universität wien

MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

Landcover-Klassifikation von hochaufgelösten Orthofotos mithilfe von Convolutional Neural Networks und TensorFlow

verfasst von / submitted by

Felix Pekárek, BSc

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien, 2020 / Vienna, 2020

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	A 066 856
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Kartographie und Geoinformation
Betreuet von / Supervisor:	Ass.-Prof. Mag. Dr. Andreas Riedl

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abstract

The derivation of land cover information is an important method in remote sensing. Usually orthophotos or satellite images are used as data sources. Various methods are available for the land cover classification process, some of which are more complex than others. One of the most current methods is the use of so-called Convolutional Neural Networks. This is a special artificial intelligence that is explicitly suited for the classification of images.

The master's thesis is being developed in cooperation with the Federal Office of Metrology and Surveying, which is already conducting their own landcover classification. This will now be extended by a Convolutional Neural Network. This should result in a more detailed classification with additional landcover classes. Orthophotos and terrain models provided by the Federal Office of Metrology and Surveying serve as data sources. First of all, training data had to be created from these datasets, which will be used for the training of the Convolutional Neural Network. Several variants were trained, whereby different image channels were used for training. The trained networks were then used for the classification of a total of four orthophotos in different areas of Austria. These were then checked and evaluated for their accuracy. Furthermore, the results were compared to the already existing land cover classification in the Federal Office of Metrology and Surveying. The Convolutional Neural Network was created using the framework TensorFlow in Python and the subsequent analysis was carried out using a geoinformation system.

In this way it could be shown that the use of Convolutional Neural Networks is quite reasonable. The existing classes could be refined or extended in most cases. The artificial intelligence produced more accurate results than the classification of the Federal Office of Metrology and Surveying. In another case, however, the classification of the Convolutional Neural Network was much less accurate due to a lack of training in areas with dense population.

Kurzfassung

Die Ableitung von Bodenbedeckungsinformationen ist eine wichtige Methode in der Fernerkundung. In der Regel werden dazu Orthofotos oder Satellitenbilder verwendet. Für den Prozess der Klassifikation stehen verschiedene Methoden zur Verfügung, von denen manche komplexer als andere sind. Zu den aktuellsten Methoden zählt der Einsatz von Convolutional Neural Networks. Dabei handelt es sich um eine spezielle künstliche Intelligenz, die explizit für die Klassifikation von Bildern geeignet ist.

Die Masterarbeit entsteht in Kooperation mit dem Bundesamt für Eich- und Vermessungswesen, die bereits eine eigene Landcover-Klassifikation durchführen. Diese soll nun durch ein Convolutional Neural Network erweitert werden. Dadurch soll eine detailreichere Klassifikation mit zusätzlichen Landcover-Klassen entstehen. Als Datenquelle dienen hochaufgelöste Orthofotos und Geländemodelle, die vom Bundesamt für Eich- und Vermessungswesen bereitgestellt wurden. Aus diesen Daten mussten zunächst Trainingsdaten erstellt werden, welche anschließend für das Training des Convolutional Neural Networks verwendet wurden. Es wurden dabei mehrere Varianten trainiert, wobei verschiedene Bildkanäle zum Training eingesetzt wurden. Die trainierten Netzwerke wurden dann für die Klassifikation von insgesamt vier Orthofotos in unterschiedlichen Gebieten Österreichs verwendet. Diese wurden anschließend auf deren Genauigkeit überprüft und bewertet. Zudem wurden die Ergebnisse der bereits bestehenden Landcover-Klassifikation vom Bundesamt für Eich- und Vermessungswesen gegenübergestellt. Die Erstellung des Convolutional Neural Networks ist mithilfe des Frameworks TensorFlow in Python durchgeführt worden und die anschließende Überprüfung der Ergebnisse mithilfe eines Geoinformationssystems.

Auf diese Weise konnte gezeigt werden, dass der Einsatz von Convolutional Neural Networks durchaus sinnvoll ist. Die bestehenden Klassen konnten in den meisten Fällen verfeinert oder erweitert werden. Dabei hat die künstliche Intelligenz genauere Ergebnisse als die Klassifikation des Bundesamtes für Eich- und Vermessungswesen erzeugt. In einem anderen Fall hingegen, war die Klassifikation des Convolutional Neural Networks wesentlich ungenauer, was auf ein fehlendes Training in Gebieten mit einer dichten Besiedlung zurückzuführen war.

1. Einleitung

Die Ableitung von Bodenbedeckungsinformationen (Landcover-Klassifikation) zählt zu den wichtigsten Auswertungsprodukten, die mithilfe von Fernerkundungsdaten erstellt werden können. Das namhafteste Projekt zur Ableitung der Bodenbedeckung ist das EU-weite CORINE-Programm. Dabei werden digitale Satellitenbilder zunächst erfasst sowie vorprozessiert und im Anschluss werden die Aufnahmen in Bezug auf die Bodenbedeckung ausgewertet. Die daraus gewonnenen Bodenbedeckungsinformationen sind frei verfügbar und können im Anschluss für weiterführende Interpretationen oder Analysen verwendet werden (vgl. FERANEC J. et al. 2016: 3, Umweltbundesamt o. J.).

Auch das Bundesamt für Eich- und Vermessungswesen (BEV) macht eine solche Bodenbedeckungsanalyse für das gesamte österreichische Bundesgebiet. Dazu werden jedoch keine Satellitenbilder verwendet, sondern hochaufgelöste Orthofotos und Geländemodelle, die sich im Eigentum des BEV befinden. Der derzeitige Klassifizierungsalgorithmus ist seit 2016 im Einsatz. Derzeit können damit sechs Bodennutzungskategorien (Gebäude, Bodenflächen, Baum/Wald, Buschwerk, Niedrigvegetation und Gewässer) zuverlässig abgeleitet werden (vgl. BEV 2017: 75−79).

Die Ergebnisse dieses Algorithmus fließen derzeit in den Objektbereich ‚Bodenbedeckung' des digitalen Landschaftsmodells ein. Dieses beinhaltet alle natürlichen und anthropogenen Bestandteile der Landschaft in digitaler Form (Vektorformat). In Zukunft soll aus diesen Daten zudem das kartografische Modell des BEV abgeleitet werden. Das Ziel des BEV ist es nun, dass die bestehenden Bodennutzungsklassen um zusätzliche Klassen erweitert werden sollen. Das kartografische Modell soll ebenso von den zusätzlichen Klassen profitieren und dadurch detailreicher und genauer werden.

Das angestrebte Ergebnis von zusätzlichen Subklassen, zum Beispiel durch die Unterscheidung von Nadel- und Laubwald, soll mithilfe eines Convolutional Neural Networks (CNNs) erreicht werden. Dabei handelt es sich um eine bestimmte Art von künstlicher Intelligenz, die insbesondere für die maschinelle Verarbeitung von Bild- und Audiodaten entwickelt worden ist. Im BEV gibt es bisher keinerlei Erfahrungen mit dem Einsatz von CNNs.

Das CNN soll mithilfe des Frameworks *TensorFlow* von Google programmiert werden. *TensorFlow* kann als Gerüst für eine künstliche Intelligenz gesehen werden und unterstützt eine Vielzahl von verschiedenen Methoden im Bereich des maschinellen Lernens. Der Vorteil von *TensorFlow* liegt darin, dass hier ein kompletter Workflow vom Vorprozessieren der Daten (Preprocessing) bis zur finalen Vorhersage (Prediction) erstellt werden kann. *TensorFlow* wird in den meisten Fällen in der Programmiersprache Python geschrieben, es kann jedoch auch in anderen Sprachen wie C++ oder Java geschrieben werden.

Bevor ein CNN-Modell zur Klassifizierung von Bildern verwendet werden kann, muss dieses mithilfe von ausgewählten Beispielen, den sogenannten Trainingsdaten, trainiert werden. Im Fall der Ableitung von Bodenbedeckungsinformationen müssen Bilddaten als

Trainingsdaten verwendet werden. Hierzu müssen zunächst Orthofotos manuell klassifiziert und segmentiert werden (z. B. Gebäudeumrisse auf DKM-Basis) und diese Daten können anschließend zum Trainieren der künstlichen Intelligenz benutzt werden. Damit die Ergebnisse der Klassifikation lagerichtig und genau sind, muss ein passendes Modell gefunden werden und zudem müssen die dazugehörigen Parameter definiert werden.

Die Verwendung und Entwicklung von CNNs ist ein aktuelles Forschungsgebiet, das insbesondere in den letzten Jahren an Relevanz gewonnen hat. Der Grund für die gestiegene Bedeutung ist, dass die Hardware erst seit ungefähr zehn Jahren eine ausreichend große Rechenkraft besitzt. Insbesondere der Einsatz von Grafikprozessoren (GPU) hat zu einer gesteigerten Rechenleistung geführt und ermöglicht eine bis zu 20-mal schnellere Berechnung als auf der zentralen Recheneinheit (CPU) (vgl. KRIZHEVSKY et al. 2017).

Die Klassifizierung und Segmentierung von hochaufgelösten Luft- und Satellitenbildern, also eine Landcover-Klassifizierung, mithilfe von CNNs ist ein relativ neues Fachgebiet in der Fernerkundung, das jedoch zunehmend an Bedeutung gewinnt. So haben sich beispielsweise SHENDRYK et al. (2019) und GUO et al. (2018) mit dieser Disziplin beschäftigt. Beide Forscher haben sich dabei auf ein relativ kleines räumliches Gebiet beschränkt. Jedoch konnten sie mithilfe von CNNs gleichwertige bis bessere Ergebnisse erzielen.

Ein anderer wissenschaftlicher Bereich, der sich mit der Bilderkennung und Segmentierung von Bildern beschäftigt, ist die Medizin. Die Klassifizierung der medizinischen Daten basiert dabei auf denselben Methoden wie die Klassifizierung von Luftbildern. Der Forschungsstand im medizinischen Bereich ist allerdings wesentlich fortgeschrittener als in der Geoinformation. Zum Beispiel konnten VIGUERAS-GUILLEN et al. (2019) mithilfe eines CNNs deutlich bessere Ergebnisse erzielen, als es mit den aktuellen Methoden in der Medizin möglich ist. Auch andere Autoren wie ROUHI et al. (2015) oder KAYALIBAY et al. (2017) kommen zum Schluss, dass eine Klassifikation mithilfe von CNNs bessere Ergebnisse liefert als aktuelle Methoden.

Ein Großteil der Publikationen, sowohl im medizinischen Bereich als auch in der Fernerkundung, hat für die Programmierung der CNNs das Framework *TensorFlow* benutzt. Es ist jedoch wichtig hervorzuheben, dass die meisten Veröffentlichungen nur kleine Gebiete klassifizieren. Keine der gefundenen Publikationen beschäftigt sich mit der Klassifizierung von großen Gebieten, die über einen langen Zeitraum aufgenommen wurden.

An diesem Problem soll diese Masterarbeit ansetzen und die Klassifizierung der Orthofotos soll auf eine große Fläche ausgeweitet werden. Der bisherige Algorithmus soll durch ein CNN ergänzt bzw. signifikant verbessert werden und darüber hinaus sollen zusätzliche Bodennutzungsklassen entstehen.

Die zentrale Fragestellung, die im Rahmen dieser Masterarbeit beantwortet werden soll, lautet:

I. Ist der Einsatz eines CNNs für die Klassifizierung von Orthofotos geeignet und kann das Modell auch auf große Gebiete ausgeweitet werden?

Aus dieser Hauptfrage ergeben sich zusätzliche Nebenfragen, die auch mithilfe der Masterarbeit beantwortet werden sollen:

II. Welche Parameter sind für den Einsatz eines CNNs von Relevanz und wie sehen diese aus?
III. Wie viele Trainingsdaten sind notwendig und welche zusätzlichen Klassen können daraus gewonnen werden?
IV. Führt der Einsatz eines CNNs zu einer genaueren Klassifikation?

Alle vier Fragen werden ausschließlich im praktischen Teil der Arbeit erörtert. Am Ende der Arbeit soll ein kompletter Workflow von der Generierung der Trainingssamples bis zu einer Klassifikation vorhanden sein. Auf Basis dieser Schritte sollen die oben genannten Fragen beantwortet werden.

Nachfolgend ist eine kurze Zusammenfassung und Vorschau auf die Gliederung der Arbeit zu finden:

Im zweiten Kapitel dieser Masterarbeit soll zunächst kurz auf die verschiedenen Methoden der Landcover-Klassifikation eingegangen werden. Dabei werden sowohl bereits lang existierende als auch neuere Methoden beschrieben, zu denen auch die CNNs zählen.

Im dritten Kapitel wird zu Beginn die historische Entwicklung solcher CNNs beschrieben, wobei hierbei auch der biologische Grundgedanke hinter einem solchen Netzwerk eine Rolle spielt. Im Anschluss daran wird die Funktionsweise eines CNNs erläutert und ebenso wird auf die wichtigsten Parameter eingegangen. Dabei soll die Funktion dieser Parameter beschrieben werden und zudem wird erörtert, welche Methoden zur Findung der Parameter verwendet werden können. Darüber hinaus sollen die Funktionsweise und die Module von *TensorFlow* beschrieben werden.

Das vierte Kapitel umfasst den praktischen Teil der Analyse. Hierzu wird zunächst auf die Datengrundlage eingegangen. Dabei sollen mögliche Konstellationen, wie diese Daten verwendet werden können, beschrieben werden. Ferner beschäftigt sich das Kapitel mit dem Training, der Anpassung der Parameter und der Erstellung der finalen Klassifikationen.

Im fünften Kapitel werden die Ergebnisse aus dem praktischen Teil der Arbeit beschrieben und diskutiert. Insbesondere wird hierbei auf die Genauigkeit der Klassifikation eingegangen. Ebenso werden verschiedene Parameter des Modells einander gegenübergestellt und es wird verglichen, wie sie sich auf die endgültige Klassifizierung der Daten auswirken.

Im sechsten und abschließenden Kapitel der Arbeit werden die Ergebnisse zusammengefasst und eine Schlussfolgerung wird abgeleitet. Darüber hinaus soll diskutiert werden, wie die Klassifizierung weiter verbessert werden kann und in welcher Richtung weiterhin Forschungsbedarf besteht.

