concatenate-, Dropout-Layer und einem conv2d_block, bestehend aus jeweils zwei Conv2D-, Activation- und BatchNormalization-Layer. Zusätzlich enthält der expansive path noch einen Conv2D-Outputlayer für die Ausgabe

der Klassifizierungsmaske. Zum Aufsetzen des U-Nets wurden die in Tabelle 2 - Tabelle 10 beschriebenen Variablenwerte verwendet.

| Variable | Wert |
|---|---|
| shape | Dimensionen der ankommenden Inputvektoren. Wird in den Hyperparamtern ‚im_width', ‚im_height' und ‚layer' festgelegt |
| batch_shape | Default |
| name | Legt den Namen für den Layer fest. Muss einzigartig sein |
| dtype | Default |
| sparse | Default |
| Tensor | Default |

*Tabelle 2 - Erzeugen eines Keras-Tensors*

| Variable | Wert |
|---|---|
| input_img | Dimensionen der ankommenden Vektoren (Tabelle 2) |
| n_filters | 16, Beschreibt die Dimension des Outputspace. Bei vielen aufeinanderfolgenden Layern zunächst eine kleiner Zweierpotenz wählen, da der Outputspace für jeden Block verdoppelt wird |
| kernel_size | 3, Default nach (Ronneberger, Fischer, & Brox, 2015). Beschreibt die Größe des Fensters, in dem Werte zusammengefasst werden |
| dropout | 0.05, Vermeidet Überanpassung beim Lernprozess durch zufälliges nullen von Inputwerten. Wert zwischen 0-1, der angibt wie viele Inputwerte genullt werden. Für Denoising AutoEncoder ist der optimale Wert 0.05 (Srivastava, Hinton, Krizhevsky, Sutskever, & Salakhutdinov, 2014) |

*Tabelle 3 - Erzeugen des U-Nets*

| Variable | Wert |
|---|---|
| filters | 16, wird pro Layer verdoppelt (Tabelle 3) |
| kernel_size | 3 x 3, Default nach (Ronneberger, Fischer, & Brox, 2015) |
| strides | Default |
| padding | same, Outputlayer haben so dieselben Dimensionen wie Inputlayer |
| data_format | Default |
| dilation_rate | Default |
| activation | relu, Default nach (Ronneberger, Fischer, & Brox, 2015) |
| use_bias | Default |
| kernel_initializer | he_normal, Initiale Gewichtsmatrix. |
| bias_initializer | Default |
| kernel_regularizer | Default |
| bias_regularizer | Default |
| activity_regularizer | Default |
| kernel_constraint | Default |
| bias_constraint | Default |
| Batchnormalization | True, Dient dazu die Trainingszeit zu verbessern (Ioffe & Szegedy, 2015) |

*Tabelle 4 - Erzeugen eines Conv2D-Layer*

| Variable | Wert |
|---|---|
| pool_size | (2, 2), Default nach (Ronneberger, Fischer, & Brox, 2015) |
| strides | Default |
| padding | Default |
| data_format | Default |

*Tabelle 5 - Erzeugen eines MaxPooling2D-Layer*

| Variable | Wert |
|---|---|
| rate | 0.05, (Tabelle 3, dropout) |
| noise_shape | Default |
| seed | Default |

*Tabelle 6 - Erzeugen eines Dropout-Layer*

| Variable | Wert |
|---|---|
| filters | 128, funktioniert dem „contracting path“ entgegengesetzt (Tabelle 3, n_filters) |
| kernel_size | 3 x 3, Bewirkt, dass am Ende der Output dieselbe Höhe und Breite wie der Input aufweist |
| strides | (2, 2), Bewirkt, dass das Convolutionfenster bis zum Ende der jeweiligen Zeile/Spalte läuft |
| padding | same, Outputlayer haben so dieselben Dimensionen wie Inputlayer |
| output_padding | Default |
| data_format | Default |
| dilation_rate | Default |
| activation | Default |
| use_bias | Default |
| kernel_initializer | Default |
| bias_initializer | Default |
| kernel_regularizer | Default |
| bias_regularizer | Default |
| activity_regularizer | Default |
| kernel_constraint | Default |
| bias_constraint | Default |

*Tabelle 7 - Erzeugen eines Conv2DTranspose-Layer*

| Variable | Wert |
|---|---|
| tensors | Aktueller Conv2DTranspose-Layer und dem gegenüberliegenden conv2d_block (Abbildung 12) |
| axis | Default |

*Tabelle 8 - concatenate-Layer*

| Variable | Wert |
|---|---|
| filters | 1, Es wird nur ein Ausgabelayer benötigt |
| kernel_size | 1 x 1, Default nach (Ronneberger, Fischer, & Brox, 2015) |
| strides | Default |
| padding | Default |
| data_format | Default |
| dilation_rate | Default |
| activation | sigmoid, Standardfunktion um Ergebnisse als Wahrscheinlichkeit zu beschreiben |
| use_bias | Default |
| kernel_initializer | Default |
| bias_initializer | Default |
| kernel_regularizer | Default |
| bias_regularizer | Default |
| activity_regularizer | Default |
| kernel_constraint | Default |
| bias_constraint | Default |

*Tabelle 9 - Erzeugen des Output-Layer*

| Variable | Wert |
|---|---|
| inputs | input_img, der als Keras-Tensor erzeugte Input-Layer |
| outputs | outputs, der in Tabelle 9 erzeugte Output-Layer |

*Tabelle 10 - Erzeugen des Models*

Anschließend wird das erzeugte Model mittels der compile-Funktion für das Training konfiguriert. Dabei sind die in Tabelle 11 beschriebenen Variablenwerte verwendet worden.

| Variable | Wert |
|---|---|
| optimizer | Adam, Beste Optimierungsfunktion für große Datensets mit hochdimensionalen Parameterräumen (Kingma & Ba, 2014) |
| loss | binary_crossentropy, Standardfunktion für Modelle mit einer Wahrscheinlichkeitsausgabe (Chollet, 2018, S. 102) |
| metrics | accuracy, Gibt die Klassifikationsgenauigkeit an. Typische Metrik, für Klassifikationsaufgaben. |
| loss_weights | Default |
| sample_weight_mode | Default |
| weighted_metrics | Default |
| target_tensors | Default |

*Tabelle 11 - Konfiguration des Models*

Außerdem sind die in Tabelle 12 aufgezählten callbacks dem Model hinzugefügt worden. Die dazu verwendeten Instanziierungsvariablen werden in den Tabelle 13 - Tabelle 15 genauer erläutert.

| Callback | Funktion |
|---|---|
| EarlyStopping | Bricht das Training ab, wenn die gewählte Einheit sich nicht weiter verbessert |
| ReduceLROnPlateau | Verringert die Lernrate, wenn der Algorithmus stagniert. Vermeidet so, dass nur lokale Minima aber nicht globale Minima gefunden werden (Surmenok, 2017) |
| ModelCheckpoint | Speichert das Model nach jeder Epoche |

*Tabelle 12 - Verwendete Callbacks*

| Variable | Wert |
|---|---|
| monitor | Default |
| min_delta | Default |
| patience | 10, Anzahl der Epochen ohne Verbesserung um das Training zu unterbrechen. Gewählt in Abhängigkeit des patience-Wertes aus Tabelle 14. So wird die Lernrate zunächst zweimal verringert, bevor entschieden wird, dass das Model sich nicht mehr verbessert und weitere Iterationen unnütz sind |
| verbose | 1, Ausgabe der Werte (not quiet) |
| mode | Default |
| baseline | Default |
| restore_best_weights | Default |

*Tabelle 13 – EarlyStopping*

| Variable | Wert |
|---|---|
| monitor | Default |
| factor | 0.1, Faktor, mit dem die alte Trainingsrate multipliziert wird um die neue Trainingsrate zu erzeugen. Dieser Wert bewirkt, dass bei einer zu großen initialen Lernrate diese sich schnell verringert und somit keine Minima ‚übersieht', gleichzeitig werden geringe Lernraten nicht drastisch verringert und der Trainingsprozess dadurch gebremst |
| patience | 5, Anzahl der Epochen ohne Verbesserung um die Lernrate zu verringern. Wert wurde gezielt auf 1/10 der maximalen Epochen (Tabelle 16, epochs) gesetzt. Dies hat zum einen den Grund nicht zu viele Epochen zu verschwenden in denen das CNN nichts dazu lernt. Zum anderen aber auch nicht zu schnell die Lernrate zu reduzieren und somit das Training zu verlangsamen |
| verbose | 1, Ausgabe der Werte (not quiet) |
| mode | Default |
| min_delta | Default |
| cooldown | Default |
| min_lr | 0.00001, Gibt die untere Grenze für die Lernrate an. Wird diese Erreicht wird die Lernrate nicht weiter reduziert. Eine zu kleine Lernrate würde zu einem sehr langsamen Training führen. Deswegen ist der Wert auf 1/100 der ursprünglichen Anfangslernrate gesetzt |

*Tabelle 14 – ReduceLROnPlateau*

| Variable | Wert |
|---|---|
| filepath | model_name, (Tabelle 1, model_name) |
| monitor | Default |
| verbose | 1, Ausgabe der Werte (not quiet) |
| save_best_only | True, Nur das aktuell beste Model wird gespeichert. Ältere, schlechtere Modelle werden überschrieben |
| save_weights_only | False, Das gesamte Model wird gespeichert. Erlaubt es später das Model einfach zu laden ohne zuerst ein passendes erzeugen zu müssen dem anschließende die Gewichte zugewiesen werden |
| mode | Default |
| period | Default |

*Tabelle 15 – ModelCheckpoint*

Nach der Instanziierung des Models kann im nächsten Schritt durch die fit-Methode mit dem Training des U-Net begonnen werden. Die Tabelle 16 enthält und erläutert die dazu verwendeten Parameter.

| Variable | Wert |
|---|---|
| x | X_train, NumPy-Array, welches die Trainingsdaten enthält |
| y | y_train, NumPy-Array, welches die Klassifizierungsmasken enthält |
| batch_size | 8, Typischerweise eine Zweierpotenz. Bedingt durch die geringe Menge an Trainingsdaten wurde ein Werte gewählt, der eine regelmäßige Gradientenaktualisierung ermöglicht, aber nicht zu klein ist und das Training langsam werden lässt (Brownlee, 2019) |

| | |
|---|---|
| epochs | 50, Anzahl der maximalen Trainingsepochen. Eine Epoche ist beendet, wenn sämtliche Trainingsdaten einmal vom Netzwerk verarbeitet wurden. Dieser Wert ist als Kompromiss zwischen einer vertretbaren Trainingsdauer und einem guten Klassifizierungsergebnisses gewählt worden |
| verbose | 1, Ausgabe der Werte (not quiet) |
| callbacks | EarlyStopping, ReduceLROnPlateau, ModelCheckpoint, vgl. Tabelle 12 - Verwendete Callbacks-Tabelle 15 |
| validation_split | Default |
| validation_data | (X_valid, y_valid), NumPy-Arrays mit den Validierungsdaten (X_valid) und den zugehörigen Klassifizierungsmasken (y_valid) |
| shuffle | Default |
| class_weight | Default |
| sample_weight | Default |
| initial_epoch | Default |
| steps_per_epoch | Default |
| validation_steps | Default |
| validation_freq | Default |

*Tabelle 16 - Trainieren des Models*

### 4.3.1 Allgemeiner Ablauf des Trainingsprozesses

In diesem Abschnitt werden zunächst die sich nicht ändernden Prozesse der in Kapitel Jupyter Notebook „kies_pos_nir_unet“Jupyter Notebook „kies_pos_rgb_unet“ genauer erläuterten Jupyter Notebooks behandelt. Zunächst werden die in Tabelle 17 genannten Hyperparameter mit den entsprechenden Werten gesetzt.

| Hyperparameter | Wert |
|---|---|
| base_pos | LocalPathTo\\sentinel\\cnn_data\\pos\\ndwi_based\\small\\ |
| h, im_width | 512 |
| w, im_height | 512 |
| augs_per_img | 2 |
| ts | 0.1 |

*Tabelle 17 - Gleichbleibende Hyperparameter*

Anschließend werden die Trainingsdaten aus dem in dem Hyperparameter ‚base_pos‘ definierten Verzeichnis geladen, sowie auf den Wertebereich [0-1] standardisiert. Außerdem wird jede Szene zweimal unterschiedlich augmentiert. Die verwendeten Augmentationsverfahren werden dabei für jede Augmentation neu und zufällig ausgewählt. Anschließend wird der Trainingsdatensatz in einen Validierungsdatensatz und den eigentlichen Trainingsdaten zufällig aufgeteilt. Im darauffolgenden Schritt wird zunächst das U-Net aufgesetzt und konfiguriert (Tabelle 2 - Tabelle 16) und durch den Aufruf ‚model.fit‘ mit dem trainieren des Neuronalen Netzes begonnen. Dabei wird das beste Model, sowie die Trainingshistorie gespeichert, bevor die Klassifikationsgenauigkeit auf dem Validierungsdatensatz evaluiert und gespeichert wird.