2. Landcover-Klassifikation

Dieses Kapitel beschäftigt sich mit dem theoretischen Teil der Landcover-Klassifikation in der Fernerkundung und Geoinformation. Dafür gibt es verschiedene Methoden, die sich zum Teil in hohem Maße voneinander unterscheiden. Dazu werden zunächst die theoretischen Grundlagen der verschiedenen Methoden erläutert. Ferner wird kurz die aktuelle Landcover-Klassifikation im österreichischen BEV vorgestellt. Im Anschluss werden die CNNs präsentiert, die das aktuellste Forschungsfeld auf dem Themengebiet der Landcover-Klassifikation sind. Darüber hinaus wird auf die zunehmend stärkere Integration von CNNs in Geoinformationssysteme eingegangen.

2.1. Methoden der Landcover-Klassifikation

Die Klassifikation der Bodenbedeckung auf Basis von Fernerkundungsdaten ist ein komplexes Problem, das durch viele Faktoren beeinflusst wird und noch immer ein großes Forschungsfeld ist. Die Ergebnisse einer solchen Klassifikation können unter anderem als Basis für umweltbezogene und sozioökonomische Untersuchungen verwendet werden. Im Normalfall sind Klassifikationen mit einer hohen räumlichen Auflösung nur für kleine und begrenzte Gebiete verfügbar. Dies ist auf die hohen Kosten für hochaufgelöste Fernerkundungsprodukte und den großen Aufwand, den eine Ableitung der Bodenbedeckung bedeutet, zurückzuführen (vgl. HAYES et al. 2014: 112).

Für die Ableitung solcher Produkte stehen verschiedene Methoden zur Verfügung, von denen einige wesentlich komplexer sind als andere. Grundsätzlich kann eine Landcover-Klassifikation nach verschiedenen Aspekten eingeteilt und kategorisiert werden. Eine gängige Einteilung ist die Kategorisierung in sogenannte überwachte und unüberwachte (‚supervised' und ‚unsupervised') Methoden. Bei der überwachten Methode werden zuvor Trainingsbeispiele generiert, die dann zum Training eines Klassifikationsmodells dienen. Bei der unüberwachten Methode werden die spektralen Informationen eines Bildes auf eine bestimmte Weise geclustert und auf dieser Basis wird eine Einteilung vorgenommen (vgl. HAYES et al. 2014: 112−113, LU und WENG 2007: 829).

Eine andere mögliche Einteilung der verschiedenen Methoden erfolgt nach LU und WENG (2007) auf Basis der Informationen darüber, welche Pixelinformationen zur Klassifikation benutzt werden. Folgende Typen werden von den Autoren genannt:

Tabelle 1: Einteilung der Landcover-Klassifikationsmethoden nach LU und WENG (2007)

Kategorie	Eigenschaften	Beispiele für Klassifikatoren
Per-pixel classifiers	Die spektralen Daten der Trainingsdaten werden auf gewisse Weise zusammengefasst und dienen dann zur Klassifikation.	Maximum likelihood, artificial neural network, support vector machine

Kategorie	Eigenschaften	Beispiele für Klassifikatoren
Subpixel classifiers	Ist der vorherigen Kategorie ähnlich, jedoch können sich die Bodenbedeckungskategorien überschneiden.	Fuzzy-set classifiers, subpixel classifiers
Object-oriented classifiers	Die Pixel werden auf Basis von unterschiedlichen Eigenschaften in Gruppen segmentiert und die Segmentierung wird anschließend klassifiziert.	eCognition
Per-field classifiers	Ist eine GIS-basierte Methode, in der auch zusätzliche Raster- und Vektordaten eingebunden sind.	GIS-Software

Alle in Tabelle 1 genannten Methoden haben ihre Vor- und Nachteile. Insbesondere für hochaufgelöste Daten ist der Aufwand für eine Ableitung der Bodenbedeckung groß. Je größer die zu klassifizierenden Gebiete sind, desto aufwendiger ist die Erarbeitung der Methoden, da diese sonst zu ungenauen Klassifizierungen führen würden (vgl. LU und WENG 2007: 848−849).

Aktuelle Methoden, zum Beispiel die der neuronalen Netzwerke, stecken noch immer in den Anfängen und werden kaum genutzt. Erst in den letzten acht Jahren wurde die Forschung in diese Richtung intensiviert und führte zu vielversprechenden Ergebnissen. Insbesondere die CNNs, die auf die Bildklassifikation spezialisiert sind, stellen eine aussichtsreiche Methode zur Ableitung der Bodenbedeckung dar (vgl. YANG et al. 2018: 251).

Auch im BEV wird derzeit eine Landcover-Klassifikation für das gesamte österreichische Bundesgebiet durchgeführt. Dazu werden die Orthofotos automatisch mithilfe der Software eCognition von Trimble segmentiert und anschließend werden die Segmente anhand von bestimmten Merkmalen klassifiziert. Es handelt sich somit um eine objektorientierte Klassifizierungsmethode (siehe Tabelle 1). Dazu werden neben den radiometrischen Informationen der Orthofotos auch die digitalen Geländemodelle verwendet. Auch das digitale Landschaftsmodell muss für einige wenige Sonderfälle (z. B. Brücken oder Gewässer) zur Klassifikation herangezogen werden. Die Klassen, die zurzeit mithilfe dieser Methode zuverlässig abgeleitet werden können, sind: Gebäude, Wald, Buschwerk, niedere Vegetation, Gewässer, vegetationslose Bodenflächen (vgl. BEV 2017: 75−78). Es soll nun durch eine verbesserte bzw. neuere Methodik wie die CNNs ergänzt werden.

2.2. Convolutional Neural Networks und Landcover-Klassifikation

Die CNNs zählen zu den modernsten Methoden im Bereich der Landcover-Klassifikation. Sie gehören zur ‚Per-pixel'-Kategorie und sind eine überwachte Methode. Ursprünglich waren CNNs ausschließlich dazu in der Lage, den gesamten Inhalt eines Bildes einer

einzelnen Kategorie zuzuteilen. Mittlerweile können sie jedoch auch jeden Pixel eines Bildes einer passenden Landcover-Klasse zuteilen. Herkömmliche neuronale Netzwerke lassen sich theoretisch auch für komplexe Bildklassifizierungsmethoden verwenden, jedoch sind die CNNs auf diese Aufgabe spezialisiert und deshalb besser geeignet (vgl. LIU 2019: 2, YANG et al. 2018: 251).

Der Vorteil dieser CNNs ist, dass sie die Funktionsweise des menschlichen Gehirns und infolgedessen auch des menschlichen Sehens imitieren. Im Vergleich zu den anderen Klassifizierungsmethoden schneidet ein CNN hinsichtlich der Genauigkeit in der Regel wesentlich besser ab. Insbesondere in den letzten fünf Jahren sind die CNNs zunehmend besser und genauer geworden. Dies ist einerseits auf die immer größere Rechenleistung und andererseits auf die immer umfangreicheren verfügbaren Datenmengen zurückzuführen. Für die Nutzung eines CNNs ist insbesondere die Anzahl der Trainingsdaten von großer Bedeutung. Mittlerweile sind die Funktionalitäten eines CNNs auch in der Software ArcGIS Pro der Firma ESRI integriert, was den Stellenwert der CNNs nochmals verdeutlicht. Die dazu passenden Werkzeuge sind seit Mai 2019 verfügbar und sollen den NutzerInnen bei der Anwendung eines CNN unterstützen (vgl. SINGH 2019, vgl. LIU 2019: 2).

Das Forschungsgebiet der CNNs zur Landcover-Klassifikation ist ein junger Zweig der Fernerkundung. Dabei wurden, insbesondere im Bereich der Satellitenfernerkundung, in den letzten Jahren große Fortschritte gemacht. Das Problem dabei ist jedoch, dass die räumliche Auflösung der Satellitendaten wesentlich geringer ist als zum Beispiel die von Orthofotos. Der Grund liegt in der schwierigeren und teureren Beschaffung von hochaufgelösten Daten, die aufgrund des Preises kaum für wissenschaftliche Arbeiten verwendet werden können. Insbesondere in diesem Bereich ist deshalb großer Forschungsbedarf vorhanden (vgl. MAHDIANPARI et al. 2018: 2−4).

3. Das Convolutional Neural Network (CNN)

In diesem Kapitel wird die geschichtliche Entwicklung von CNNs beschrieben, die eng mit dem biologischen Grundgedanken hinter dem menschlichen Sehen zusammenhängt. Im Anschluss wird die Funktionsweise von CNNs beschrieben, deren Kenntnis für das grundlegende Verständnis des praktischen Teils notwendig ist. Dazu werden zunächst die verschiedenen Arten von CNNs erklärt und erst dann werden deren relevanten Bestandteile erläutert. Im Anschluss werden die wichtigsten Parameter eines CNNs, die Hyperparameter, beschrieben und es wird erläutert, welche Herausforderungen es diesbezüglich geben kann. Abschließend wird kurz auf das Framework *TensorFlow* und die Implementierung desselben eingegangen.

Da sich die Masterarbeit mit der Klassifizierung der Bodenbedeckung beschäftigt, wird in diesem Kapitel hauptsächlich auf die flächendeckende Klassifizierung (Bildsegmentierung bzw. Image-Segmentation) eingegangen. Es gibt bei CNNs noch andere Klassifizierungsmethoden, die für diese Masterarbeit jedoch nur eine untergeordnete Rolle spielen.

3.1. Geschichte von Computer-Vision und CNNs

Die Idee, dass Computer bestimmte Bildinhalte erkennen können (Computer-Vision), und die Entwicklung der CNNs reicht bis in die 1960er Jahre zurück. Die Grundlagen dafür wurden im Jahr 1968 von HUBEL und WIESEL gelegt. Die beiden Autoren führten Untersuchungen an Gehirnen von Affen und Katzen durch und untersuchten insbesondere den visuellen Cortex. Dabei handelt es sich um jenen Bereich im Gehirn, der für die visuelle Wahrnehmung verantwortlich ist. Im Zuge dieser Studie wurden zwei neue Zelltypen entdeckt, die unter anderem für die Erkennung von Mustern von großer Bedeutung sind:

- Einfache Zellen: Diese Art von Zellen verwendet das Gehirn zur Erkennung von Kanten und Linien mit einer bestimmten Ausrichtung. Die einfachen Zellen reagieren somit auf die Orientierung eines visuellen Reizes.
- Komplexe Zellen: Diese dienen ebenfalls zur Erkennung von Kanten und Linien, besitzen jedoch ein wesentlich größeres Wahrnehmungsfeld. Dadurch können sie auch mehrere Objekte gleichzeitig wahrnehmen. Der wesentliche Unterschied zu den einfachen Zellen ist jedoch, dass komplexe Zellen die Bewegungen von Objekten erkennen können.

Hubel und Wiesel haben aus dieser Entdeckung geschlussfolgert, dass mithilfe der Funktionsweise dieser beiden Zellen Aufgaben zur Mustererkennung gelöst werden können (vgl. HUBEL und WIESEL 1968: 238−242).

Im Jahr 1980 wurde der Autor FUKUSHIMA von dieser Erkenntnis inspiriert und er entwarf das Neocognitron. Es beinhaltet bereits zwei elementare Schichten eines CNNs: den Convolutional Layer (beim Neocognitron heißt diese Schicht ‚S-Cells') und der Downsampling Layer (beim Neocognitron ‚C-Cells'). Beide Schichten sind bei modernen

CNNs von großer Relevanz und werden in allen Modellen verwendet (vgl. FUKUSHIMA 1980: 194).

Das erste CNN, wie es auch in der heutigen Ausprägung zu finden ist, wurde von LECUN et al. (1998) entworfen. Die Autoren ließen sich dazu vom Neocognitron inspirieren und präsentierten 1998 erstmals ein funktionierendes CNN. Die ersten Versuche beschränkten sich darauf, dass handgeschriebene Zahlen aus dem bekannten MNIST-Datensatz (Modified National Institute of Standards and Technology database) automatisch erkannt werden (siehe Abbildung 1). Während aktuelle Modelle eine Erkennungsrate von nahezu 100 % haben, klassifizierte das damalige Modell insgesamt 82 % der Beispiele richtig. Der Ansatz von LECUN et al. (1998) gilt heute als große Errungenschaft im Bereich des Visual-Computing und dient als Grundlage für alle modernen CNNs.

Abbildung 1: Beispiele aus dem MNIST-Datensatz (https://upload.wikimedia.org/wikipedia/commons/2/27/MnistExamples.png)

In den darauffolgenden Jahren und insbesondere in den letzten zehn Jahren wurden die verwendeten Modelle regelmäßig verbessert. Die CNNs waren in der Regel jedoch nicht im Alltag anwendbar, da es den Computern an der dafür benötigten Rechenkapazität mangelte. Im Jahr 2006 implementierten CHELLAPILLA et al. das erste CNN auf einer Grafikkarte (GPU), was zu deren Durchbruch führte. Im Vergleich zu einer Berechnung am Hauptprozessor (CPU) wurden die Ergebnisse auf der GPU viermal schneller berechnet. Mittlerweile kann durch den Einsatz von Grafikkarten eine Beschleunigung um den Faktor 60 erzielt werden (vgl. CIRESAN et al. 2011).

Durch die bessere und günstigere Verfügbarkeit von adäquater Hardware haben sich in den letzten Jahren zunehmend CNNs mit vielen verschiedenen Zwischenschichten (siehe Kapitel 3.3) gebildet. Aufgrund des Einsatzes einer großen Anzahl an Zwischenschichten wird der Ansatz auch als Deep-Learning-Methode bezeichnet.

3.2. Arten von Convolutional Neural Networks

Ein CNN kann für verschiedene Arten von Klassifizierungen benutzt werden. Das Grundgerüst des Modells bleibt dabei im Wesentlichen immer gleich und beinhaltet immer einen oder mehrere Convolutional und Pooling Layer. Die ersten Versuche mit einem CNN bezogen sich ausschließlich die Bilderkennung. Dabei sollte das neuronale Netzwerk die Bildinhalte von mehreren Bildern unterscheiden und klassifizieren können (z. B. Bilder von Hunden und Katzen oder Zahlen wie im MNIST-Datensatz). Die aktuellen Modelle können mittlerweile bis zu 1000 Klassen unterscheiden und erzielen dabei zuverlässige Ergebnisse (vgl. GÉRON 2019: 462).