Bei der Klassifizierung von Szenen bezüglich darin enthaltener Kiesabbauflächen wird durch das Modell eine Heatmap erzeugt. Diese Heatmap besitzt die gleiche Auflösung wie das Eingangsbild. Dabei enthält jeder Pixel einen Wert von 0 - 1. Dieser Wert beschreibt die Wahrscheinlichkeit, mit der der Pixel zu einer Kiesabbaufläche gehört.

### 4.3.2 Jupyter Notebook „kies_pos_nir_unet“

In diesem Jupyter Notebook sind die in Tabelle 18 genannten Hyperparameter verwendet worden.

| Hyperparameter | Variable |
|---|---|
| network_name | kies_pos_nir_unet |
| layer | 1 |
| model_name | kies_pos_nir_unet.h5 |
| his_name | kies_pos_nir_unet_history.p |

*Tabelle 18 - Hyperparameter Jupyter Notebook „kies_pos_unet"*

Für das Training sind die Trainingsdaten aus ‚nir‘ (siehe Datenprozessierung) verwendet worden. Insgesamt wurde das Model über 50 von 50 Epochen trainiert und das beste Model in Epoche 47 erzielt. Auf den Validierungsdaten wurde eine Klassifikationsgenauigkeit von ca. 98,35% erreicht. Abbildung 17 und Abbildung 18 zeigen den genauen Verlauf der Klassifikationsgenauigkeit und der Verlustfunktion. Basierend auf diesen Abbildungen ist die Gleichmäßigkeit des Lernprozesses auffällig. Das CNN scheint in der Lage zu sein, sehr schnell relevanten Informationen zu extrahieren. Lediglich um die zehnte Epoche kommt es zu einer kurzfristigen Überanpassung der Gewichte auf die Trainingsdaten. Diese wird jedoch in den folgenden Epochen wieder ausgeglichen.

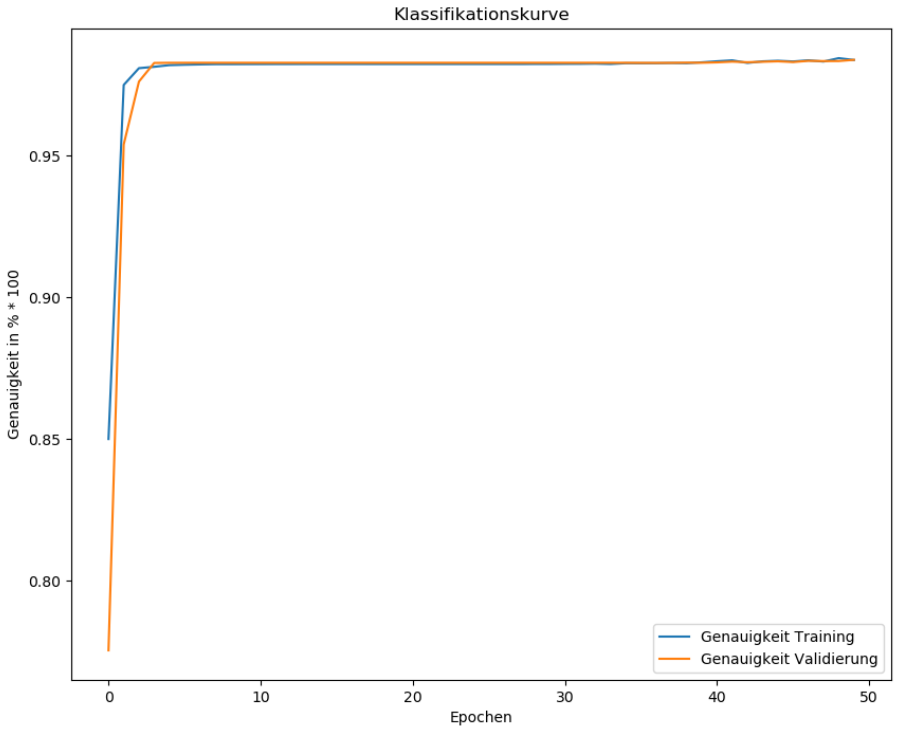


*Abbildung 17 - Klassifikationsgenauigkeitsverlauf „kies_pos_nir_unet"*

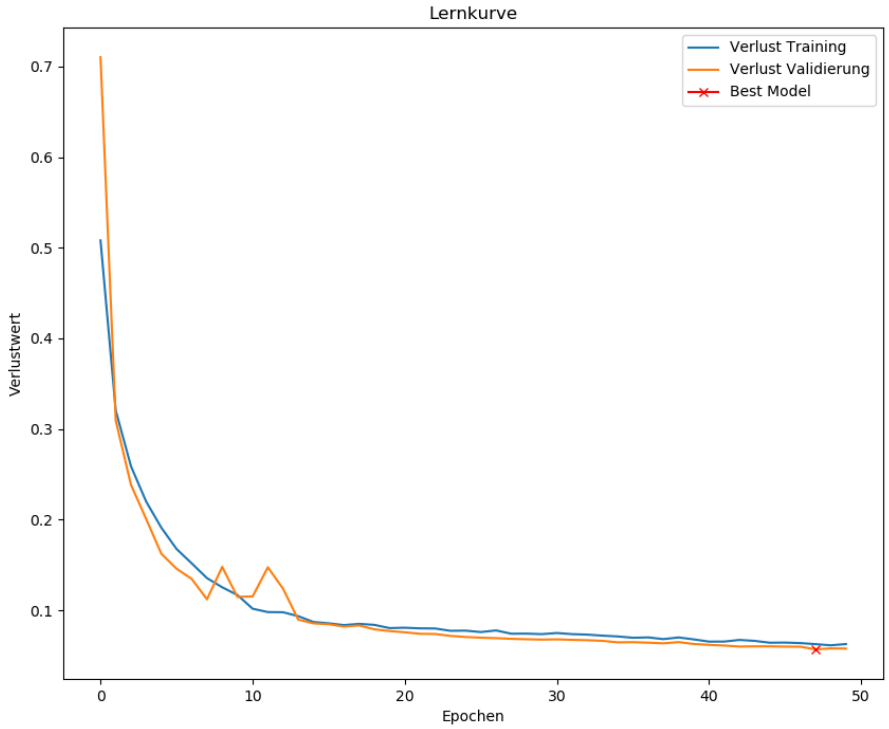


*Abbildung 18 - Verlustfunktionsverlauf „kies_pos_nir_unet"*

### 4.3.3 Jupyter Notebook „kies_pos_nirr_unet”

In diesem Jupyter Notebook sind die in Tabelle 19 genannten Hyperparameter verwendet worden.

| Hyperparameter | Variable |
|---|---|
| network_name | kies_pos_nirr_unet |
| layer | 2 |
| model_name | kies_pos_nirr_unet.h5 |
| his_name | kies_pos_nirr_unet_history.p |

*Tabelle 19 - Hyperparameter Jupyter Notebook „kies_pos_nirr_unet"*

Für das Training sind die Trainingsdaten aus ‚r‘ und ‚nir‘ (siehe Datenprozessierung) verwendet worden. Insgesamt wurde das Model über 50 von 50 Epochen trainiert und das beste Model in Epoche 44 erzielt. Auf den Validierungsdaten wurde eine Klassifikationsgenauigkeit von ca. 99,08% erreicht. Abbildung 19 und Abbildung 20 zeigen den genauen Verlauf der Klassifikationsgenauigkeit und der Verlustfunktion. Basierend auf diesen Abbildungen ist die immer wiederkehrende Überanpassung des Modells auf die Trainingsdaten auffällig. Diese nimmt im weiteren Verlauf des Trainings immer weiter ab und erreicht in Epoche 44 ihr Minima, bevor sie wieder leicht zu nimmt.

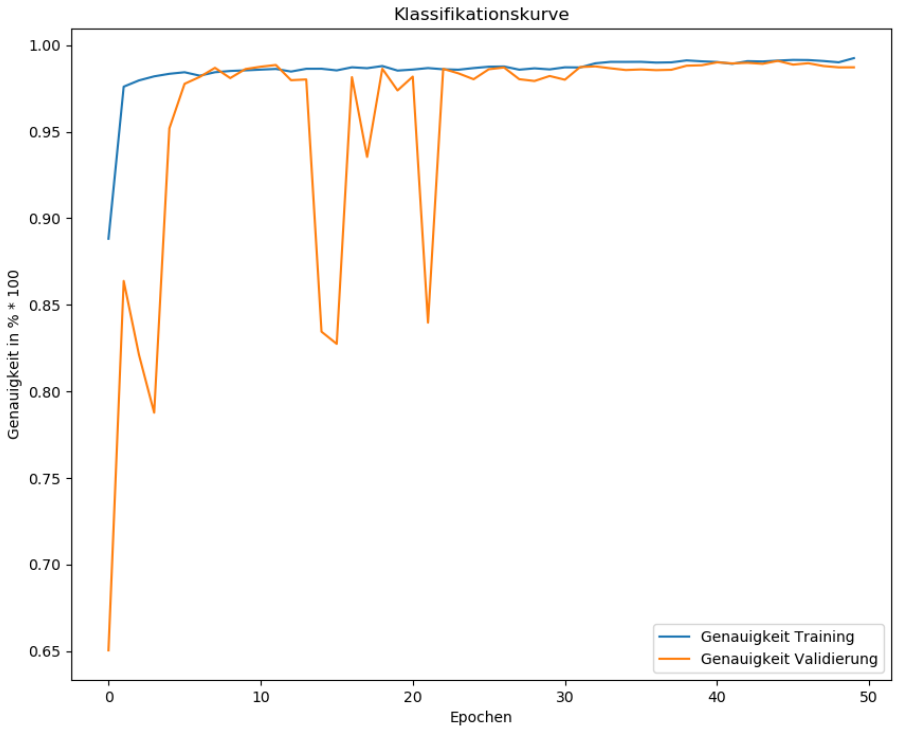


*Abbildung 19 - Klassifikationsgenauigkeitsverlauf „kies_pos_nirr_unet"*

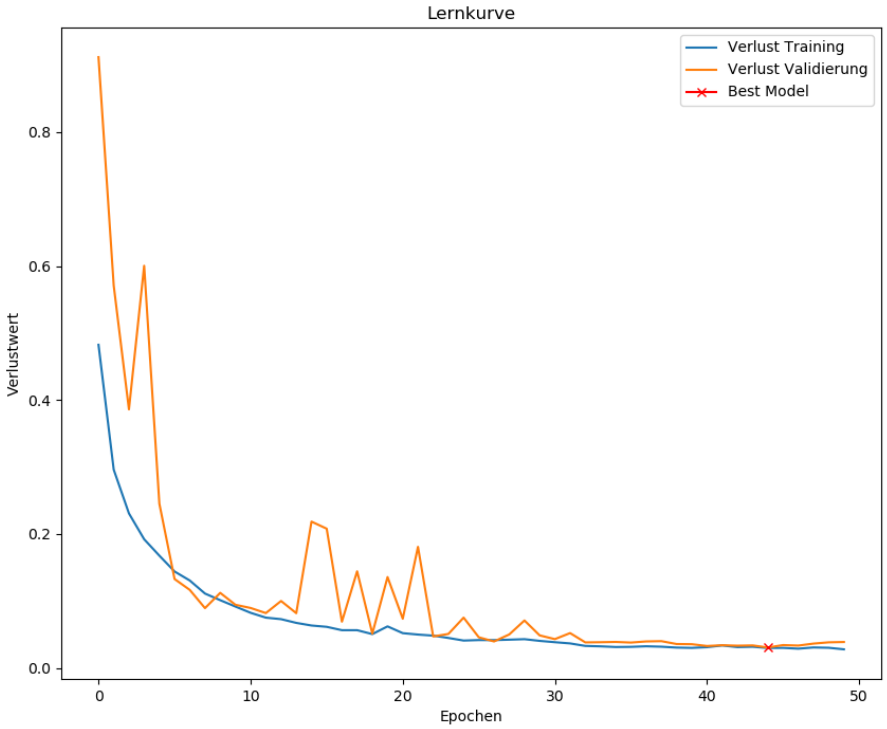


*Abbildung 20 - Verlustfunktionsverlauf „kies_pos_nirr_unet“*

### 4.3.4 Jupyter Notebook „kies_pos_nirg_unet“

In diesem Jupyter Notebook sind die in Tabelle 20 genannten Hyperparameter verwendet worden.

| Hyperparameter | Variable |
|---|---|
| network_name | kies_pos_nirg_unet |
| layer | 2 |
| model_name | kies_pos_nirg_unet.h5 |
| his_name | kies_pos_nirg_unet_history.p |

*Tabelle 20- Hyperparameter Jupyter Notebook „kies_pos_nirg_unet"*

Für das Training sind die Trainingsdaten aus ‚g‘ und ‚nir‘ (siehe Datenprozessierung) verwendet worden. Insgesamt wurde das Model über 46 von 50 Epochen trainiert und das beste Model in Epoche 35 erzielt. Auf den Validierungsdaten wurde eine Klassifikationsgenauigkeit von ca. 99,28% erreicht. Abbildung 21 und Abbildung 22 zeigen den genauen Verlauf der Klassifikationsgenauigkeit und der Verlustfunktion. Basierend auf diesen Abbildungen ist die zunächst sehr schlechte Anpassung der Gewichtungen auffällig. Trotzdem ist das CNN in der Lage, eine bessere Gewichtsverteilung zu bestimmen. Um Epoche 12 werden zunächst die Gewichtungen an die Trainingsdaten überangepasst, erreichen kurz darauf aber trotzdem eine Verteilung in dem das CNN nur noch langsam und verhältnismäßig wenig dazu lernt.

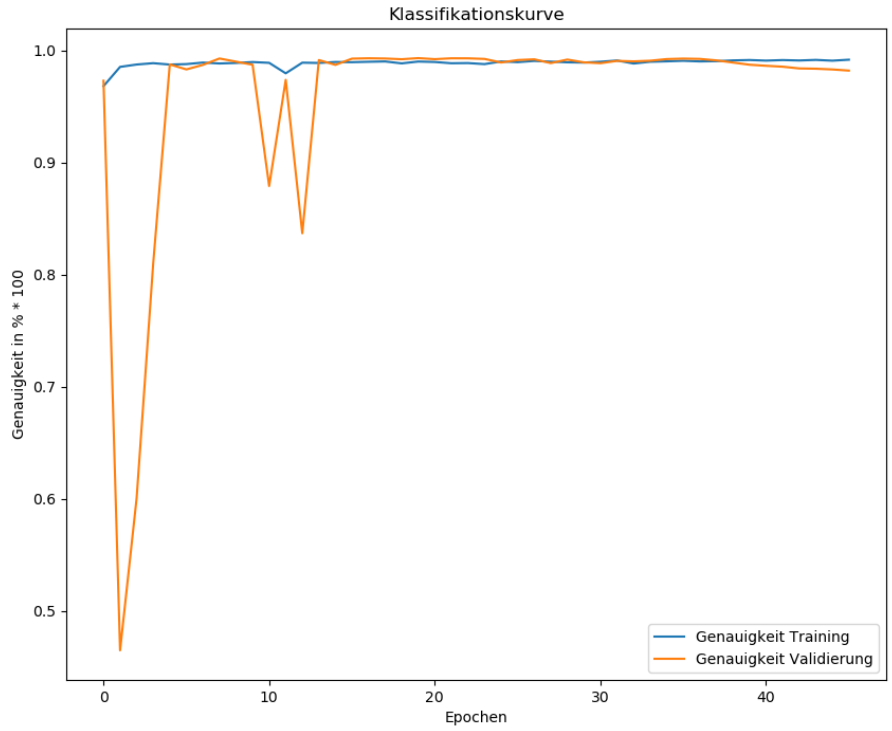


*Abbildung 21 - Klassifikationsgenauigkeitsverlauf „kies_pos_nirg_unet"*

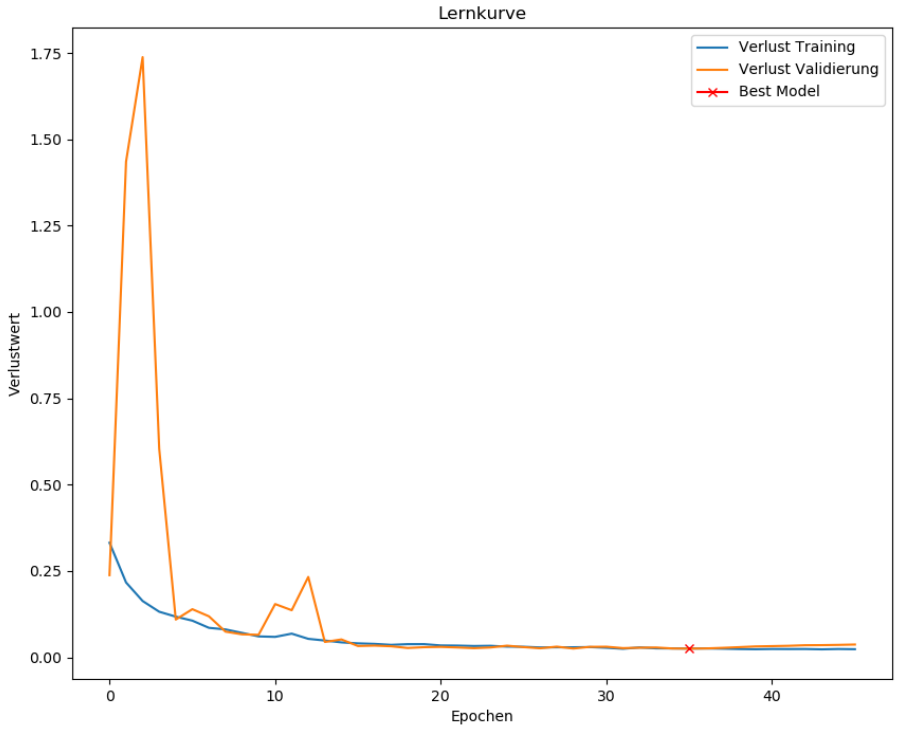


*Abbildung 22 - Verlustfunktionsverlauf „kies_pos_nirg_unet"*

### 4.3.5 Jupyter Notebook „kies_pos_nirb_unet“

In diesem Jupyter Notebook sind die in Tabelle 21 genannten Hyperparameter verwendet worden.

| Hyperparameter | Variable |
|---|---|
| network_name | kies_pos_nirb_unet |
| layer | 2 |
| model_name | kies_pos_nirb_unet.h5 |
| his_name | kies_pos_nirb_unet_history.p |

*Tabelle 21 - Hyperparameter Jupyter Notebook „kies_pos_nirb_unet"*

Für das Training sind die Trainingsdaten aus ‚b‘ und ‚nir‘ (siehe Datenprozessierung) verwendet worden. Insgesamt wurde das Model über 50 von 50 Epochen trainiert und das beste Model in Epoche 49 erzielt. Auf den Validierungsdaten wurde eine Klassifikationsgenauigkeit von ca. 99,61% erreicht. Abbildung 23 und Abbildung 24 zeigen den genauen Verlauf der Klassifikationsgenauigkeit und der Verlustfunktion. Basierend auf diesen Abbildungen ist die Gleichmäßigkeit des Lernprozesses auffällig. Das CNN scheint in der Lage zu sein, sehr schnell relevante Informationen zu extrahieren. Es findet keine extreme Überanpassung des Modells an die Trainingsdaten statt und die Lernkurve ist insgesamt kaum mit Ausreißern versehen.

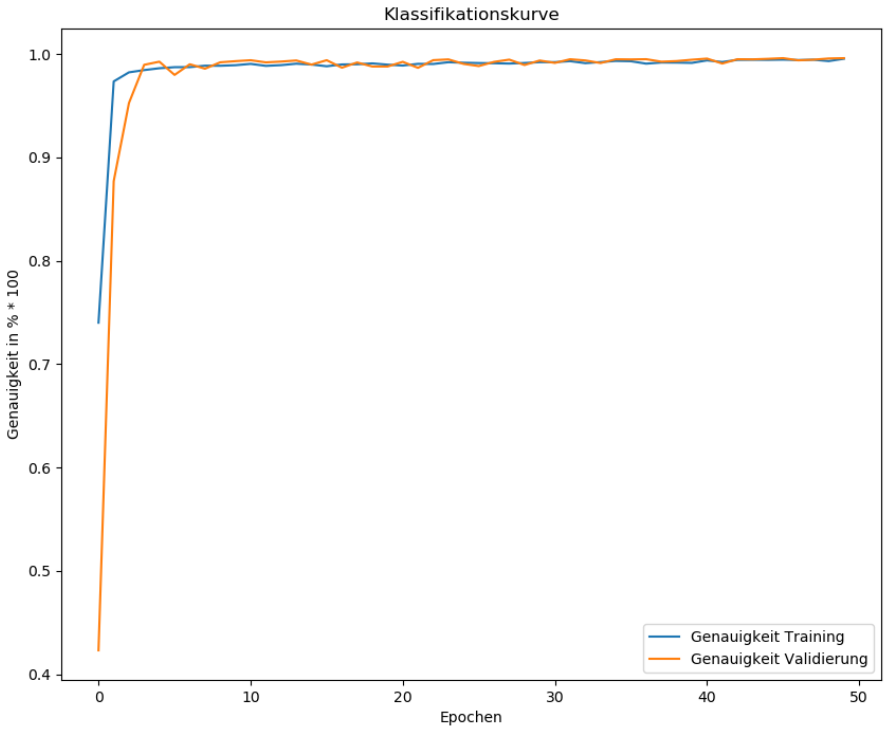


*Abbildung 23 - Klassifikationsgenauigkeitsverlauf „kies_pos_nirb_unet"*

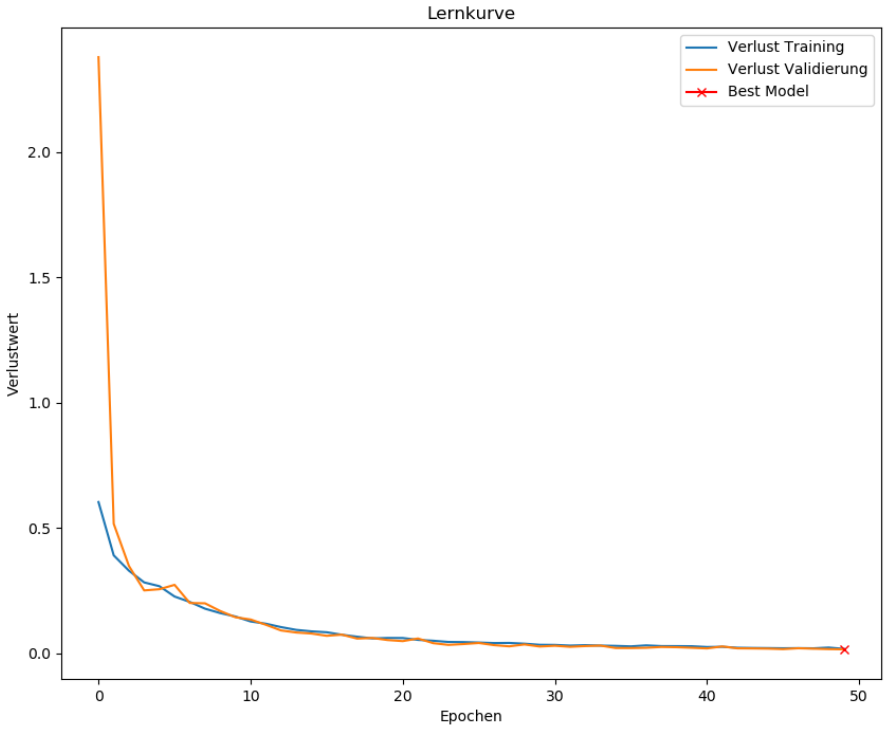


*Abbildung 24 - Verlustfunktionsverlauf „kies_pos_nirb_unet"*

### 4.3.6 Jupyter Notebook „kies_pos_rgb_unet“

In diesem Jupyter Notebook sind die in Tabelle 22 genannten Hyperparameter verwendet worden.

| Hyperparameter | Variable |
|---|---|
| network_name | kies_pos_rgb_unet |
| layer | 3 |
| model_name | kies_pos_rgb_unet.h5 |
| his_name | kies_pos_rgb_unet_history.p |

*Tabelle 22 - Hyperparameter Jupyter Notebook „kies_pos_rgb_unet"*

Für das Training sind die Trainingsdaten aus ‚r‘, ‚g‘ und ‚b‘ (siehe Datenprozessierung) verwendet worden. Insgesamt wurde das Model über 49 von 50 Epochen trainiert und das beste Model in Epoche 38 erzielt. Auf den Validierungsdaten wurde eine Klassifikationsgenauigkeit von ca. 99,64% erreicht. Abbildung 25 und Abbildung 26 zeigen den genauen Verlauf der Klassifikationsgenauigkeit und der Verlustfunktion. Basierend auf diesen Abbildungen ist die Gleichmäßigkeit des Lernprozesses auffällig. Das CNN scheint in der Lage zu sein, sehr schnell relevante Informationen zu extrahieren. Lediglich um die 22 Epoche kommt es zu einer kurzfristigen Überanpassung der Gewichte auf die Trainingsdaten. Diese wird jedoch in den folgenden Epochen wieder ausgeglichen.