Eine größere Herausforderung ist die Detektion und Lokalisation von Objekten auf einem Bild. Da ein CNN nicht selbstständig weiß, wo sich die Objekte auf einem Bild befinden, muss dies dem Modell zunächst beigebracht werden. Dazu müssen die Objekte auf den Bildern mit Begrenzungsrahmen versehen werden, die einer bestimmten Klasse zugeordnet werden. Dadurch kann das Modell lernen, wo sich Objekte auf einem Bild befinden, und dies ebenso auf andere unbekannte Bilder anwenden. Aktuelle Modelle können auch mehrere Begrenzungsrahmen unterschiedlicher Klassen auf einem einzelnen Bild erkennen (vgl. PLANCHE und ANDRES 2019: 15).

Die größte Herausforderung, die auch Teil dieser Masterarbeit ist, stellt jedoch die sogenannte semantische Segmentation von Bildern dar. Dabei wird jedes einzelne Pixel eines Fotos (z. B. eines Orthofotos) einer bestimmten Klasse, zum Beispiel Wald, Gras, Straße, zugeordnet. Es gibt mehrere Ansätze der pixelweisen Klassifizierung von Bildern, als beliebteste Art gilt jedoch ein sogenanntes Fully Convolutional Network, das in Kapitel 3.4 vorgestellt wird. Dabei unterscheidet sich die Architektur deutlich von derjenigen der anderen zuvor erwähnten Methoden, was diese Methode ein wenig komplexer macht (vgl. PLANCHE und ANDRES 2019: 16).

Für ein besseres Verständnis werden im nachfolgenden Kapitel die bedeutendsten Layer eines CNNs beschrieben (siehe Kapitel 3.3).

3.3. Aufbau eines Convolutional Neural Networks

Zu Beginn dieses Kapitels ist es notwendig, einen kurzen Überblick über die grundlegende Funktionsweise eines CNNs und die in diesem Zusammenhang relevanten Begriffe zu geben. Dies ist eine Voraussetzung für das weitere Verständnis der nachfolgenden Kapitel. Erst im Nachhinein werden die einzelnen Zwischenschichten genauer präsentiert.

3.3.1. Überblick über Convolutional Neural Networks

Die zentralen Bestandteile eines CNNs sind die Eingabeschicht (Input Layer), die Zwischenschichten (Hidden Layer) und die Ausgabeschicht (Output Layer). Als Eingabeschicht werden bei einem CNN in der Regel Bilder verwendet. Diese Bildinformationen werden zu Beginn in die Zwischenschichten überführt, wo der größte Teil der Berechnungen stattfindet. Diese Schichten bestehen aus einer mehrmaligen Abfolge von Convolutional Layer und Pooling Layer (siehe Kapitel 3.3.2 und 3.3.3).

Zusätzlich können auch andere Zwischenschichten, zum Beispiel Aktivierungs-Layer, Normalisierungs-Layer oder Dropout Layer (siehe Kapitel 3.3.4), eingebaut werden (vgl. PLANCHE und ANDRES: 77).

Bei einem klassischen CNN folgt nach den oben genannten Zwischenschichten ein sogenannter Fully Connected Layer (siehe Kapitel 3.3.4). Dieser überführt die Informationen aus dem CNN in ein herkömmliches neuronales Netzwerk. Mithilfe dieses gewöhnlichen neuronalen Netzwerkes wird die endgültige Zuteilung zu einer Klasse durchgeführt. Der Fully Connected Layer wird ausschließlich für die Klassifizierung eines Bildes, aber nicht für die Segmentierung eines Bildes benutzt. Für die Segmentierung wird auf den Fully Connected Layer verzichtet und stattdessen ein herkömmlicher Convolutional Layer benutzt. Dadurch entsteht ein Fully Convolutional Network, das in Kapitel 3.4 genauer erklärt wird (vgl. PLANCHE und ANDRES: 92−93).

Während des Trainings eines CNNs wird eine große Anzahl von Bildern in das neuronale Netzwerk eingespeist. Zusätzlich zu den Bildern werden passende Labels eingebracht. Diese können entweder einfache Beschreibungen (z. B. ‚Katze' oder ‚Hund') oder komplexe Maskierungen bzw. Segmentierungen sein. Bei den Masken muss jedes Objekt im Bild einer bestimmten Klasse zugeteilt und auch eingezeichnet werden. Durch das Training lernt das Modell, bestimmte Merkmale auf einem Bild zu erkennen, und leitet daraus letztendlich eine Klassifikation ab (vgl. PLANCHE und ANDRES: 203).

Da die Trainingsbilder in den meisten Fällen aus mehreren tausend Bildern bestehen, können diese dem CNN nicht in einem einzigen Schritt präsentiert werden. Stattdessen erfolgt die Einspeisung der Daten in kleinen Teilen, den sogenannten Batches. Sind alle Batches durch das Modell gelaufen, wird eine Epoche beendet. Ist eine solche beendet, werden gewisse Anpassungen am Modell vorgenommen und im Anschluss wird eine neue Epoche gestartet. Mit jedem Durchgang lernt das Modell, eine genauere Klassifikation der Bilder durchzuführen. Damit eine Genauigkeit bzw. Zuverlässigkeit überprüft werden kann, ist es von großer Bedeutung, dass nicht alle vorhandenen Daten zum Training benutzt werden. Vor dem Training muss daher ein sogenannter Train/Test-Split durchgeführt werden. Dazu werden zunächst 20 % der Daten als Validierungsdaten zurückgehalten. Diese werden nach jeder Epoche zur Überprüfung der Genauigkeit verwendet. Das Modell wird so lange trainiert, bis sich die Genauigkeitsmaße nicht mehr verbessern bzw. konstant bleiben (vgl. PLANCHE und ANDRES: 40−49). In Abbildung 2 ist ein Beispiel für ein einfaches CNN-Modell dargestellt.

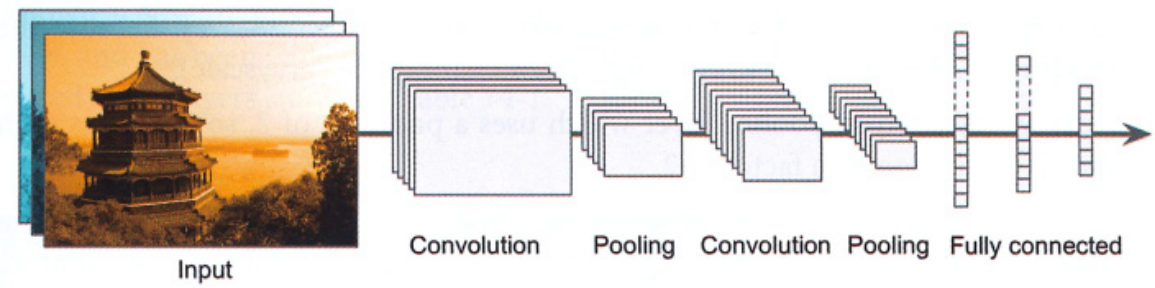


Abbildung 2: Beispiel für den Aufbau eines einfach CNNs (GÉRON 2019: 461)

Die einzelnen Schichten und die dazugehörigen Parameter werden in den folgenden Kapiteln für ein besseres Verständnis genau erläutert.

3.3.2. Convolutional Layer

Der Convolutional Layer ist jener namensgebende Teil eines CNNs, in dem die wesentlichsten und umfangreichsten Berechnungen durchgeführt werden. In einem Modell kommt nicht nur ein solcher Layer vor, vielmehr kann ihre Anzahl sehr umfangreich werden. Über hundert solcher Schichten sind dabei keine Seltenheit. Für jeden dieser Layer ist ein Input und ein Output vorhanden. Beim ersten Convolutional Layer in einem Modell sind die Bilder der Input in diese Schicht. In den darauffolgenden Layers wird der Output der vorangegangenen Schicht als Input herangezogen (vgl. GÉRON 2019: 448−453).

Während der Berechnungsphase werden Filter über den Input des Convolutional Layers geschoben. Diese Filter sind eine Anordnung von numerischen Werten, die auch als Gewichte bzw. *Weights* bezeichnet werden. Solche Filter haben in der Regel eine Größe von 3 × 3 oder 5 × 5 Pixel und eine Tiefe, die sich über die Tiefe des Inputs erstreckt (z. B. bei einem RGB-Bild hat der Filter eine Tiefe von 3). Mit einem solchen Filter können spezielle Merkmale eines Bildes, zum Beispiel vertikale oder horizontale Linien, hervorgehoben werden (vgl. GÉRON 2019: 450−451). Zwei Beispiele für solche Filter sind in Abbildung 3 dargestellt.

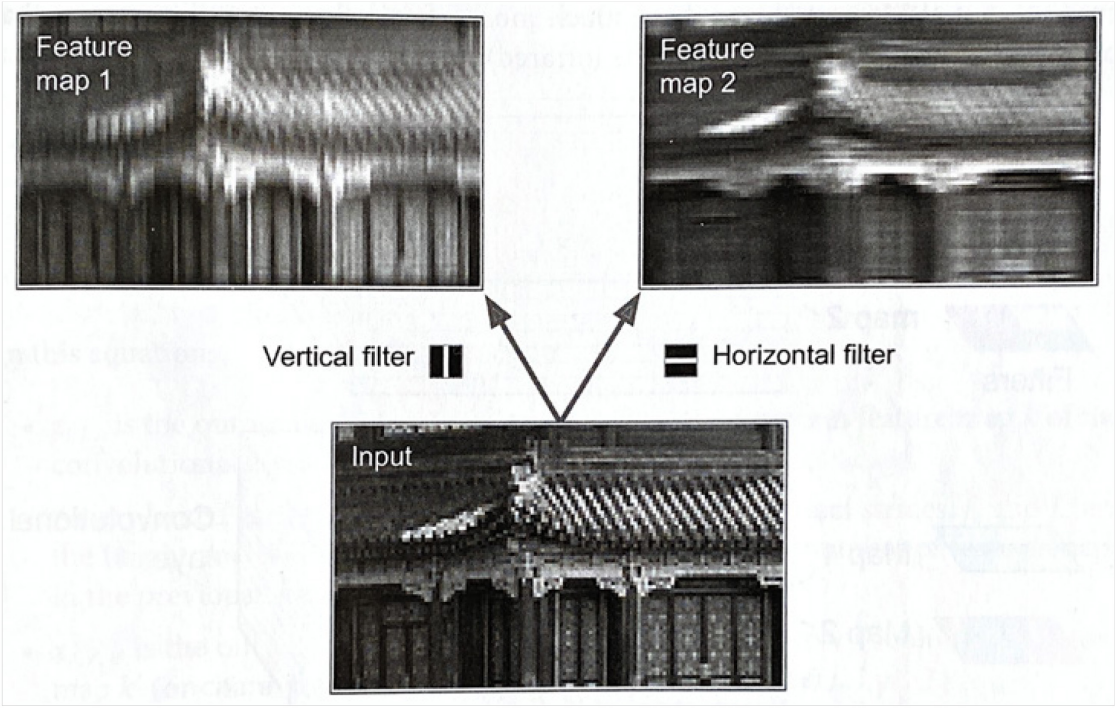


Abbildung 3: Beispiel für zwei verschiedene Filter (GÉRON 2019: 451)

Jedes Bild hat nicht nur einen Filter, sondern gleich mehrere. Da die Filter wesentlich kleiner als der Input sind, werden sie über den Input geschoben. An jeder Position werden die Gewichte der Filter mit den Werten des Inputs multipliziert und anschließend werden

diese Werte summiert (siehe Abbildung 4). Im Anschluss wird der Filter um eine vom Benutzer gewählte Schrittweite weitergeschoben. An der neuen Position wird der Vorgang erneut durchgeführt, bis der komplette Input auf diese Weise bearbeitet wurde. Die Zahlenwerte, die durch diese Berechnung entstehen, werden auch als Neuronen bezeichnet. Im Gegensatz zu einem herkömmlichen neuronalen Netzwerk liegen diese Neuronen bei einem CNN in einer zweidimensionalen Anordnung vor. Diese wird auch als Feature-Map bezeichnet. Jeder Filter produziert dabei eine eigene Feature-Map (vgl. GÉRON 2019: 448−453).

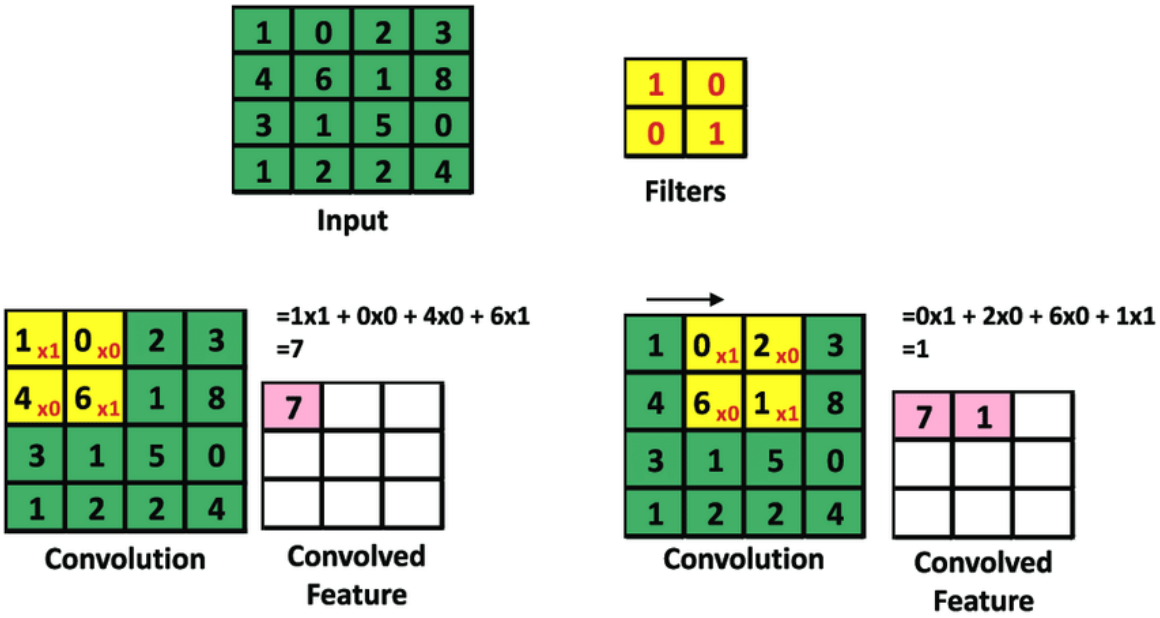


Abbildung 4: Darstellung der Operationen zur Erstellung eines gefalteten Objektes (NG et al. 2019: 254)

Die Dimensionen von solchen Feature-Maps ergeben sich nach NG et al. 2019 aus mehreren Parametern:

- Die Inputdimensionen (W)
- Die Filterdimensionen (F)
- Die Schrittgröße (S): Die Schrittgröße gibt die Zahl der Pixel an, um die der Filter bei jedem Schritt verschoben wird.
- Das Padding (P): Beim Padding werden an den äußeren Kanten des Inputs zusätzliche ‚leere' oder zufällige Werte hinzugefügt. Damit kann bewerkstelligt werden, dass der Output dieselben Dimensionen wie der Input hat.