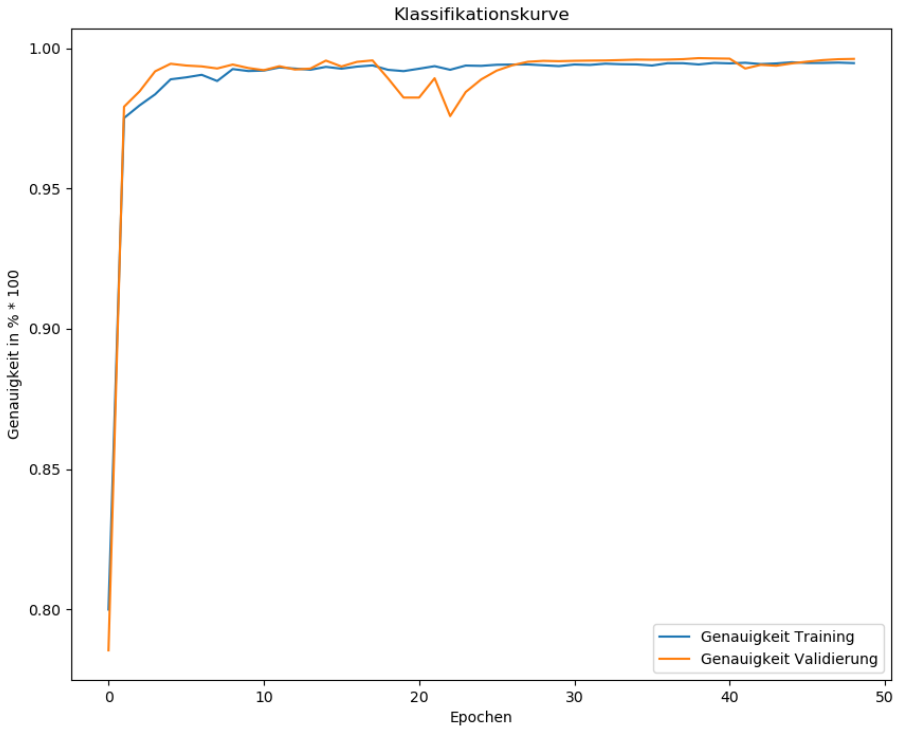


*Abbildung 25 - Klassifikationsgenauigkeitsverlauf „kies_pos_rgb_unet"*

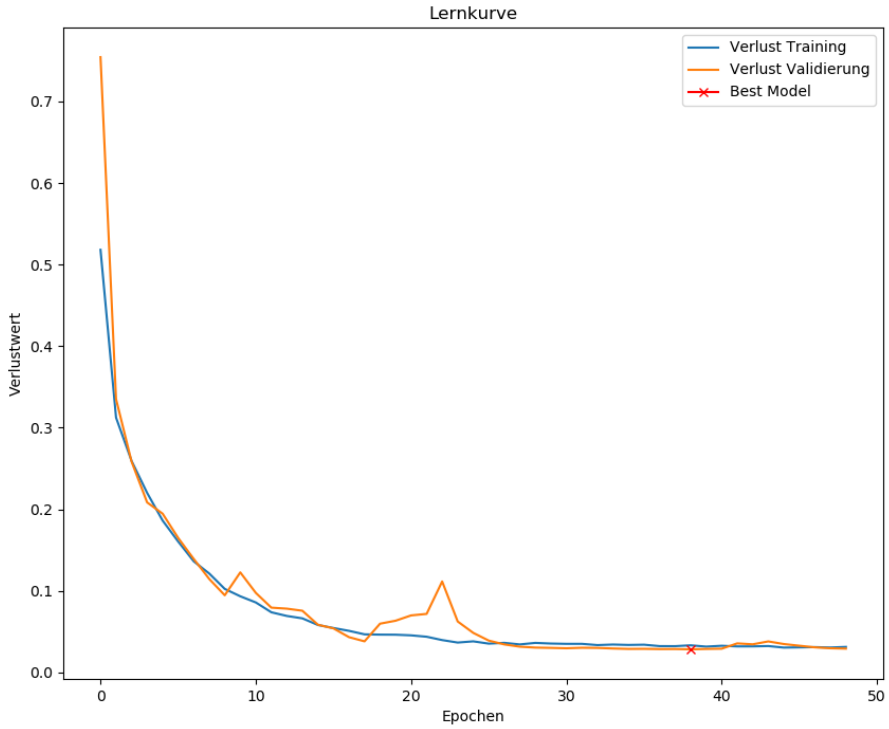


*Abbildung 26 - Verlustfunktionsverlauf „kies_pos_rgb_unet"*

### 4.3.7 Jupyter Notebook „kies_pos_unet"

In diesem Jupyter Notebook sind die in Tabelle 23 genannten Hyperparameter verwendet worden.

| Hyperparameter | Variable |
|---|---|
| network_name | kies_pos_unet |
| layer | 4 |
| model_name | kies_pos_unet.h5 |
| his_name | kies_pos_unet_history.p |

*Tabelle 23 - Hyperparameter Jupyter Notebook „kies_pos_unet"*

Für das Training sind die Trainingsdaten aus ‚r', ‚g', ‚b' und ‚nir' (siehe Datenprozessierung) verwendet worden. Insgesamt wurde das Model über 43 von 50 Epochen trainiert und das beste Model in Epoche 32 erzielt. Auf den Validierungsdaten wurde eine Klassifikationsgenauigkeit von ca. 99,7% erreicht. Abbildung 27 und Abbildung 28 zeigen den genauen Verlauf der Klassifikationsgenauigkeit und der Verlustfunktion. Basierend auf diesen Abbildungen ist die Gleichmäßigkeit des Lernprozesses auffällig. Das CNN scheint in der Lage zu sein, sehr schnell relevante Informationen zu extrahieren. Es findet keine extreme Überanpassung des Modells an die Trainingsdaten statt und die Lernkurve ist insgesamt kaum mit Ausreißern versehen.

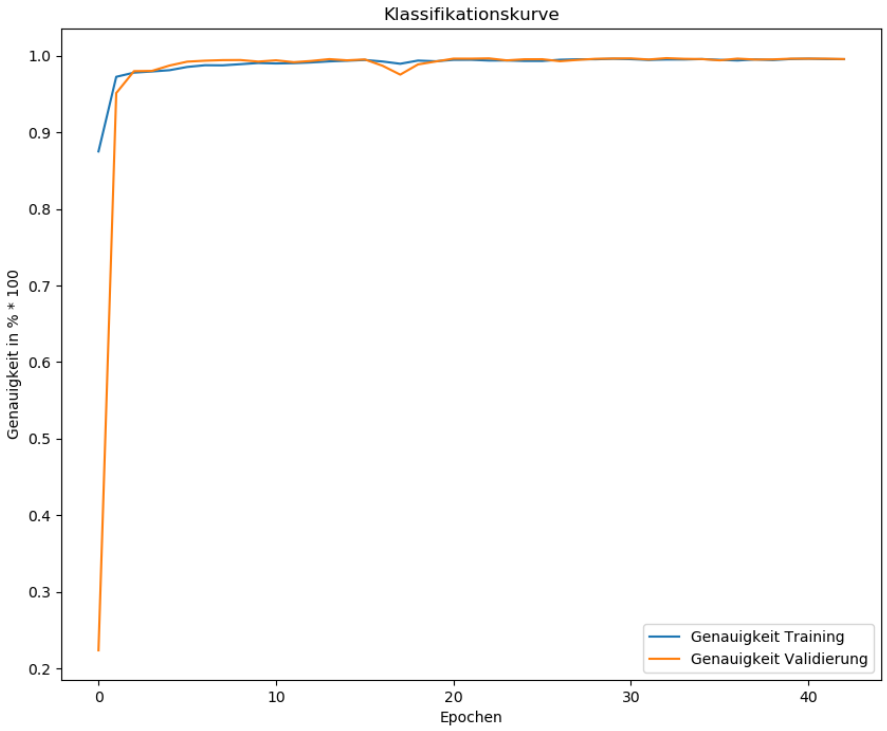


*Abbildung 27 - Klassifikationsgenauigkeitsverlauf „kies_pos_unet"*

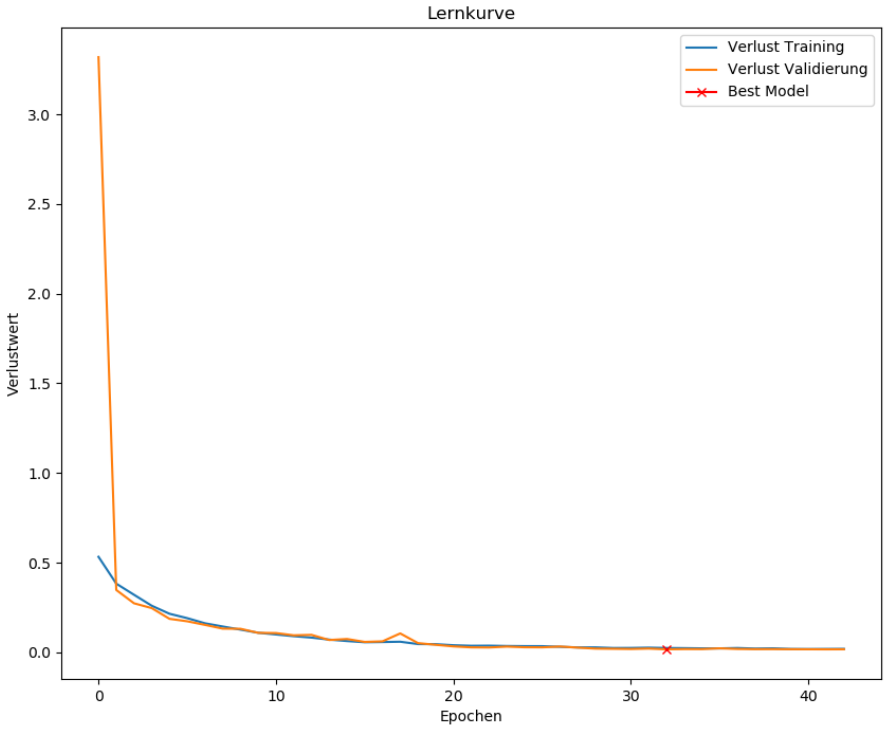


*Abbildung 28 - Verlustfunktionsverlauf „kies_pos_unet"*

### 4.3.8 Jupyter Notebook „kies_ndvi_unet”

In diesem Jupyter Notebook sind die in Tabelle 24 genannten Hyperparameter verwendet worden.

| Hyperparameter | Variable |
|---|---|
| network_name | kies_pos_ndvi_unet |
| layer | 5 |
| model_name | kies_pos_ndvi_unet.h5 |
| his_name | kies_pos_ndvi_unet_history.p |

*Tabelle 24 - Hyperparameter Jupyter Notebook „kies_pos_ndvi_unet"*

Für das Training sind die Trainingsdaten aus ‚r‘, ‚g‘, ‚b‘ und ‚nir‘ (siehe Datenprozessierung), sowie die NDVI-Werte errechnet und verwendet worden. Insgesamt wurde das Model über 47 von 50 Epochen trainiert und das beste Model in Epoche 36 erzielt. Auf den Validierungsdaten wurde eine Klassifikationsgenauigkeit von ca. 99,72% erreicht. Abbildung 29 und Abbildung 30 zeigen den genauen Verlauf der Klassifikationsgenauigkeit und der Verlustfunktion. Basierend auf diesen Abbildungen ist die fast kontinuierlich geringere Loss-Funktion auf den Validierungsdaten auffällig. Lediglich um Epoche 28 kommt es zu einer kurzfristigen und geringeren Überanpassung auf die Trainingsdaten. Diese wird in den darauffolgenden Epochen wieder ausgeglichen und es wird ziemlich direkt im Anschluss auch das Modell gefunden, welches den geringsten Loss-Wert aufweist.

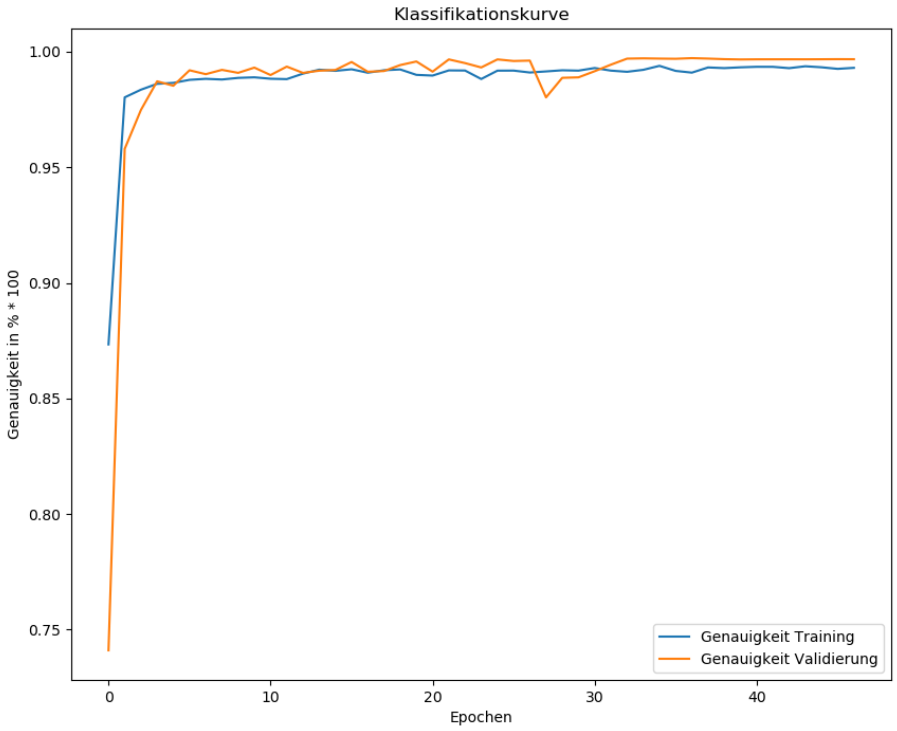


*Abbildung 29 - Klassifikationsgenauigkeitsverlauf „kies_pos_ndvi_unet"*

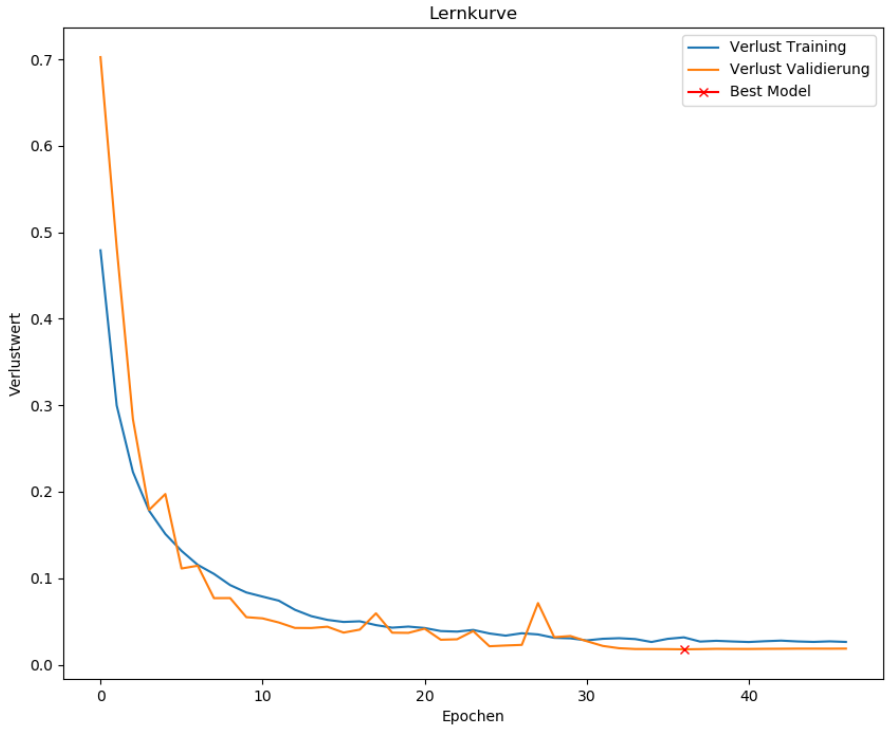


*Abbildung 30 - Verlustfunktionsverlauf „kies_pos_ndvi_unet"*

### 4.3.9 Jupyter Notebook „kies_ndwi_unet”

In diesem Jupyter Notebook sind die in Tabelle 25 genannten Hyperparameter verwendet worden.

| Hyperparameter | Variable |
|---|---|
| network_name | kies_pos_ndwi_unet |
| layer | 5 |
| model_name | kies_pos_ndwi_unet.h5 |
| his_name | kies_pos_ndwi_unet_history.p |

*Tabelle 25 - Hyperparameter Jupyter Notebook „kies_pos_ndwi_unet"*

Für das Training sind die Trainingsdaten aus ‚r‘, ‚g‘, ‚b‘ und ‚nir‘ (siehe Datenprozessierung), sowie die NDWI-Werte errechnet und verwendet worden. Insgesamt wurde das Model über 50 von 50 Epochen trainiert und das beste Model in Epoche 43 erzielt. Auf den Validierungsdaten wurde eine Klassifikationsgenauigkeit von ca. 99,46% erreicht. Abbildung 31 und Abbildung 32 zeigen den genauen Verlauf der Klassifikationsgenauigkeit und der Verlustfunktion. Basierend auf diesen Abbildungen ist die Gleichmäßigkeit des Lernprozesses auffällig. Das CNN scheint in der Lage zu sein, sehr schnell relevanten Informationen zu extrahieren. Es findet keine plötzliche Überanpassung des Modells auf die Trainingsdaten statt.

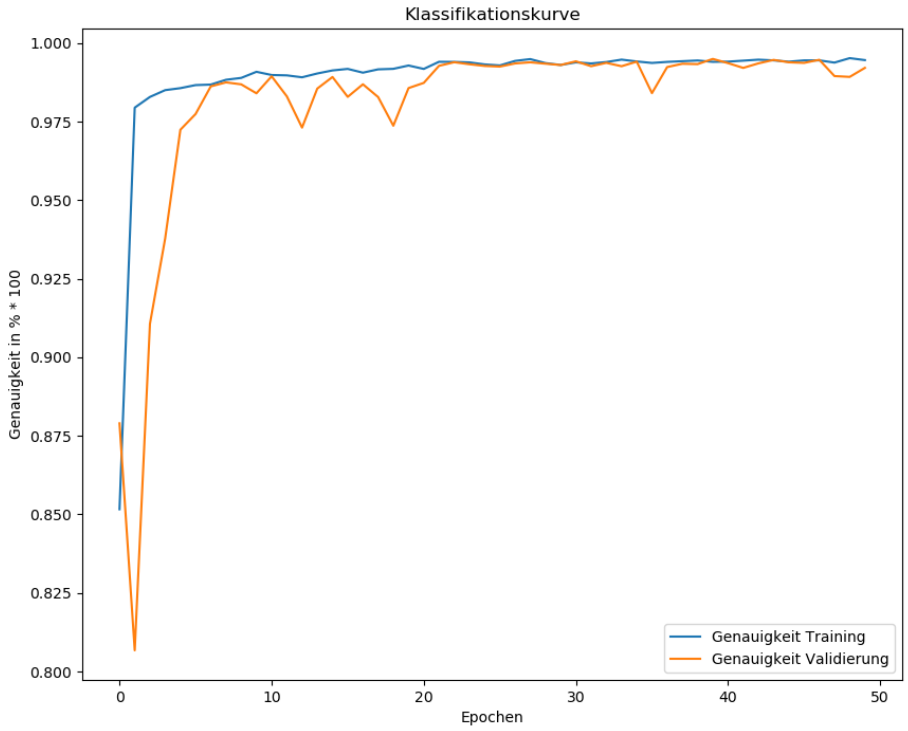


*Abbildung 31 - Klassifikationsgenauigkeitsverlauf „kies_pos_ndwi_unet"*

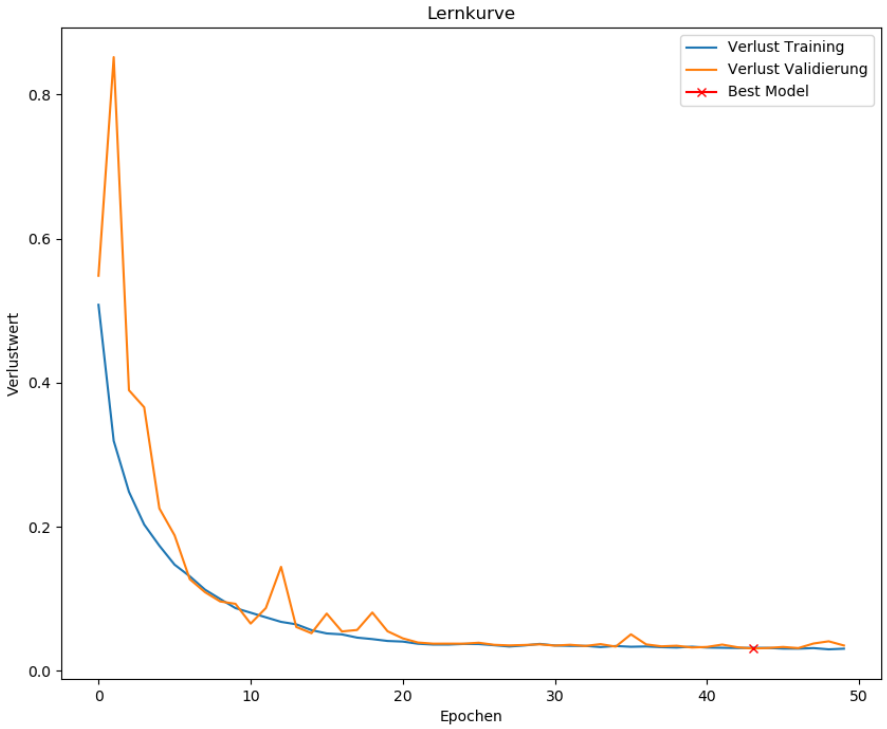


*Abbildung 32 - Verlustfunktionsverlauf „kies_pos_ndwi_unet"*

### 4.3.10 Jupyter Notebook „kies_ndbi_unet”

In diesem Jupyter Notebook sind die in Tabelle 26 genannten Hyperparameter verwendet worden.

| Hyperparameter | Variable |
|---|---|
| network_name | kies_pos_ndbi_unet |
| layer | 5 |
| model_name | kies_pos_ndbi_unet.h5 |
| his_name | kies_pos_ndbi_unet_history.p |

*Tabelle 26 - Hyperparameter Jupyter Notebook „kies_pos_ndbi_unet"*

Für das Training sind die Trainingsdaten aus ‚r‘, ‚g‘, ‚b‘ und ‚nir‘ (siehe Datenprozessierung), sowie die und NDBI-Werte errechnet und verwendet worden. Insgesamt wurde das Model über 50 von 50 Epochen trainiert und das beste Model in Epoche 40 erzielt. Auf den Validierungsdaten wurde eine Klassifikationsgenauigkeit von ca. 99,54% erreicht. Abbildung 33 und Abbildung 34 zeigen den genauen Verlauf der Klassifikationsgenauigkeit und der Verlustfunktion. Basierend auf diesen Abbildungen ist die Ungleichmäßigkeit des Lernprozesses auffällig. Der Loss-Wert der Validierungsdaten springt immer wieder über den der Trainingsdaten, um in den darauffolgenden Epochen wieder unter diesen zu fallen. Erst ab Epoche 25 scheint sich das Lernverhalten und die Gewichtung stabilisiert zu haben und das Modell lernt gleichmäßiger.