Die Dimensionen der Feature-Maps können mit folgender Formel berechnet werden:

$$\text{Größe} = \frac{(W - F + 2P)}{S} + 1$$

Die einzelnen Neuronen besitzen zudem ein sogenanntes rezeptives Feld, das durch die Größe des Filters definiert wird. Manchmal wird dieses Feld auch als *Field of View* bezeichnet. Das rezeptive Feld ist jener Bereich im Input, der von dem Filter des jeweiligen Neurons abgedeckt wird. Das rezeptive Feld ist somit jener Bereich bzw. jenes Feld, aus dem sich die Zahl des jeweiligen Neurons ergeben hat (vgl. LUO et al.

2016: 1−2). In Abbildung 4 sind die rezeptiven Felder durch die gelben Bereiche in den Convolutions visualisiert.

Die Filter müssen nicht selbst definiert werden, sondern sie werden im Zuge des Trainings automatisch an die Problemstellung angepasst. Es wird somit versucht, die Filter so anzupassen, dass sie die Merkmale der gesuchten Klassen hervorheben und erkennen können (vgl. GÉRON 2019: 450).

Der letzte Schritt im Convolutional Layer beinhaltet die Aktivierung dieser Feature-Map mit einer Aktivierungsfunktion. Im Wesentlichen überführt sie die Neuronen in ein komplexes nichtlineares System. (Das Ausgangssignal ist nicht immer proportional zum Eingangssignal). Damit sollen besondere Merkmale im Bild, zum Beispiel Kanten oder spezifische Farben, besonders hervorgehoben werden (vgl. ADAMY 2014: 1).

In der Trainingsphase wird nach jedem Batch der gemachte Fehler berechnet. Dies geschieht über eine sogenannte Loss-Funktion, welche die durch das Modell berechnete Vorhersage und das tatsächliche Label vergleicht und anschließend den Fehler berechnet. Dafür gibt es verschiedene Loss-Funktionen, die in Kapitel 3.5 beschrieben werden.

Das Ergebnis der Loss-Funktion wird zudem für eine Fehlerrückrechnung (Backpropagation) herangezogen. In der Trainingsphase werden alle Ergebnisse, auch die der Zwischenschichten, behalten, wodurch der gemachte Fehler vom Output zum Input zurückgerechnet wird. Es kann somit geprüft werden, welches Neuron bzw. welcher Filter zum Fehler beigetragen hat. Mit diesem Wissen kann anschließend der sogenannte Error-Gradient für jeden Parameter im Modell berechnet werden. Dieser gibt an, in welche Richtung und wie stark die Gewichte der Filter verändert werden müssen, damit es zu einer Verringerung des soeben berechneten Fehlers kommt. Auf Basis dieses Gradienten werden anschließend die Parameter durch das Gradientenverfahren (Gradient Descent Step) angepasst. Das Ziel ist es, den Fehler so auf ein Minimum zu verringern (vgl. INDOLIA et al. 2018: 684−686).

3.3.3. Pooling Layer

Der Pooling Layer kann, muss jedoch nicht im Anschluss an einen Convolutional Layer folgen. Er besitzt keine lernbaren bzw. anpassbaren Parameter und dient ausschließlich zur Reduktion der Datenmenge, die durch das Modell berechnet werden muss. Dadurch werden die Rechenlast, die benötigte Speichermenge und die Anzahl der lernbaren Parameter erheblich reduziert (vgl. GÉRON 2019: 456−458).

Beim Pooling Layer müssen ebenso die Größe des Filters, das Padding und der Stride angegeben werden. Der Filter wird dabei erneut über das Bild geschoben und wie beim Convolutional Layer werden pro Neuron ausschließlich die Werte im Bereich des rezeptiven Feldes für die Berechnung herangezogen. Grundsätzlich gibt es zwei wesentliche Pooling-Funktionen:

- **Max-Pooling:** Es wird der höchste Wert im Rezeptiven Feld des Output-Neurons genommen. Dieser Wert bildet anschließend das Output-Neuron (siehe Abbildung 5).

- **Average-Pooling:** Die Werte im Rezeptiven Feld des jeweiligen Output-Neurons werden gemittelt. Auch hier führt der gemittelte Wert zum Wert des Output-Neurons.

Das Max-Pooling ist dabei die am weitesten verbreitete Operation, die verwendet wird. Pooling Layer können jedoch auch den Nachteil haben, dass wichtige Informationen aufgrund der Verkleinerung verloren gehen. Im Normallfall ist dies jedoch kaum ein Problem und kann in den meisten Fällen ignoriert werden (vgl. INDOLIA et al. 2018: 681−682).

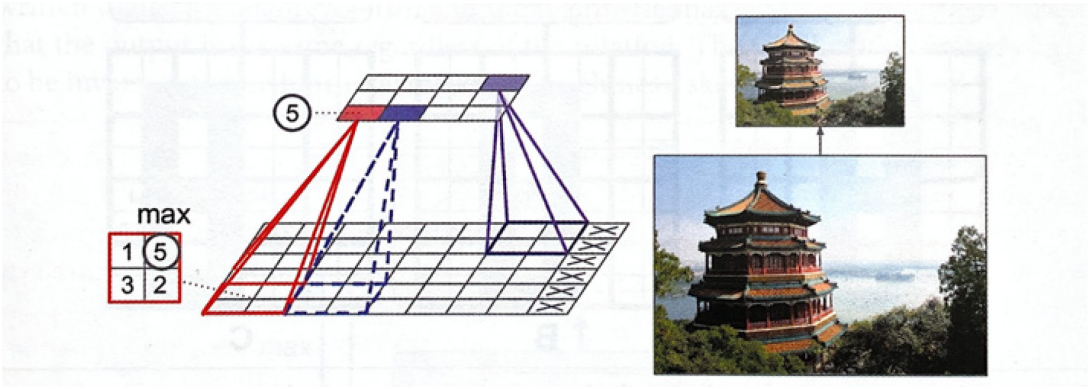


Abbildung 5: Beispiel für ein Max-Pooling (GÉRON 2019: 457)

3.3.4. Andere wichtige Zwischenschichten

Neben dem Convolutional und dem Pooling Layer gibt es noch zusätzliche Zwischenschichten, die bei der Klassifikation von Bildern und der Stabilität des Modells helfen können.

Dropout Layer: Der Dropout Layer wurde erstmals im Jahr 2012 von HINTON et al. vorgestellt. Mittlerweile ist diese Zwischenschicht eine der beliebtesten Regulierungsmethoden, die ein Overfitting (siehe Kapitel 3.6) verhindern sollen. Aktuelle Modelle profitieren zudem von einer verbesserten Genauigkeit von ungefähr 2 % (vgl. GÉRON 2019: 365−368; HINTON et al. 2012: 1−6).

Bei einem Dropout hat jedes Neuron eine zuvor definierte Wahrscheinlichkeit, deaktiviert zu werden. Es wird bei dem jeweiligen Trainingsschritt also ignoriert, kann aber bereits im nächsten Trainingsschritt wieder aktiv sein. Bei einem CNN werden in der Regel zwischen 40 und 50 % der Neuronen deaktiviert. Nach dem Trainingsvorgang, werden wieder alle Neuronen verwendet und es wird kein Dropout mehr angewandt (vgl. GÉRON 2019: 365−368; HINTON et al. 2012: 2).

Durch die Deaktivierung der Neuronen muss ein anderer Weg durch das neuronale Netzwerk gefunden werden. Dadurch lernen Neuronen, dass sie sich nicht ausschließlich auf die benachbarten Neuronen verlassen können und dass alle Inputs eines Neurons beachtet werden müssen (siehe Abbildung 6). Durch die Anwendung eines Dropout Layers wird das Modell zuverlässiger trainiert und kann nach der Trainingsphase besser generalisieren (vgl. GÉRON 2019: 365−368).

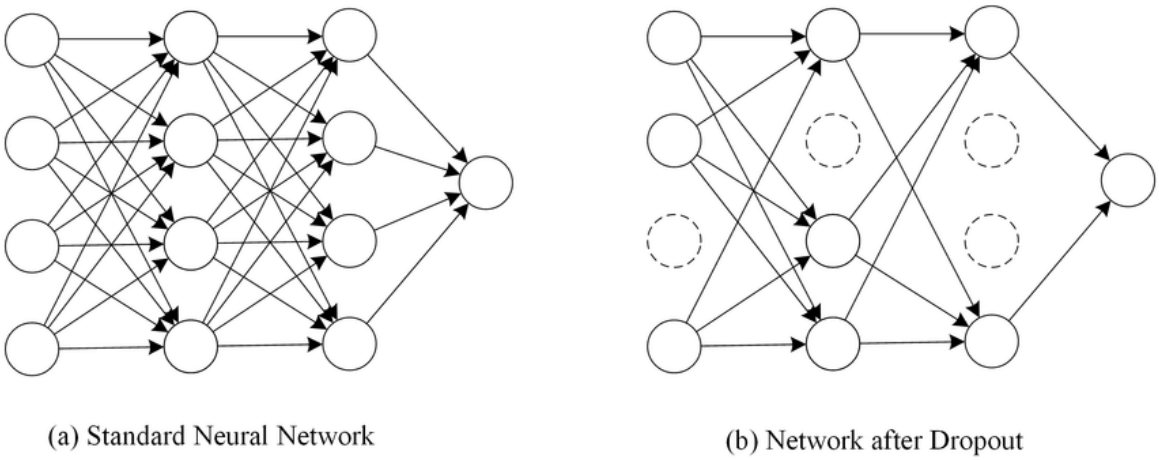


Abbildung 6: Beispielhafte Darstellung für einen Dropout (KHALIFA und FRIGUI 2016: 11)

Normalisierungs-Layer: Die Normalisierung, die oft auch als Batch-Normalization bezeichnet wird, wird im Normalfall vor oder hinter einer Aktivierung vorgenommen. Bei dem Vorgang erfolgt eine Mittelwertzentrierung (also eine Zentrierung der Werte um Null) und eine anschließende Standardisierung der Daten (vgl. IOFFE und SZEGEDY 2015: 1).

Der Normalisierungsprozess wurde erstmals im Jahr 2015 von den Autoren IOFFE und SZEGEDY präsentiert. Sie konnten mit der Batch-Normalization bei allen damals vorhandenen Modellen wesentliche Verbesserungen in der Lernzeit und in der Top-Fünf-Fehlerrate (siehe Kapitel 3.3.4) erzielen. Durch den Einsatz eines solchen Layers kann auch die Notwendigkeit einer Regulierungsmethode wie dem Dropout Layer entfallen (vgl. IOFFE und SZEGEDY 2015: 1).

Fully Connected Layer: Wie bereits in im Kapitel 3.3.1 beschrieben, wird dieser Layer für diese Masterarbeit von keiner großen Relevanz sein, da er bei der Segmentierung von Bildern (Image Segmentation) nicht verwendet wird. In einem klassischen CNN dient der Fully Connected Layer jedoch zur finalen Klassifikation des Bildinhaltes, wodurch er trotzdem von großer Relevanz und für das Verständnis bedeutsam ist.

Der Fully Connected Layer besteht aus einem oder mehreren (oftmals 2000) Neuronen, die mit allen Neuronen des Inputs, zum Beispiel den jeweiligen Feature-Maps aus dem vorangegangenen Layer, verbunden sind. Dazu müssen die Feature-Maps zunächst ausgerollt (‚flatten') werden. Die ausgerollten Neuronen werden anschließend summiert und dann mit einer erlenbaren Gewichtung multipliziert. Daraus ergibt sich die gewichtete Summe (‚weighted sum'). Diese Gewichtungen des Fully Connected Layers werden ähnlich wie die Filter der Convolutional Layer während des Trainings erlernt (vgl. PU et al. 2019: 5−7).

Die Ergebnisse sind dimensionslose Neuronen, die keinerlei Form mehr besitzen und nur einen Wert repräsentieren. Die Anzahl dieser Neuronen wird durch den Benutzer festgelegt. Im Normalfall werden mehrere Fully Connected Layer hintereinander positioniert. Der letzte Layer eines Modells sollte immer ein Fully Connected Layer sein, der so viele Neuronen hat, wie es finale Klassen gibt. Dieser wird je nach

Aufgabenstellung mit einer speziellen Aktivierungsfunktion aktiviert. Das Output-Neuron, das am Ende den größten Wert aufweist, gibt auch die vorhergesagte Klasse an (vgl. PU et al. 2019: 5−7). Weist zum Beispiel das zweite Neuron den höchsten Wert auf, ist das Bild dem Modell zufolge, der zweiten Klasse zuzuordnen.