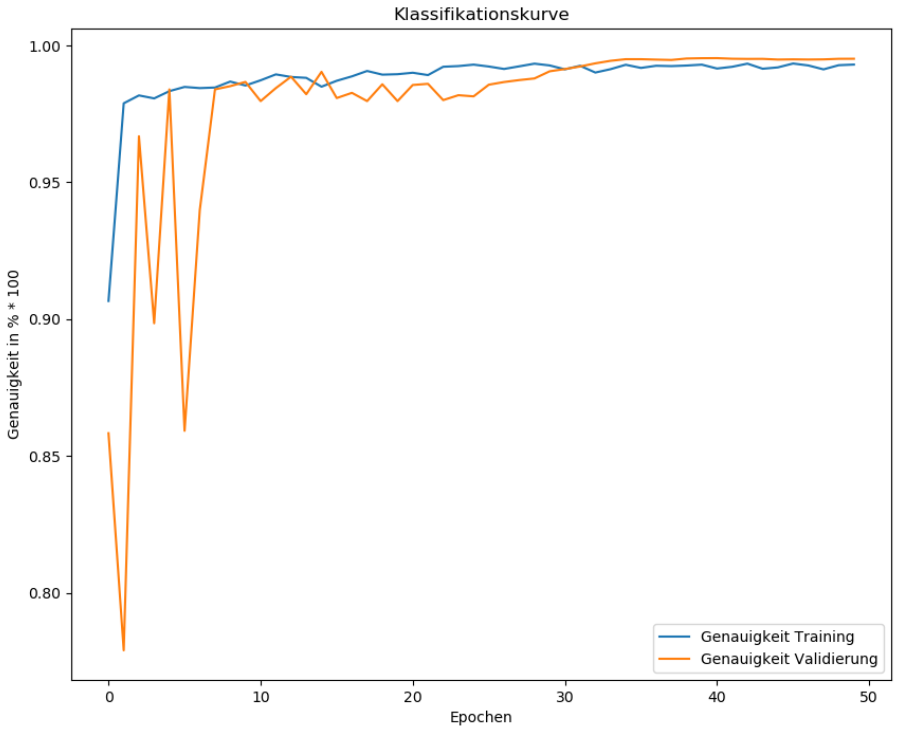


*Abbildung 33 - Klassifikationsgenauigkeitsverlauf „kies_pos_ndbi_unet"*

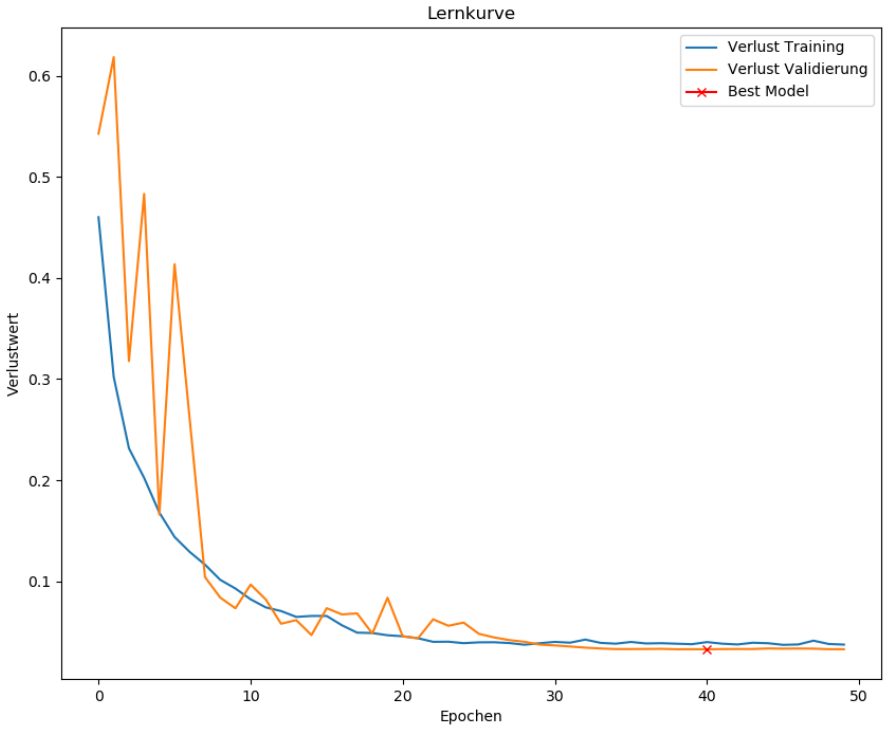


*Abbildung 34 - Verlustfunktionsverlauf „kies_pos_ndbi_unet"*

### 4.3.11 Jupyter Notebook „kies_ndvwi_unet”

In diesem Jupyter Notebook sind die in Tabelle 27 genannten Hyperparameter verwendet worden.

| Hyperparameter | Variable |
|---|---|
| network_name | kies_pos_ndvwi_unet |
| layer | 6 |
| model_name | kies_pos_ndvwi_unet.h5 |
| his_name | kies_pos_ndvwi_unet_history.p |

*Tabelle 27 - Hyperparameter Jupyter Notebook „kies_pos_ndvwi_unet"*

Für das Training sind die Trainingsdaten aus ‚r‘, ‚g‘, ‚b‘ und ‚nir‘ (siehe Datenprozessierung), sowie die NDVI- und NDWI-Werte errechnet und verwendet worden. Insgesamt wurde das Model über 50 von 50 Epochen trainiert und das beste Model in Epoche 39 erzielt. Auf den Validierungsdaten wurde eine Klassifikationsgenauigkeit von ca. 99,59% erreicht. Abbildung 35 und Abbildung 36 zeigen den genauen Verlauf der Klassifikationsgenauigkeit und der Verlustfunktion. Basierend auf diesen Abbildungen ist die Ungleichmäßigkeit des Lernprozesses auffällig. Der Loss-Wert der Validierungsdaten springt immer wieder über den der Trainingsdaten, um in den darauffolgenden Epochen wieder unter diesen zu fallen. Erst ab Epoche 39 scheint sich das Lernverhalten und die Gewichtung stabilisiert zu haben und das Modell lernt gleichmäßiger.

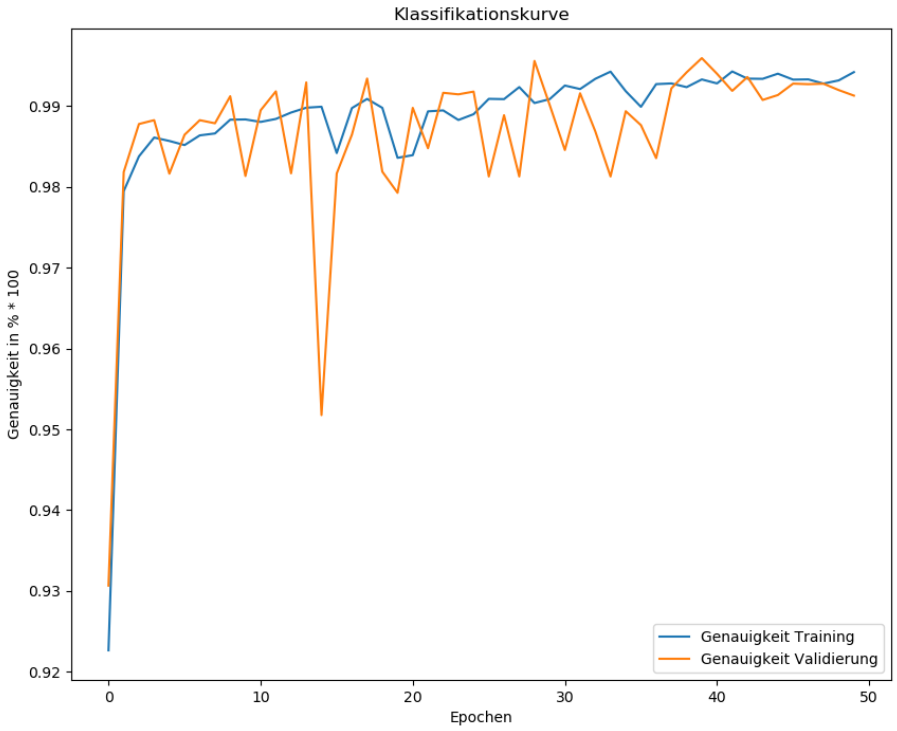


*Abbildung 35 - Klassifikationsgenauigkeitsverlauf „kies_pos_ndvwi_unet"*

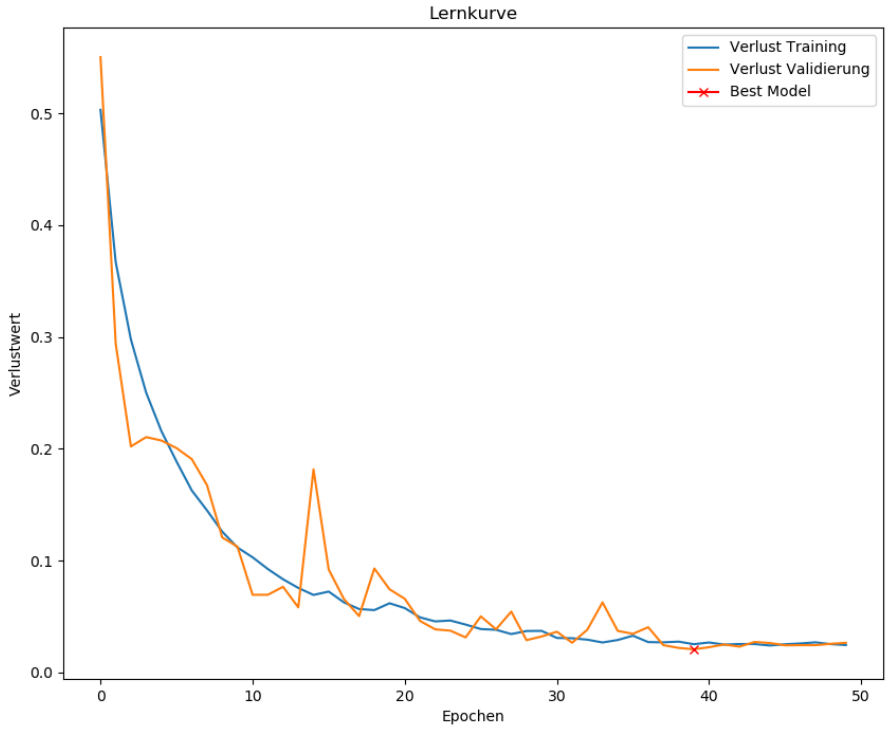


*Abbildung 36 - Verlustfunktionsverlauf „kies_pos_ndvwi_unet"*

# 5 Bewertung von Klassifikationsergebnissen

In diesem Kapitel wird das Modell mit der größten Validierungsgenauigkeit, „kies_pos_ndvi_unet“ (Abschnitt 4.3.8), bewertet. Die anderen Modelle werden nicht bewertet, da diese auf den Validierungsdaten schlechtere Ergebnisse bezüglich der Validierungsgenauigkeit haben und zu erwarten ist, dass deren Klassifikationsergebnisse dementsprechend schlechter sein werden.