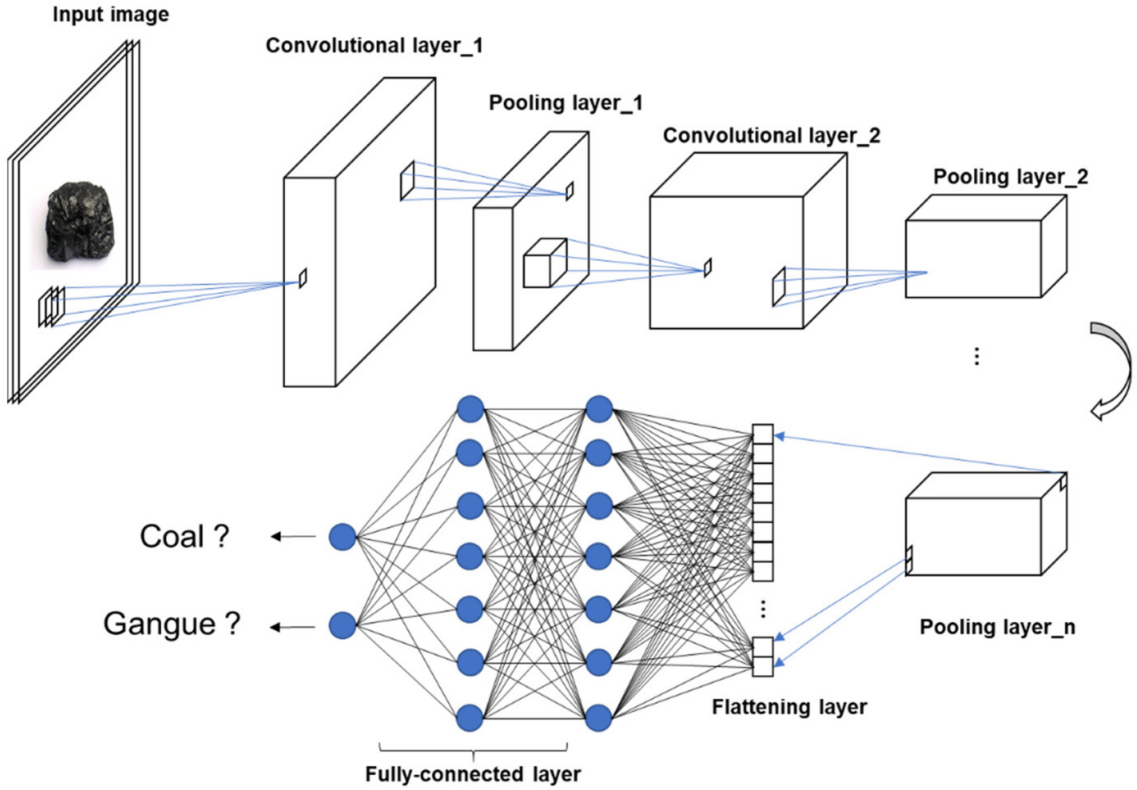


Abbildung 7: Beispiel für ein Modell mit Fully Connected Layer (PU et al. 2019: 3)

3.4. Fully Convolutional Network

Wie oben beschrieben ist ein CNN eine Abfolge von Convolutional und Pooling Layer und einem oder mehreren anschließenden Fully Connected Layer. Diese Form kann jedoch nicht zur Segmentierung von Bildern eingesetzt werden. Bei einem Fully Convolutional Network wird der Fully Connected Layer entfernt und durch einen herkömmlichen Convolutional Layer ersetzt. Der Unterschied besteht darin, dass der Output eines Convolutional Layers, also die Feature-Maps, eine räumliche Dimension hat. Die Neuronen des Fully Connected Layers besitzen jedoch keine Dimensionen und spiegeln nur Werte wider. Dies führt dazu, dass vom Fully Connected Layer lediglich eine zuvor definierte Inputgröße akzeptiert wird und verwendet werden kann. Bei einer abweichenden Größe kommt es zu einem sofortigen Abbruch der Berechnung. Im Gegensatz zu den Fully Connected Layers benötigen Convolutional Layers keine fixierte Inputgröße und können mit verschiedenen Inputgrößen umgehen (vgl. GÉRON 2019: 487).

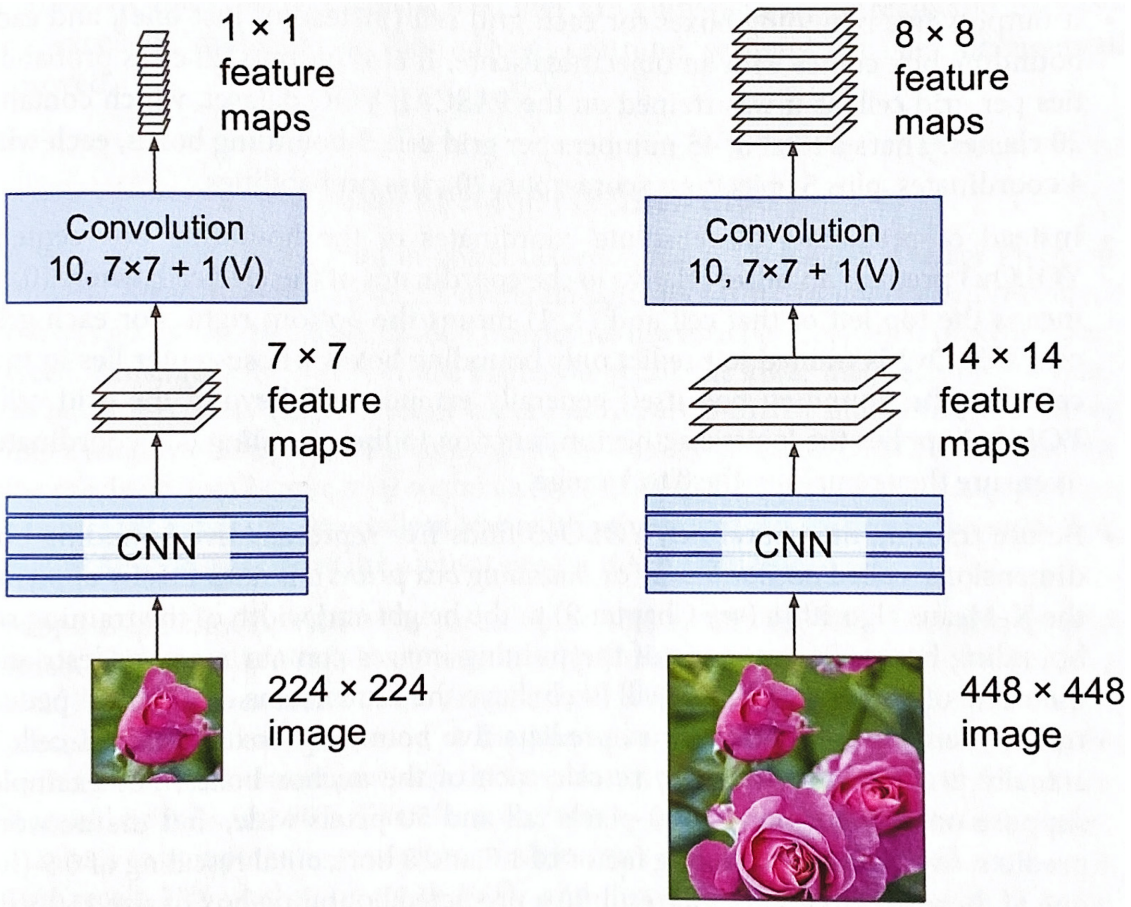


Abbildung 8: Umgang eines Convolutional Layers mit verschiedenen Inputgrößen (GÉRON 2019: 489)

Da durch den Vorgang einer Convolution die räumliche Auflösung (die Anzahl der Pixel) der Feature-Maps kontinuierlich geringer wird, können die Ergebnisse nicht einfach für eine Segmentierung verwendet werden. Zuvor müssen diese wieder auf die Größe des originalen Bildes hochskaliert werden. Deshalb besteht ein Fully Convolutional Network nach JÉGOU et al. (2017: 11−14) aus folgenden Teilen:

- Downsampling: Beinhaltet die klassischen Convolutional und Pooling Layer. Dieser Vorgang dient zur Extraktion von Kontextinformationen sowie von semantischen Informationen.
- Upsampling: Damit die Informationen aus dem Downsampling wiederverwendet werden können, muss die Information wieder auf die Größe des Bildes gebracht werden. Dazu werden oft sogenannte Skip-Connections verwendet. Dabei werden Informationen aus den Feature-Maps des Downsamplings in jene des Upsamplings übertragen. Dadurch kommt es zu keinem Verlust von Informationen aus dem Downsampling. Ohne diesen Vorgang der Informationsübertragung wäre die Segmentation der Bilder kaum möglich.

Für das Upsampling der Daten stehen verschiedene Methoden zur Verfügung. Die am häufigsten genutzte Methode ist der Transposed Convolutional Layer. Dabei werden zwischen den Pixelreihen und -spalten der Feature-Maps zusätzliche leere Spalten und Reihen eingefügt. Zusätzlich werden am Rand auch leere Pixel eingesetzt. Auf Basis dieser neuen Feature-Map wird anschließend eine herkömmliche Convolution mit einem

Filter durchgeführt. Der Output ist durch das Einfügen der leeren Pixel ein größeres Bild als der Input (siehe Abbildung 9) (vgl. DUMOULIN und FISIN 2016: 19−28).

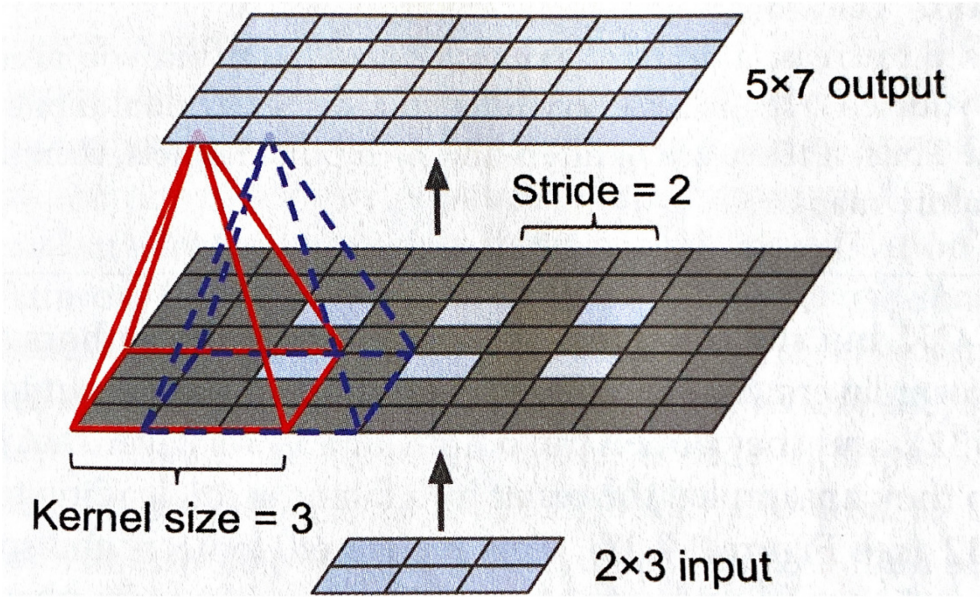


Abbildung 9: Upsampling durch einen Transposed Convolutional Layer (GÉRON 2019: 493)

Eines der bekanntesten Fully Convolutional Networks ist das U-Net, das auch für diese Masterarbeit benutzt wird (siehe Kapitel 4.4.1). Das U-Net wurde von RONNEBERGER et al. (2015) veröffentlicht und zu Beginn ausschließlich für die Segmentierung von medizinischen Bildern verwendet. Mittlerweile wird es jedoch in vielen anderen Bereichen, zum Beispiel in der Geoinformation, erfolgreich eingesetzt. Für eine Segmentierung der Bilder müssen diese zunächst mit Masken versehen werden. Dazu müssen die Bilder zunächst händisch segmentiert und klassifiziert werden. Dies kann mit einem klassischen Bildbearbeitungsprogramm, zum Beispiel Photoshop, oder einem Geoinformationssystem gemacht werden. Dazu werden die gewünschten Klassen lagegenau eingezeichnet und die Labels als Graustufenbild exportiert. Jede gewünschte Klasse bekommt dabei einen bestimmten zuvor zugeteilten Grauwert. Der genaue Klassifizierungsvorgang wird in Kapitel 4.2 erläutert.

Die Anordnung der Layer im U-Net besteht ebenso aus einem Downsampling und einem Upsampling. Die Architektur des Upsamplings orientiert sich an der Abfolge des Downsamplings inklusive der Dimension und Anzahl der Feature-Maps des Upsamplings. Es handelt sich grundsätzlich um ein umgedrehtes Downsampling. Mithilfe einer sogenannten Skip-Connection werden Informationen aus den Feature-Maps des Downsamplings in jene des Upsamplings übertragen. So können auch zentrale Kontextinformationen des Bildes für die Segmentierung benutzt werden. Nach einem erfolgreichen Training können die gelernten Informationen auf ein neues unbekanntes Bild übertragen werden, wodurch dieses pixelgenau klassifiziert wird (vgl. RONNEBERGER et al. 2015: 1-4).

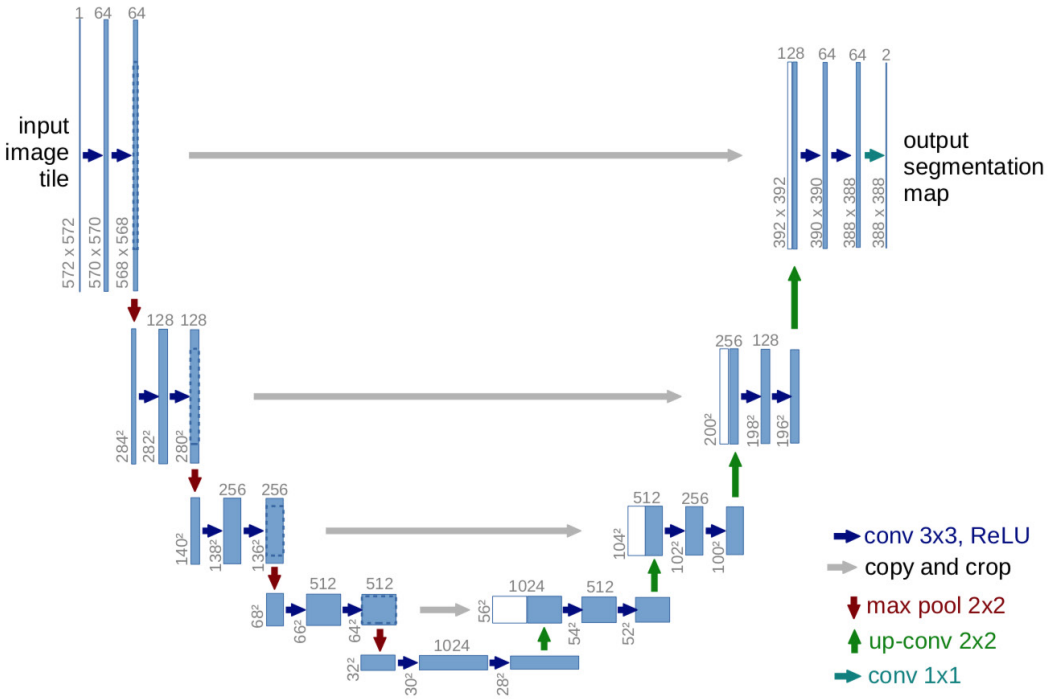


Abbildung 10: Architektur des U-Net (RONNEBERGER et al. 2015: 2)

Aufgrund der Architektur als Fully Convolutional Network kann als Input ein Bild mit einer beliebigen Größe gewählt werden. Es wird jedoch empfohlen, immer ein Vielfaches von 256 Pixeln als Größe zu wählen. Während des Trainings müssen kleinere Bilder (512 × 512 oder 256 × 256 Pixel) benutzt werden, da hier mehr Rechenkapazität benötigt wird. Nachdem das Modell jedoch trainiert ist, können wesentlich größere Bilder segmentiert werden, da zum Beispiel keine Fehlerrückführung mehr gerechnet werden muss (vgl. YAKUBOVSKIY o. J.).

Da im Zuge des praktischen Teils dieser Masterarbeit ein U-Net verwendet wird, soll dieses im dritten Kapitel näher erläutert werden.

3.5. Hyperparameter in einem Convolutional Neural Network

In einem CNN-Modell gibt es mehrere relevante Einstellungen, sogenannte Hyperparameter, die vom Nutzer selbstständig festgelegt werden müssen. Bevor ein neuronales Netzwerk selbständig und flexibel Entscheidungen bezüglich einer Klassifikation treffen kann, muss es mit den passenden Hyperparametern trainiert werden. Diese müssen zunächst bestimmt und festgelegt werden. Für die Findung derselben stehen verschiedene Methoden zur Verfügung. Die wichtigsten Hyperparameter werden im Folgenden vorgestellt:

Batchsize: Die Batchsize beschreibt die Anzahl der Trainingsbeispiele, die dem Modell in einem Trainingsschritt gezeigt werden. Das Modell wird auf Basis solcher Batches trainiert. Da die Hardwarekapazitäten der PCs limitiert sind (RAM und VRAM), wird dem Modell nur ein Teil der Bilder auf einmal gezeigt. Die Batchsize sollte dabei immer ein Exponent von zwei sein. Nachdem alle Beispiele (bzw. alle Batches) dem Modell präsentiert wurden, wird eine sogenannte Epoche beendet. Nach einer Epoche wird

zunächst der Loss und eine Fehlerrückführung (siehe nachfolgende Hyperparameter) auf Basis dieses Wertes berechnet. Als Abschluss wird das zuvor trainierte Modell noch auf die Validierungsdaten getestet und danach wird eine neue Epoche gestartet (vgl. GÉRON 2019: 326).

Batches mit einem großen Umfang haben den Vorteil, dass das Modell schneller berechnet wird, jedoch benötigen sie viel Videospeicher. Zudem können große Batches in der Praxis dazu führen, dass das Modell instabil wird und am Ende nicht mehr gut generalisieren kann. MASTERS und LUSCHI (2018: 16) sind auch zu dem Schluss gekommen, dass kleine Batchsizes wesentlich bessere Modelle in kürzerer Zeit erzeugen. Andere Autoren wie HOFFER et al. (2017: 8−9) gehen hingegen davon aus, dass Modelle mit den richtigen Hyperparametern auch mit großen Batchsizes trainiert werden können.

Loss-Funktion: Damit die Performance des Modells während des Trainings gemessen werden kann, wird die Loss-Funktion benötigt. Diese Funktion misst den Fehler, den das Modell während des Trainings macht. Auf Basis dieses Wertes wird bei jeder Epoche die Fehlerrückführung berechnet, wodurch dieser Wert von großer Relevanz ist. Bei dieser Funktion wird im Anschluss einer Epoche versucht, den Wert des Losses zu minimieren (Gradient Descent) und diesen dem Wert Null anzunähern (vgl. TRUONG 2019).

Dazu stehen verschiedene Loss-Funktionen zur Verfügung, die je nach der Problemstellung ausgewählt werden müssen. Jede dieser Loss-Funktionen hat bestimmte Vor- und Nachteile:

- **Cross-Entropy:** Beim Cross-Entropy-Loss kann die Performance eines Modells getestet werden, bei dem der Output eine Wahrscheinlichkeit für eine Zugehörigkeit einer Klasse zwischen 0 und 1 angibt. Bei der Cross-Entropy handelt es sich um eine logarithmische Kurve (siehe Abbildung 11). Diese bestraft schlecht oder falsch vorhergesagte Klassenzugehörigkeiten mit einem entsprechend hohen Loss-Wert (vgl. N. N. 2017).

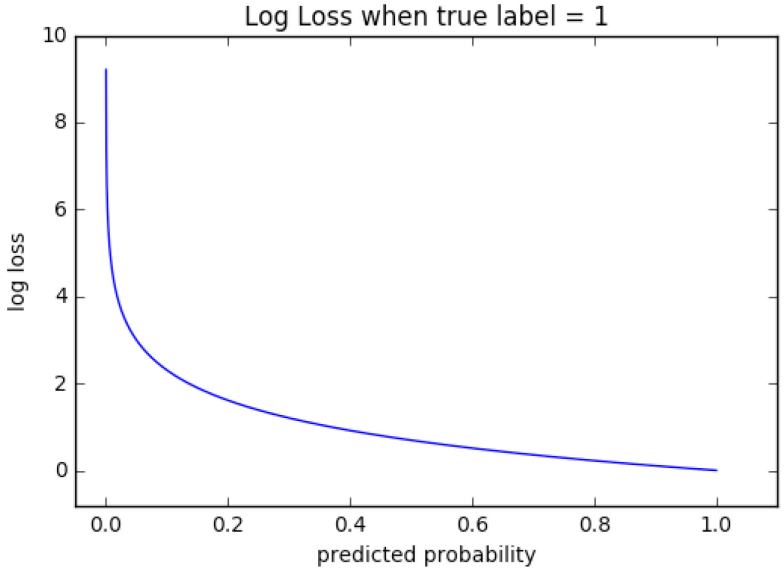


Abbildung 11: Cross-Entropy-Loss (https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html)

Bei der Segmentierung von Bildern muss die Loss-Funktion an die Problemstellung angepasst werden, sodass jeder individuelle Pixel beobachtet wird. Dabei wird die Wahrscheinlichkeit der Zugehörigkeit eines Pixels zu einer bestimmten Klasse mit dem tatsächlichen Label verglichen und der Fehler pro Pixel berechnet. Im Anschluss daran werden diese Ergebnisse für das ganze Bild gemittelt. Diese Berechnungsweise kann jedoch zu Problemen führen, wenn es eine ungleiche Verteilung der verschiedenen Klassen gibt, da der Fokus dann nahezu ausschließlich auf jenen Klassen liegt, die überrepräsentiert sind. Um dies zu verhindern kann die Weighted Cross-Entropy verwendet werden, die ein solches Problem verhindern soll. Bei dieser bekommen die einzelnen Klassen während des Trainings eine bestimmte Gewichtung zugeteilt (vgl. AURELIO et al. 2019: 1).

- **Focal-Loss:** Im Bereich der Segmentierung von Bildern hat sich der Focal-Loss zu einer beliebten Loss-Funktion entwickelt. Er kann gut mit einer ungleichen Verteilung von verschiedenen Klassen umgehen, da er den Fokus auf Klassen legt, die schwer zu klassifizieren sind. Im Normalfall sind dies jene Klassen, die im Datensatz unterrepräsentiert sind (vgl. LIN et al. 2017: 1−2).
  Während des Trainings besitzen somit jene Klassen, die einfach zu klassifizieren sind, eine leichtere Gewichtung bei der Berechnung des Fehlers als jene Klassen, die schwer zu klassifizieren sind (vgl. LIN et al. 2017: 1−2).
- **Dice-Loss:** Der Dice-Loss ist ebenso eine beliebte Loss-Funktion, die bei der Bildsegmentierung eingesetzt wird. Im Wesentlichen wird mit dem Dice-Loss die Überlappungsgenauigkeit zwischen der Vorhersage und den zuvor eingezeichneten Labels (Ground Truth) verglichen. Je höher die Überlappung ist, desto höher der Wert, wobei das Maximum bei 1 erreicht ist (vgl. NIERADZIK 2018 und JORDAN 2018).
- **Kombinationen:** Alle Losses können auch miteinander kombiniert werden. Dazu werden die Ergebnisse der jeweiligen Funktionen addiert. Eine beliebte Kombination in der Bildsegmentierung ist die Kombination des Cross-Entropy-Losses und des Dice-Losses durch eine Addition (vgl. NIERADZIK 2018).

Input-Neuronen: Die Anzahl der Input-Neuronen kann ausschließlich durch die Größe des Eingangsbilds bestimmt und angepasst werden. Je größer die Trainingsbilder sind, desto mehr Input-Neuronen sind vorhanden. Ein Graustufenbild mit 256 × 256 Pixeln hat zum Beispiel insgesamt 65 536 Input-Neuronen. Je größer die Anzahl der Input-Neuronen ist, desto mehr Videospeicher und Rechenleistung benötigt der Computer (vgl. GÉRON 2019: 325).

Zwischenschichten: Die Anzahl der zuvor beschriebenen Zwischenschichten kann ebenso vom Nutzer festgelegt werden und stark variieren. Zudem muss nicht jede Zwischenschicht zwingend verwendet werden. Zum Beispiel kann auf den Dropout Layer verzichtet werden. Wichtig dabei ist jedoch, dass die Abfolge von den Schichten immer an gewisse Vorgaben gebunden ist. Beim Dropout Layer ist zum Beispiel folgende Reihenfolge einzuhalten: Convolutional Layer → Pooling Layer → Dropout Layer. Die Anzahl der Zwischenschichten, die verwendet werden, muss an die Problemstellung

angepasst und ausgetestet werden. Einige bekannte Modellvarianten (siehe Kapitel 3.7) haben teilweise mehr als 100 Zwischenschichten (vgl. GÉRON 2019: 323−325).

Feature-Maps pro Convolutional Layer: Bei allen Convolutional Layer kann die Anzahl der Feature-Maps vom Benutzer ausgewählt werden. Wie bereits in Kapitel 3.3.2 beschrieben, entscheidet die Anzahl der Feature-Maps, wie viele Filter im Modell berechnet werden. Die Anzahl kann von ein paar wenigen bis zu über tausend variieren. Die Anzahl ist dabei im Wesentlichen auf den verfügbaren Videospeicher begrenzt. Sie muss jedoch auch mit Vorsicht gewählt und an die Komplexität der Daten angepasst werden. Je komplexer die Aufgabenstellung ist, desto mehr Feature-Maps können und sollten verwendet werden (vgl. PLANCHE und ANDRES 2019: 82-83, GÉRON 2019: 325).

Output-Neuronen: Die Anzahl der Output-Neuronen ist davon abhängig, wie viele Klassen bestimmt werden sollen. Bei einer binären Klassifikation sind zwei Output-Neuronen zu wählen. Wenn mehrere Klassen klassifiziert werden sollen, werden so viele Neuronen benötigt, wie es auch Klassen im Modell gibt (vgl. GÉRON 2019: 325).

Aktivierungsfunktionen: Die Aktivierungsfunktion ist besonders nach der Convolution wichtig (siehe Kapitel 3.3.2). Grundsätzlich kann hierzu eine beliebige mathematische lineare oder nichtlineare Funktion benutzt werden. In der Regel werden die Aktivierungsfunktionen jedoch auf Basis der aktuellsten Forschungsarbeiten ausgewählt.

Folgende wichtige nichtlineare Aktivierungsfunktionen werden zurzeit bei modernen neuronalen Netzwerken verwendet:

- **Sigmoid-Funktion**: Die Sigmoid-Funktion erlaubt ausschließlich Werte zwischen 0 und 1, wodurch die Werte für jedes Neuron normalisiert werden.

  $$f(x) = \frac{1}{1 + e^{-x}}$$

  Wenn die Werte auf der x-Achse sehr hoch oder niedrig sind, kann es passieren, dass das CNN aufhört zu lernen. Dies ist darauf zurückzuführen, dass die Funktionskurve bei hohen und niedrigen x-Werten sehr flach wird. Dadurch kommt es zu keiner Verbesserung bei der Vorhersage und das Modell kann nicht weiterlernen (vgl. NWANKPA et al. 2018: 5, SHARMA 2017).
- **TanH-Funktion:** Die TanH-Funktion ist der Sigmoid-Funktion sehr ähnlich. Der Unterschied liegt darin, dass sich auch hohe und niedrige Werte auf der x-Achse positiv auf das Modell auswirken (vgl. NWANKPA et al. 2018: 7, SHARMA 2017).

  $$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$
- **ReLU (Rectified Linear Unit):** Die ReLU-Funktion zählt zu den meistgenutzten und modernsten Aktivierungsfunktionen im Bereich der Computer-Vision. Im Gegensatz zu den anderen nichtlinearen Funktionen kann sie leicht berechnet werden, wodurch die Rechenzeit stark verkürzt wird (vgl. NWANKPA et al. 2018: 8−9, SHARMA 2017).

  $$f(x) = \max(0, x)$$

Ein Problem der ReLU-Funktion kann sein, dass alle negativen Werte auf der x-Achse automatisch zu einer 0 werden. Dadurch können negative Werte nicht erfasst werden und es kann zum ‚Absterben' eines Neurons kommen (‚dying ReLU'). Ein solches ‚abgestorbenes' Neuron kann nicht wieder aktiviert werden (vgl. NWANKPA et al. 2018: 8−9, SHARMA 2017).

- **Leaky ReLU:** Die Leaky-ReLU-Funktion ist der ReLU-Funktion sehr ähnlich. Im Gegensatz zur ReLU-Funktion nimmt sie bei negativen Werten jedoch nicht 0 an, sondern fällt leicht ab. Somit sind auch negative Werte möglich, wodurch Neuronen nicht absterben können (vgl. NWANKPA et al. 2018: 9, SHARMA 2017).

Learning-Rate: Die Lernrate ist einer der wichtigsten Hyperparameter in einem CNN. Die Herausforderung bei der Lernrate ist, dass die richtige Geschwindigkeit für die Daten gefunden wird. Wenn die Lernrate viel zu hoch angesetzt wird, kann die Fehlerrückführung nicht korrekt durchgeführt werden, wodurch der Loss bzw. der Fehler sehr groß wird. Bei einer nur leicht erhöhten Lernrate lernt das Modell zu Beginn zwar schnell, jedoch erreicht es nie einen optimalen Zustand. Bei einer zu niedrig angesetzten Geschwindigkeit kann das Modell zwar die Parameter lernen und anpassen, jedoch dauert dies wesentlich länger. Daher gilt es, die optimale Lernrate zu finden. Dazu gibt es mehrere Methoden, die in Kapitel 3.8.2 vorgestellt werden (vgl. GÉRON 2019: 325−326).