Für die Bewertung werden zunächst zwei unterschiedliche Sentinel 2-Szenen vom 24.07.2019 und 25.07.2019 klassifiziert. Diese Szenen wurden gewählt, da diese zum einen sehr aktuell sind und zum anderen eine geringe Wolkenbedeckung aufweisen. Die erste Szene stammt aus dem Raum Düsseldorf und enthält einige Trainingsgebiete. Die zweite Szene stammt aus dem Raum Köln/Bonn und enthält keine Trainingsgebiete. Zunächst wird bewertet, wie viele der auf den Szenen als Kiesabbaufläche gekennzeichneten Gebiete auch wirklich Kiesabbauflächen sind. Anschließend wird darauf eingegangen wie genau eine Kiesabbaufläche bezüglich ihrer Form und Größe erkannt wird.

Bei der Klassifizierung durch das oben genannte Modell werden in den erzeugten Heatmaps (Kapitel 4.3.1) zunächst alle Pixel mit einem Wert von weniger als 0,5 nicht betrachtet und verworfen. Pixel, die einen Wert größer/gleich 0,5 besitzen werden behalten, basierend auf räumlicher Nähe zu Gebieten zusammengefasst und überprüft. Diese Überprüfung wird visuell mittels der in Google Maps, Bing Maps und Openstreetmap hinterlegten Daten durchgeführt.

Nach dem Filtern und zusammenfassen der Pixel zu Gebieten sind für die Szene aus dem Raum Düsseldorf insgesamt 11 Gebiete und aus dem Raum Köln/Bonn 41 Gebiete überprüft worden, ob es sich bei dem Gebiet auch um eine Kiesabbaufläche handelt. Die Ergebnisse der Überprüfung sind in Tabelle 28 - Tabelle 30 beschrieben. Schwach, Mittel und Stark beschreiben dabei die überwiegend vorhandenen Pixelwerte in einem Gebiet. Schwach sind Pixel von 0,5-<0,7, Mittel sind Pixel von 0,7-<0,9 und Stark sind Pixel >0,9.

| Düsseldorf | | Pixelwerte | | |
|---|---|---|---|---|
| | | Schwach | Mittel | Stark |
| Abbaugebiet | Ja | 2 | 6 | 1 |
| | Nein | 1 | 0 | 1 |

*Tabelle 28 - Klassifizierungsergebnis Düsseldorf*

| Düsseldorf | | Pixelwerte | | |
|---|---|---|---|---|
| | | Schwach | Mittel | Stark |
| Trainingsgebiet | Ja | 0 | 4 | 0 |
| | Nein | 3 | 2 | 2 |

*Tabelle 29 - Klassifizierungsergebnis Düsseldorf bezüglich der Trainingsgebiete*

| Köln/Bonn | | Pixelwerte | | |
|---|---|---|---|---|
| | | Schwach | Mittel | Stark |
| Abbaugebiet | Ja | 4 | 4 | 7 |
| | Nein | 20 | 2 | 4 |

*Tabelle 30 - Klassifizierungsergebnis Köln/Bonn*

Insgesamt sind somit lediglich 46% der als Kiesfläche klassifizierten Gebiete auch wirklich aktuelle Kiesabbaugebiete. Auffallend ist, dass insbesondere in Köln/Bonn viele Gebiete Fehlklassifiziert wurden, die überwiegend schwache Pixelwerte aufweisen. Dieses Problem kann man dadurch lösen, indem der Schwellwert, zum Verwerfen von Pixeln, erhöht wird. Dadurch würden jedoch auch einige Flächen verworfen werden, die nur einen schwachen Wert in der Klassifikation aufweisen, jedoch zu Kiesabbauflächen gehören. Wendet man dies an erhöht sich die Rate der korrekt klassifizierten Flächen auf 72%.

Als nächstes wird bewertet, wie genau eine Kiesfläche bezüglich ihrer Form und Ausdehnung aus den Satellitenbildern extrahiert werden kann. Bei der Betrachtung von den als Kiesflächen ausgegeben Gebieten ist auffällig, dass in einigen Fällen die Form des Abbaugebietes sehr gut extrahiert wird, bei anderen genau das Gegenteil der Fall ist. Hier werden nur einzelne Bereiche der Kiesabbaufläche als solche Ausgegeben. In Abbildung 37-Abbildung 39 wird das Problem nochmals verdeutlicht.

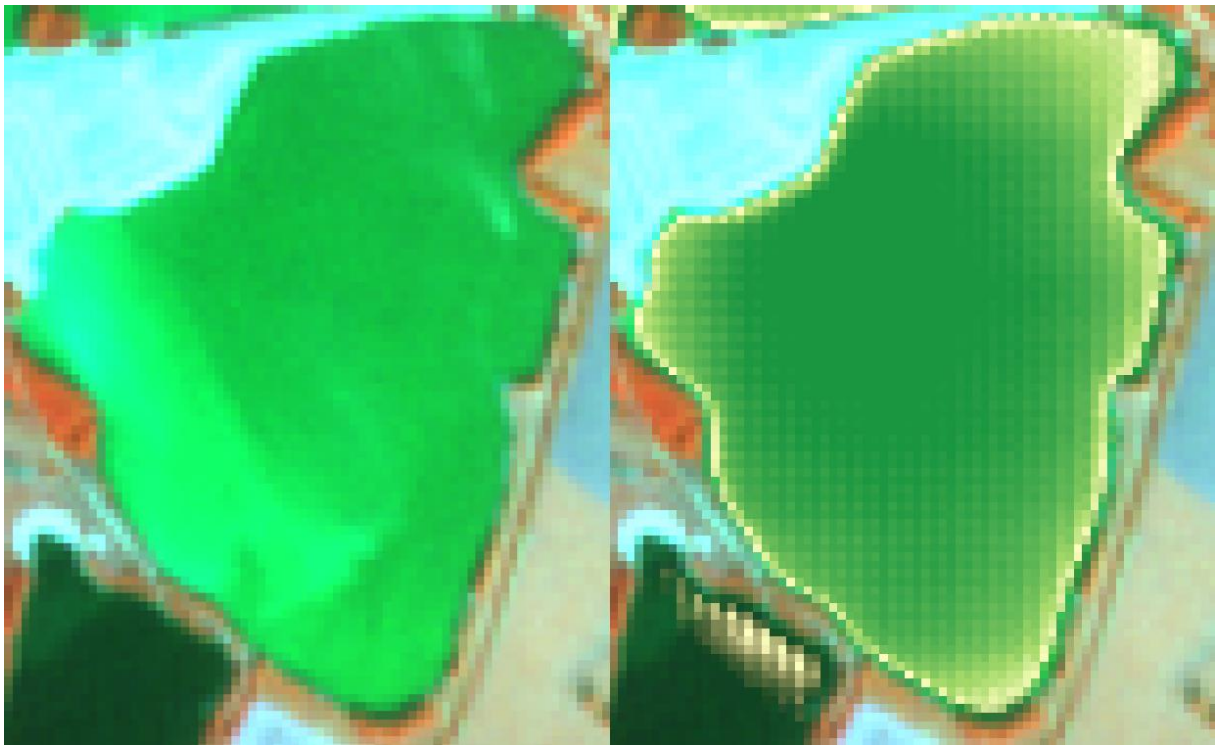

*Abbildung 37 - Vollständige Segmentierung der Kiesfläche*

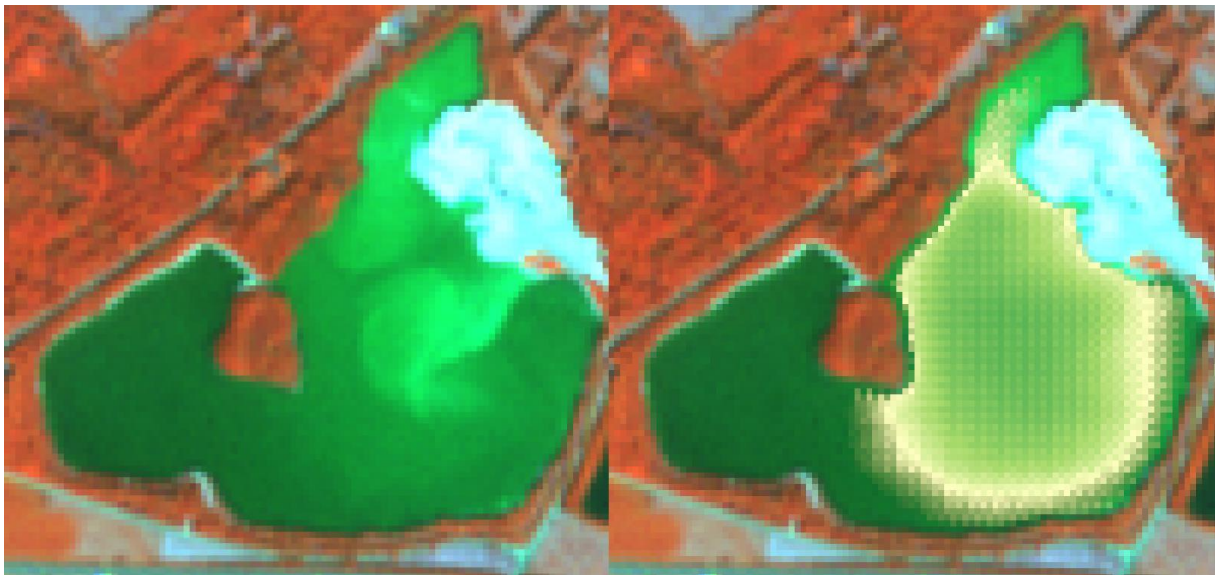

*Abbildung 38 - Unvollständige Segmentierung der Kiesfläche*

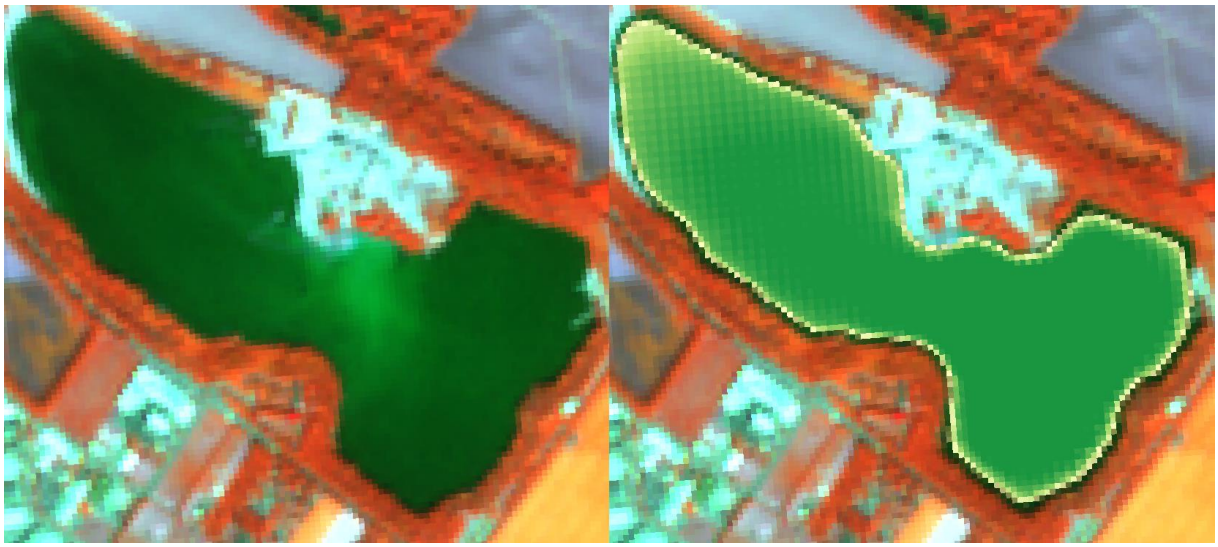

*Abbildung 39 - Vollständige Segmentierung der Kiesfläche*

Bei der visuellen Interpretation fällt auf, dass Pixel die eine sehr hohe Wahrscheinlichkeit besitzen (auf der rechten Seite in Abbildung 37 - Abbildung 39 durch dunklere Grüntöne dargestellt) zu einer Kiesabbaufläche zu gehören, in der korrespondierenden linken Hälfte der Abbildung tendenziell eher einen helleren Grünton aufweisen als Pixel mit einem geringeren Wahrscheinlichkeitswert. Die in Abbildung 37-Abbildung 39 ausgeschnittenen Bereiche der Gesamtszene sind in der Kanalkombination NIR, G, B als RGB-Bild dargestellt. Dies legt nahe, dass das CNN Pixel mehr anhand ihrer spektralen Eigenschaft klassifiziert, als anhand räumlicher Nähe zu Nachbarpixeln.