3.6. Herausforderungen bei Convolutional Neural Networks

Beim Training von CNNs müssen wesentliche Grundsätze beachtet werden, da es sonst schnell zu einem ineffizienten oder fehlerhaften Training kommen kann. Viele dieser Probleme können mithilfe von Validierungsdaten rasch erkannt werden. Dazu werden bereits vor dem Training des Modells zunächst 20 % der Trainingsdaten entnommen und nicht für den Trainingsvorgang verwendet (siehe Kapitel 3.3.1). Zu folgenden Problemen kann es laut GÉRON (2019: 23−32) und PLANCHE und ANDRES (2019: 49) beim Training eines CNNs kommen:

- **Ungenügende Anzahl an Trainingsdaten:** Zum Trainieren eines CNNs wird eine große Anzahl an Daten benötigt. Je komplexer das Problem ist, desto mehr Trainingsdaten werden benötigt. Bereits für das Lösen von einfachen Problemen werden mehr als tausend Beispiele benötigt (vgl. GÉRON 2019: 23).
- **Nichtrepräsentative Trainingsdaten:** Für das Training müssen Daten verwendet werden, die das Problem repräsentieren. Dazu muss für jede Klasse eine große Anzahl von Beispielen vorhanden sein und zudem ist es notwendig, dass die Erfassungsmethode für die Beispiele möglichst genau ist.
  Ebenso sollte es vermieden werden, dass es zu einer unausgeglichenen Verteilung von verschiedenen Klassen kommt. Die Anzahl der Beispiele pro Klasse sollte im Datensatz somit gleichmäßig repräsentiert sein. In den meisten Fällen weisen Datensätzen ein solches Problem auf (*Data imbalance problem*), das jedoch durch ausgewählte Loss-Funktionen und Gewichtungsmethoden ausgeglichen werden kann (vgl. AURELIO et al. 2019: 1).
- **Schlechte Datenqualität:** Wenn in den Trainingsdaten Fehler auftreten oder viele Ausreißer vorhanden sind, ist es dem Modell nicht möglich, die dahinterstehenden

Muster zu erlernen. Dies führt nach dem Training zu einer ungenauen oder falschen Klassifikation der Daten. Das Problem kann durch die manuelle Kontrolle und Bereinigung der Daten behoben werden oder durch eine sehr genaue manuelle Klassifikation der Trainingsbeispiele vor dem eigentlichen Training (vgl. GÉRON 2019: 26).

- **Overfitting:** Beim Overfitting kommt es zu einer Übergeneralisierung der Trainingsdaten. Das Modell funktioniert ausgezeichnet für die Trainingsdaten, kann bei unbekannten Daten jedoch nicht mehr generalisieren und macht falsche Vorhersagen zu den Klassenzugehörigkeiten. Ein Overfitting entsteht, wenn das Modell zu komplex, relativ zu der Menge und der Qualität der Daten gesehen, ist. Ein solcher Sachverhalt kann durch die Auswahl eines besser geeigneten Modells mit weniger Parametern verhindert werden. Andere Methoden sind Regulierungsmethoden im Modell (siehe Kapitel 3.3.4) oder die Beschaffung von zusätzlichen Daten (vgl. PLANCHE und ANDRES 2019: 49).
  Damit ein Overfitting rechtzeitig erkannt werden kann, müssen die Genauigkeit und der Fehler der Validierungsdaten herangezogen werden, die nach jeder Epoche berechnet werden. Steigen diese beiden Werte bei der Validierung plötzlich stark an (große Fehler bei der Klassifizierung) und verbessern sich die Werte bei den Trainingsdaten trotzdem kontinuierlich, kann von einem Overfitting ausgegangen werden. Das Training sollte an diesen Wendepunkt vorzeitig beendet werden, da das Modell dort seinen Idealzustand erreicht hat. Dieser Vorgang wird auch als ‚Early-Stopping' bezeichnet und erspart dem Nutzer übermäßige Rechenzeit, in der keine Verbesserung des Modells mehr stattfinden würde (vgl. PLANCHE und ANDRES 2019: 49).
- **Underfitting:** Das Underfitting ist das Gegenteil des Overfittings. Das Modell ist also für die Daten zu einfach aufgebaut und kann den komplexen Sachverhalt nicht erlernen. Dies kann ausschließlich durch die gezieltere Auswahl von passenden Daten oder durch die Erstellung eines komplexeren Modells verhindert werden. Es kann ebenso mit der Beobachtung des Loss-Verlaufes der Trainingsdaten und der Validierungsdaten erkannt werden. Bei einem Underfitting ist der Loss der Validierungsdaten im Vergleich zu den Trainingsdaten sehr hoch und sinkt nur schwach oder gar nicht ab (vgl. PLANCHE und ANDRES 2019: 49).

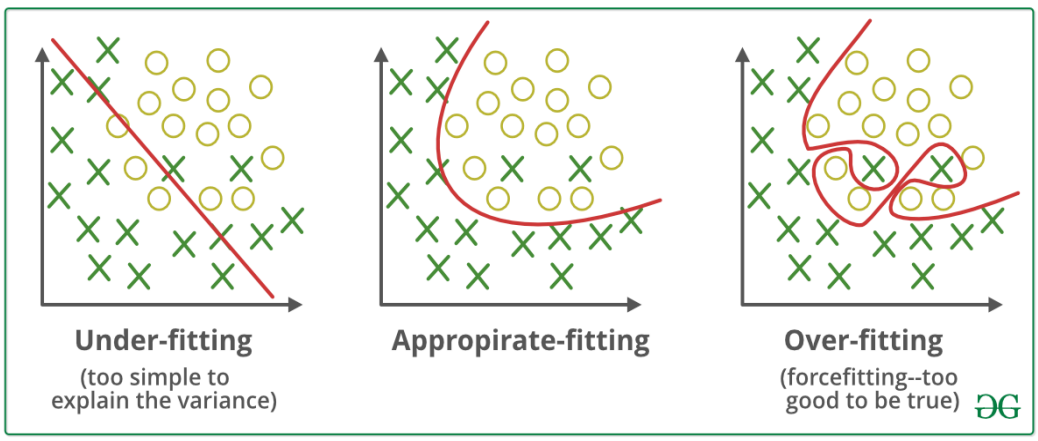


Abbildung 12: Beispiele für Underfitting, ein gutes Modell und ein Overfitting (GeeksforGeeks 2017)

- **Vanishing- oder Exploding-Gradients:** Die Fehlerrückrechnung und die Anpassung der Parameter durch das Gradientenverfahren (siehe Kapitel 3.3.2) sind eine unumgängliche Methode, ohne die ein neuronales Netzwerk nicht funktionieren kann.
  Selten kann es passieren, dass die Gradienten in den tief liegenden Zwischenschichten zu niedrig (Vanishing Gradients) oder zu hoch (Exploding Gradients) werden. Dies geschieht vor allem bei Aktivierungsfunktionen wie bei der Sigmoid-Funktion, die sich bei hohen und niedrigen Werten stark abflachen und nur mehr Aktivierungswerte nahe 0 oder 1 erzeugen (vgl. HANIN 2018: 1, KRÄHENBÜHL et al. 2015: 1−2).
  Damit dieses Problem nicht entsteht, kann eine Funktion benutzt werden, die nicht ‚gesättigt' werden kann. Ein Beispiel dafür wäre die ReLU-Funktion, die im positiven Bereich offen ist. Die negativen Effekte der ReLU-Funktion wurden bereits in Kapitel 3.5 beschrieben. Statt der ReLU-Funktion kann auch die LeakyReLU-Funktion herangezogen werden, die die negativen Aspekte der ReLU-Funktion ausgleichen soll (vgl. NWANKPA et al. 2018: 9).

3.7. Bekannte CNN-Modelle

In der Regel besteht ein klassisches CNN-Modell aus einer Kombination von Convolutional Layers, Pooling Layers und anderen wichtigen Zwischenschichten. Diese Reihenfolge kann beliebig lange wiederholt werden (siehe Kapitel 3.3). Dadurch wird das CNN kontinuierlich tiefer, weshalb ein solches Verfahren auch als Deep-Learning-Methode bezeichnet wird. Am Ende dieser Abfolgen folgt ein herkömmliches neuronales Netzwerk (Fully connected layer), das für die endgültige Klassifikation verantwortlich ist (siehe Kapitel 3.3.4). Abhängig von der finalen Klassifizierung (Binär oder Multiklassen) muss hier eine Sigmoid-Aktivierung (Binär) oder eine Softmax-Aktivierung (Multiklasse) angewendet werden. Ein solches Modell kann schnell und problemlos selbst programmiert werden.

Im Gegensatz zu diesen einfachen Modellen stehen die komplexen und großen Modelle. In den letzten zehn Jahren ist eine Vielzahl von unterschiedlichen äußerst performanten und umfangreichen Modellen entwickelt worden, die jedoch alle auf dieser grundlegenden Variante aufbauen. Die Performance dieser komplexen Modelle wird in der jährlich stattfindenden ImageNet-Challenge gemessen. Dabei werden die sogenannte Top-Eins-Fehlerrate und die Top-Fünf-Fehlerrate gemessen. Die Top-Eins-Fehlerrate gibt die Anzahl der Bilder an, die nicht richtig klassifiziert wurden. Die Top-Fünf-Fehlerrate hingegen bezeichnet die Anzahl der Bilder, bei denen die richtige Antwort nicht in den Top-Fünf-Vorhersagen war (vgl. RUSSAKOVSKY et al. 2013: 1−4).

Aufgrund der teilweise hochkomplexen Architekturen wird auf eine genaue Erklärung der Modelle verzichtet, da dies den Umfang dieser Arbeit überschreiten würde. Folgende wichtige Modelle gibt es:

- **LeNet-5:** Das Modell wurde erstmals 1998 von LECUN et al. (1998) veröffentlicht und war eines der ersten komplexeren Modelle. Es wurde benutzt, damit Zahlen

auf Schecks automatisch erkannt werden können. Es handelt sich heute um ein sehr einfaches Modell, war jedoch das erste seiner Art (vgl. PECHYONKIN 2018).

- **AlexNet:** AlexNet wurde 2012 von KRIZHEVSKY et al. (2012) veröffentlicht. Mit einer Top-Fünf-Fehlerrate von 17 % hatte das Modell im Jahr 2012 die bis dahin beste Performance. Bis heute wird dieses Modell auch als Grundstein der aktuellen CNNs gesehen (vgl. ALOM et al. 2018: 11).
- **GoogLeNet:** Das Modell wurde bei Google Research von SZEGEDY et al. (2015) veröffentlicht und hat eine Top-Fünf-Fehlerrate unter 7 %. Das gute Ergebnis ist darauf zurückzuführen, dass das Modell wesentlich tiefer als vorherige Modelle war (vgl. ALOM et al. 2018: 12).
- **ResNet:** ResNet wurde von HE et al. im Jahr 2015 veröffentlicht und hat eine Top-Fünf-Fehlerrate von unter 3,6 %. Das umfangreichste ResNet besitzt 152 Zwischenschichten, wobei es auch Varianten mit weniger Schichten gibt. Es gehört bis heute zu den akkuratesten Modellen im Bereich der CNNs (vgl. ALOM et al. 2018: 12−13). Da das ResNet zu einer der besten Modellvarianten zählt, wird es auch im Zuge dieser Masterarbeit verwendet. Eine genauere Erklärung des ResNet ist in Kapitel 4.4.1 zu finden.

Es gibt noch viele andere bekannte und gut funktionierende Modelle. Jedes davon hat etwaige Vor- und Nachteile. Die Vorstellung all dieser Modelle ist jedoch von keiner großen Relevanz für diese Masterarbeit.

3.8. Implementierung eines Convolutional Neural Networks

Für die Implementation eines CNNs, inklusive den notwendigen Schritten und Methoden, können verschiedenen Programmbibliotheken in Python benutzt werden. Die bekannteste, die auch für den praktischen Teil der vorliegenden Masterarbeit benutzt wird, ist *TensorFlow* von Google. Andere bekannte Bibliotheken, die ähnlichen Funktionen wie *TensorFlow* besitzen, sind *PyTorch* von Facebook oder *Caffe*.

3.8.1. TensorFlow

TensorFlow wurde im Jahr 2015 vom Google-Brain-Team der Öffentlichkeit präsentiert und anschließend unter einer Open-Source-Lizenz (Apache 2.0) veröffentlicht. Bei *TensorFlow* handelt es sich um ein Framework, also um Programmiergerüst, das den Nutzer bei der Implementierung einer künstlichen Intelligenz unterstützen soll. Die Programmbibliothek ist insbesondere für den Einsatz im Bereich des maschinellen Lernens geeignet und speziell dafür programmiert worden. *TensorFlow* kann nahezu alle numerischen Operationen auf der Grafikkarte ausführen, die dort wesentlich schneller berechnet werden können, als es auf dem Hauptprozessor möglich ist. Mittlerweile wird *TensorFlow* von unzähligen, zum Teil auch bekannten Projekten, zum Beispiel Google Search, Gmail oder Google Photos, verwendet. Da viele kommerzielle und nichtkommerzielle Projekte wie auch wissenschaftliche Projekte auf *TensorFlow* setzen, hat es sich zum populärsten Machine-Learning-Framework auf GitHub entwickelt und wird von vielen Entwicklern unterstützt (vgl. ABADI et al. 2016a: 265−266).

Die Grundeinheit in *TensorFlow* sind Tensoren, die von einer numerischen Operation zur nächsten übergeben werden (daher auch der Name ‚*TensorFlow*'). Tensoren können wie eine multidimensionale Matrize aufgebaut sein, aber auch nur einfache einzelne Werte enthalten. Der Umgang mit den Tensoren und den ebenso dazugehörigen Tensoren-Operationen sind dabei die Kernfeatures von *TensorFlow* (vgl. ADABI et al. 2016b: 2).

Zusätzlich zu den Kernfunktionen wurde in *TensorFlow* die Programmbibliothek *Keras* implementiert. Ursprünglich handelt es sich dabei um eine von *TensorFlow* unabhängige Programmbibliothek, die für die Programmierung von neuronalen Netzwerken verwendet wird. Mittlerweile wurde *Keras* von den Entwicklern komplett in *TensorFlow* eingearbeitet und muss nicht mehr als ergänzende Programmbibliothek in Python installiert werden. Die Weiterentwicklung von *Keras* wird mittlerweile auch von den Entwicklern bei Google durchgeführt (vgl. ATIENZA 2020: 2−3).