Bewertend ist abschließend festzustellen, dass es in diesem Fall möglich ist Kiesflächen, mit einer gewissen Fehlklassifikationsrate, mittels CNN zu detektieren. Gleichzeitig ist es aber nicht gelungen Masken zu erstellen, aus denen die aktuelle Form und Größe der Kiesfläche extrahiert werden kann.

# 6 Fazit

In dieser Arbeit wurde gezeigt, dass es möglich ist Kiesflächen mittels CNNs in Sentinel-2 Satellitenbildern zu erkennen. Dabei ist allerdings auch festgestellt worden, dass die Erkennung und insbesondere Extraktion noch nicht einwandfrei funktioniert (Kapitel 5). Insbesondere bei einer Inhomogenität in den spektralen Informationen innerhalb der Kiesfläche fällt es dem Modell, schwer diese komplett als Kiesfläche zu erkennen. Im Umfang dieser Arbeit konnten die in Kapitel 5 erläuterten Probleme bei der Klassifikation nicht weiter untersucht werden.

In Zukunft wäre es eine Möglichkeit das CNN mit mehr Daten zu trainieren, um dem Modell eine bessere Definition von Kiesabbauflächen beizubringen. Eine weitere Option wäre es, weitere Kanalkombinationen, die noch nicht angewendet worden sind, auszuprobieren und Kanäle der Sentinel-2 Satelliten, die für das aktuelle Modell noch nicht verwendet wurden, als zusätzlichen Input hinzuzunehmen. Zudem wäre es ein weiterer Weg, um dem Modell Eigenschaften von Kiesabbauflächen beizubringen, Aufnahmen der Sentinel-1 Satelliten mit zu verwenden. Im Gegensatz zu Sentinel-2 ist Sentinel-1 ein Radar-Satellitensystem und nimmt dementsprechend die Erdoberfläche in einem anderen Frequenzbereich des elektromagnetischen Spektrums auf. Damit wäre es möglich Informationen zu den charakteristischen Eigenschaften von Kiesabbauflächen zu erhalten, die von Sentinel-2 nicht aufgezeichnet werden können. Neben den aufgezählten Wegen das Modell über zusätzliche spektrale Informationen zu verbessern, kann auch über die Einstellung von Trainingsparametern das CNN genauer für die Aufgabe der Kiesflächendetektion abgestimmt werden. Dies könnte Beispielsweise sein, dass die Größe des Kernels angepasst wird. Mit einem größeren Kernel könnte Beispielsweise erreicht werden, dass großflächigere Eigenschaften besser erkannt werden, während ein kleinerer Kernel bewirken kann, das kleinflächige Beziehungen zwischen einzelnen Pixeln deutlicher zum Vorschein treten. Auch wäre es hier eine Option mehr Kernel zu verwenden um weitere Eigenschaften zu extrahieren, die bei der aktuellen Zahl an Kerneln noch nicht detektiert worden sind. Als alternativen Ansatz, zum Detektieren von Kiesflächen, kann versucht werden nicht die Kiesfläche als Ganzes zu erkennen, sondern nur den Uferbereich zu klassifizieren, wodurch ebenfalls die Form und Größe der zu detektierenden Kiesfläche erkannt werden kann.

Vollends ist festgestellt worden, dass dieses Thema der Kiesflächendetektion mittels CNNs sehr komplex ist. In Zukunft bestehen viele Möglichkeiten das Thema weiter zu verfolgen und zu bearbeiten um die Modelle zu verbessern. Einige dieser Techniken zum Verbessern der Klassifikationsgenauigkeit wurden bereits im vorherigen Absatz erwähnt.

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Literaturverzeichnis


Andelfinger, V. P., & Hänisch, T. (2015). *Internet der Dinge: Technik, Trends und Geschäftsmodelle.* Springer Gabler. doi:10.1007/978-3-658-06729-8

Awad, M., & Khanna, R. (2015). Support Vector Machines for Classification. In *Efficient Learning Machines: Theories, Concepts, and Applications for Engineers and System Designers* (S. 39-66). Apress.

Bacher, J., Pöge, A., & Wenzig, K. (2011). *Clusteranalyse: Anwendungsorientierte Einführung in Klassifikationsverfahren.* De Gruyter.

Brownlee, J. (12. August 2019). *Machine Learn Mastery: Gradient Descent For Machine Learning*. Abgerufen am 30. September 2019 von https://machinelearningmastery.com/gradient-descent-for-machine-learning/

Butzmann, L. (März 2019). Ausreißerdetektion in Temperaturlinien auf der Grundlage von Machine Learning Algorithmen.

Chollet, F. (2018). *Deep Learning mit Python und Keras.* mitp.

*ESA EO Missions: Sentinel-2*. (kein Datum). Abgerufen am 01. Oktober 2019 von https://earth.esa.int/web/guest/missions/esa-operational-eo-missions/sentinel-2

Euen, B. (2015). *Intelligenz und kognitive Kompetenzen.* Books on Demand.

Frochte, J. (2018). Support Vector Machines. In *Machinelles Lernen* (S. 286-303). Carl Hanser Verlag GmbH & Co. KG.

Gao, B.-C. (01. 12 1996). NDWI - A normalized difference water index for remote sensing of vegetation liquid water from space. *Remote Sensing of Environment*(58), S. 257-266. doi:10.1016/S0034-4257(96)00067-3

Godfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning.* MIT Press. Abgerufen am 30. September 2019 von http://www.deeplearningbook.org

Hale, J. (20. September 2018). *Towards Data Science: Deep Learning Framework Power Scores 2018*. Abgerufen am 30. September 2019 von https://towardsdatascience.com/deep-learning-framework-power-scores-2018-23607ddf297a

Hertzberg, J. (2018). Einführung in die KI.

Hudson, H. C. (1982). *Classifying social data: new applications of analytical methods for social science research.* Jossey-Bass.

IEEE 802.3 Ethernet Working Group. (2012). *IEEE 802.3 Ethernet Working Group Communication.* Abgerufen am 01. Oktober 2019 von http://www.ieee802.org/3/ad_hoc/bwa/BWA_Report.pdf

Ioffe, S., & Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. Abgerufen am 30. September 2019 von https://arxiv.org/abs/1502.03167

*Keras.* (kein Datum). Abgerufen am 30. September 2019 von https://keras.io/why-use-keras/

Kingma, D. P., & Ba, J. (2014). Adam: A Method for Stochastic Optimization. Abgerufen am 30. September 2019 von https://arxiv.org/abs/1412.6980v8

Lin, Y., Lv, F., Zhu, S., Yang, M., Cour, T., Yu, K., . . . Huang, T. (2011). Large-scale image classification. *CVPR 2011*, S. 1689-1696. doi:10.1109/CVPR.2011.5995477

Marthi, S. (23. Juli 2018). Landuse Classification from Satellite Imagery using Deep Learning. Abgerufen am 07. Oktober 2019 von https://www.youtube.com/watch?v=QbgIwOq7ksQ&t=

Nguyen, C. N., & Zeigermann, O. (2018). *Machine Learning - kurz & gut: Eine Einführung mit Python, Pandas und Scikit-Learn.* O'Reilly.

Raschka, S., & Mirjalili, V. (2018). *Machine Learning mit Python und Scikit-learn und TensorFlow.* mitp.

Ronneberger, O., Fischer, P., & Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. (N. Navab, J. Hornegger, W. M. Wells, & A. F. Frangi, Hrsg.) *Medical Image Computing and Computer-Assisted Intervention - MICCAI 2015*, S. 234-241. doi:10.1007/978-3-319-24574-4_28

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (01. Oktober 1986). Learning representations by back-propagating errors. *Nature*(323), S. 533-536. doi:10.1038/323533a0

Russell, S., & Norvig, P. (2009). *Artificial Intelligence: A Modern Approach.* Pearson.

Sarle, W. (2002). *Neural Network FAQ*. Abgerufen am 30. September 2019 von ftp://ftp.sas.com/pub/neural/FAQ.html#A_kinds

Schlegel, D. (2015). *Deep Machine Learning on GPUs.* Abgerufen am 07. Oktober 2019 von http://www.ziti.uni-heidelberg.de/ziti/uploads/ce_group/seminar/2014-Daniel_Schlegel.pdf

Simon, H. (Januar 2011). *University of British Columbia, Department of Computer Science.* Abgerufen am 30. September 2019 von https://www.cs.ubc.ca/~arnaud/cs340/lec1_CS340_handouts.pdf

Simonyan, K., & Zisserman, A. (September 2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv 1409.1556*. Abgerufen am 02. Oktober 2019 von https://arxiv.org/abs/1409.1556

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*(15), S. 1929-1958. Abgerufen am 30. September 2019

Surmenok, P. (13. November 2017). *Towards Data Science: Estimating an Optimal Learning Rate For a Deep Neural Network*. Abgerufen am 30. September 2019 von https://towardsdatascience.com/estimating-optimal-learning-rate-for-a-deep-neural-network-ce32f2556ce0

Weston, J., Mukherjee, S., Chapelle, O., Pontil, M., Poggio, T., & Vapnik, V. (April 2001). Feature Selection for SVMs. *Advances in Neural Information Processing Systems*, S. 668-674. Abgerufen am 2019. Oktober 2019 von https://papers.nips.cc/paper/1850-feature-selection-for-svms.pdf

# Versionsverzeichnis

## Verwendete Bibliotheken in der Datenvorverarbeitung

| Bibliothek | Version |
|---|---|
| Conda | 4.7.5 |
| Gdal | 3.0.0 |
| Geojson | 2.4.1 |
| Ipykernel | 5.1.1 |
| Ipython | 7.6.0 |
| Matplotlib | 3.1.1 |
| Numpy | 1.16.4 |
| Pip | 19.1.1 |
| Python | 3.7.3 |
| Sentinelsat | 0.13 |

## Verwendete Bibliotheken für das CNN

| Bibliothek | Version |
|---|---|
| Conda | 4.7.10 |
| Gdal | 3.0.1 |
| Imgaug | 0.2.9 |
| Ipykernel | 5.1.2 |
| Ipython | 7.8.0 |
| Keras-gpu | 2.2.4 |
| Matplotlib | 3.1.1 |
| Numpy | 1.16.4 |
| Pip | 19.2.2 |
| Python | 3.7.4 |
| Scikit-learn | 0.21.2 |
| Tqdm | 4.36.1 |

# Hardwareverzeichnis

## Lokal

Prozessor: Intel Core i7-6820HQ CPU @ 2.70 GHz

RAM: 16,0 GB

Grafikkarte: NVIDIA GeForce 940MX, 2010 MB VRAM

## VM

Prozessor: Intel Xeon CPU E5-2690 v3 @ 2.60 GHz

RAM: 56,0 GB

Grafikkarte: NVIDIA Tesla K80, 11448 MB VRAM

# Anhang

Der Bachelorarbeit ist ein USB-Stick beigefügt. Auf diesem Stick befindet sich die Bachelorarbeit in digitaler Form, sowie die Zip-Datei ‚Data'. In Data befindet sich der Code der entwickelt wurde, die verwendeten Daten zum Nachvollziehen und Nachbauen der beschriebenen Prozesse und die erzeugten Ergebnisse. In dem Readme ist beschrieben, welche Dateien sich wo befinden.

# Erklärung zur selbstständigen Abfassung der Bachelorarbeit

Ich versichere, dass ich die eingereichte Bachelorarbeit[3] selbstständig und ohne unerlaubte Hilfe verfasst habe. Anderer als der von mir angegebenen Hilfsmittel und Schriften habe ich mich nicht bedient. Alle wörtlich oder sinngemäß den Schriften anderer Autoren entnommenen Stellen habe ich kenntlich gemacht.

---

Ort, Datum, Unterschrift

[3] Bei einer Gruppenarbeit gilt o. für den entsprechend gekennzeichneten Anteil der Arbeit