Mit *Tensorflow* kann ein komplettes und umfassendes Machine-Learning-Projekt aufgesetzt werden. Ein solches umfasst unter anderem das Einlesen und Vorprozessieren von Daten (tf.data), das Bearbeiten von Bildern (tf.image) und das anschließende Training auf Basis dieser Daten (tf.keras). Bevor das eigentliche Modell trainiert werden kann, müssen einige Schritte wie die Erstellung der Daten mit den passenden Labels und die Erstellung des Modells durchgeführt werden. Optional können die Daten auch in das *TensorFlow*-eigene *TFRecords*-Format umgewandelt werden, was die Nutzung beschleunigt. Eine genaue Erklärung zu den *TFRecords* ist in Kapitel 4.3.4 zu finden. Die Erstellung eines Modells wird in Kapitel 4.4 erläutert.

Bevor ein Modell festgelegt wird, müssen jedoch passende Hyperparameter (siehe Kapitel 3.5) gefunden werden. Dies wird mit einer Hyperparametersuche durchgeführt, die im folgenden Kapitel kurz vorgestellt wird.

3.8.2. Hyperparametersuche in TensorFlow

Einerseits kann der Benutzer mehrere Modellvarianten manuell durchprobieren und die Ergebnisse miteinander vergleichen. Die bessere und einfachere Methode, passende Hyperparameter zu finden, ist die halbautomatische Suche (vgl. GÉRON 2019: 320−322). Für diese stehen drei wichtige Methoden zur Verfügung:

- **Grid-Search:** Bei dieser Methode wird ein Merkmalsraum mit verschiedenen möglichen Hyperparametern bestimmt. Anschließend werden alle möglichen Kombinationen aus dem Set an den Daten ausprobiert. Je nach der Anzahl der Kombinationen dauert der Vorgang unterschiedlich lange. Am Ende kann das Modell mit jenen Hyperparametern genommen werden, das den geringsten Loss bei der Validierung hat (vgl. SAMALA et al. 2016: 2−3).
- **Random-Search:** Die Grid-Search-Methode funktioniert, wenn nur wenige Kombinationen von Hyperparametern ausgetestet werden müssen. Sollen hingegen mehrere tausend Kombinationen überprüft werden, sollte die Random-Search-Methode herangezogen werden. Dabei wird auch ein Merkmalsraum mit Hyperparametern angegeben und es wird auch bestimmt, wie viele Iterationen nach passenden Parametern gesucht werden soll. Die Werte werden so oft wie

angegeben zufällig miteinander kombiniert und ausprobiert. Im Normalfall wird dafür zu Beginn ein weiter Spielraum mit großen Abständen zwischen den jeweiligen Werten gewählt (z. B. ein großer Spielraum bei der Lernrate mit einer großen Schrittgröße zwischen den auszutestenden Werten). Nach dem ersten Durchlauf können die Ergebnisse analysiert werden und der mögliche Spielraum kann eingeengt werden (vgl. CAPONETTO 2019).
Die Random-Search-Methode ist im Normallfall äußerst zeitintensiv, da mit der Methode viele Merkmalskombinationen ausgetestet werden. Sie bietet jedoch den Vorteil, dass ein großer Merkmalsraum untersucht werden kann, der zunehmend verfeinert wird und am Ende ein optimales Ergebnis liefert (vgl. CAPONETTO 2019).

- **Bayes'sche Optimierung:** Auch bei dieser Methode werden zufällige Merkmalskombinationen ausgetestet. Im Gegensatz zu den anderen beiden Methoden wird das Ergebnis jeder Berechnung evaluiert und anschließend werden die Hyperparameter der vorherigen Berechnung anhand der Ergebnisse der vorherigen Evaluierung optimiert. Das geschieht über eine Wahrscheinlichkeitsfunktion, die versucht, dass ein bestimmtes Ziel optimiert wird (z. B. der Loss soll minimiert werden) (vgl. KOEHRSEN 2018).
  Durch die Evaluierung jedes Rechenschrittes werden die Hyperparameter kontinuierlich optimiert, bis diese einen idealen Zustand erreichen. Im Normallfall benötigt die Bayes'sche Optimierung weniger Zeit als die Random-Methode und findet zudem auch bessere Hyperparameter (vgl. KOEHRSEN 2018).

Nachdem die geeigneten Hyperparameter gefunden wurden, kann mit dem eigentlichen Training begonnen werden. Dabei werden jene Hyperparameter verwendet, die bei der Suche gefunden worden sind.

Die Frage, wie die Daten beschaffen sein müssen, wie sie in *TensorFlow* eingelesen werden können und wie ein Modell erstellt wird, soll im nachfolgenden Kapitel zur Methodik erläutert werden.

4. Methodik

Das folgende Kapitel wird sich ausführlich mit der Methodik dieser Masterarbeit beschäftigen. Ziel ist es, die Orthofotos nach deren Bodenbedeckungsklassen zu klassifizieren. Dies soll mithilfe eines CNNs ermöglicht werden, das jedoch zunächst auf die neuen Daten trainiert werden muss. Bevor mit dem eigentlichen Training und der anschließenden Klassifikation der unbekannten Orthofotos begonnen werden kann, sind diverse Zwischenschritte notwendig.

Am Beginn dieses Kapitels werden zunächst die Datengrundlagen beschrieben, die für die Trainingsdaten verwendet werden. Bei diesen handelt es sich um Orthofotos und Geländemodelle, die anfänglich nach deren Bodenbedeckung klassifiziert werden müssen. Die dazu gewählten Klassen werden im anschließenden Unterkapitel beschrieben und im Detail erläutert. Im darauffolgenden Kapitel wird auf die Nachprozessierung (*Postprocessing*) dieser Daten in ArcGIS und Python eingegangen. Diese Schritte sind insbesondere für die Performance von *TensorFlow* und auch für die finale Genauigkeit des Modells von großer Bedeutung. Das anschließende Unterkapitel beschreibt die Findung der Hyperparameter, die essenziell für das Training sind. Abschließend wird der eigentliche Trainingsvorgang mit den zuvor erstellten Daten dargestellt.

Die notwendige manuelle Klassifizierung der Bodenbedeckung auf Basis der Orthofotos erfolgt mithilfe der Software ArcGIS Pro von der Firma ESRI. Auch das Exportieren der Trainingsdaten wird mithilfe von ArcGIS Pro durchgeführt. Die anschließende Nachprozessierung dieser Daten erfolgt hingegen ausschließlich in Python, wobei die dazugehörigen Skripte im Anhang zu finden sind. Dabei handelt es sich um mehrere Python-Skripte, die jeweils für unterschiedliche Einsatzzwecke konzipiert wurden. Auch die Findung der Hyperparameter und das anschließende Training wurden mithilfe von Python durchgeführt. Die Skripte dazu sind ebenfalls im Anhang dieser Masterarbeit zu finden.

Da das CNN am Ende zur Klassifikation des gesamten österreichischen Bundesgebietes dienen soll, ist dies auch das eigentliche Untersuchungsgebiet. Da eine solche Klassifikation jedoch zu viel Zeit in Anspruch nehmen würde, werden vier zufällig ausgewählte Orthofoto-Blätter verwendet. Dabei soll geprüft werden, ob das CNN sowohl dicht besiedeltes Gebiet als auch ländliche Bereiche klassifizieren kann.

Der gesamte Workflow eines Deep-Learning-Projektes ist in der nachfolgenden Abbildung (siehe Abbildung 13) dargestellt. Die gelben Felder sind keine obligatorischen Schritte eines solchen Projektes, jedoch werden sie im Normalfall empfohlen. Diese beschleunigen den Trainingsvorgang und verbessern die Ergebnisse. Die grünen Felder hingegen können nicht übersprungen werden.

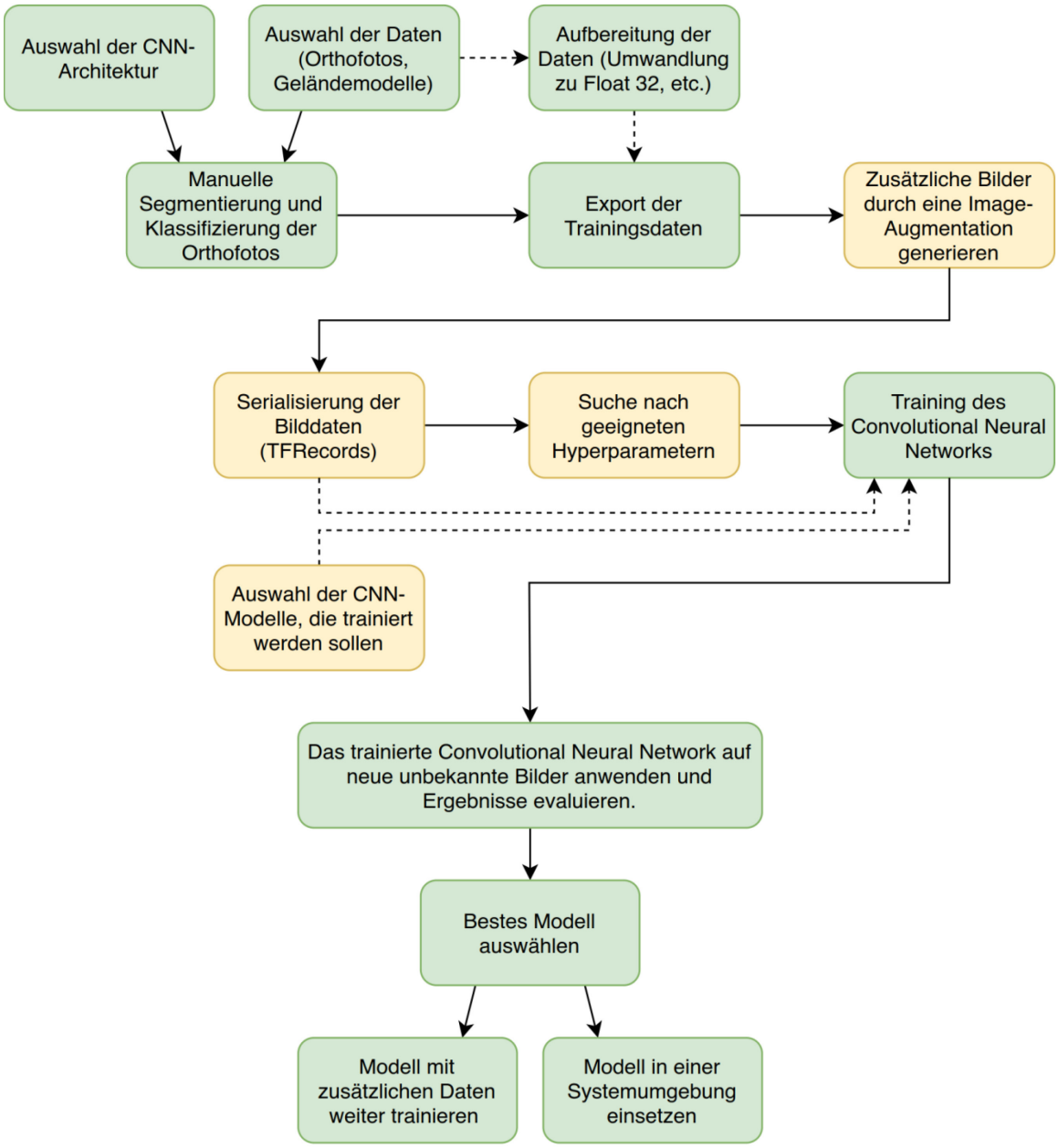


Abbildung 13: Workflow eines Deep-Learning-Projektes (eigene Erstellung)

4.1. Datengrundlage

Alle Daten, die für das Training des CNNs verwendet wurden, sind vom österreichischen BEV bereitgestellt worden. Sie sollen in den folgenden Unterkapiteln ausführlich beschrieben werden.

4.1.1. Orthofotos

Die wichtigste Grundlage für das Training des CNNs sind Orthofotos. Sie wurden für diese Masterarbeit vom BEV bereitgestellt und können dafür verwendet werden. Dabei handelt es sich um verzerrungsfreie und maßstabsgetreue Abbildungen der Erdoberfläche, die für das gesamte österreichische Bundesgebiet zur Verfügung stehen. Im Gegensatz zu herkömmlichen Luftbildern sind bei Orthofotos in der Theorie keine Verzerrungen und Verschiebungen vorhanden. Solche Verzerrungen entstehen durch die fotografische

Zentralprojektion und durch Höhenunterschiede im Gelände. Mithilfe eines digitalen Geländemodells können solche Unregelmäßigkeiten herausgerechnet werden und es entsteht ein flächen- und winkeltreues Orthofoto. Bei hohen Gebäuden oder sichttoten Bereichen im Orthofoto kann es jedoch trotzdem zu unscharfen Bildteilen oder Kippeffekten kommt. Die Lagegenauigkeit der Orthofotos hängen vom dahinter liegenden Geländemodell ab. In flachen Gebieten wird eine Genauigkeit zwischen 0,5 und 1 Meter erreicht und im Gebirge kann es zu Abweichungen zwischen 2 und 5 Metern kommen (vgl. BEV 2019a, Spektrum 2001).

Das gesamte österreichische Bundesgebiet wird in einem dreijährigen Zyklus beflogen. Die Orthofotos werden nach dem Blattschnitt der österreichischen digitalen Katastralmappe (DKM-Blatt) aufgeteilt, wobei ein Kartenblatt eine Größe von 1250 × 1000 Meter hat. Die DKM-Blätter befinden sich jeweils in den Meridianstreifen M28, M31 und M34 und insgesamt sind 68 058 Blätter vorhanden (vgl. BEV o. J.: 4−5).

Die Orthofotos werden vom BEV als GeoTIFF mit einer durchschnittlichen Auflösung von 0,2 Meter pro Pixel bereitgestellt. Zusätzlich zu den roten, grünen und blauen Farbkanälen (RGB) steht ein Infrarotkanal zur Verfügung. Dieser kann zum Beispiel für die Berechnung eines normalisierten Vegetationsindexes (NDVI) oder für die Erstellung eines Falschfarbenbildes verwendet werden. Solche Derivate von Orthofotos könnten in der Theorie auch für das Training des CNNs benutzt werden, im Fall dieser Masterarbeit wurde jedoch darauf verzichtet (vgl. BEV 2019a).

Die Orthofotos haben eine Bittiefe von 16-Bit (65 535 Farbwerte pro Kanal) und der Datentyp ist ein *unsigned integer* (Farbwerte im Bereich zwischen 0 bis 65 535). Damit die Daten von *TensorFlow* verwendet werden können, müssen die Farbwerte der Bilder zunächst in Gleitkommazahlen mit einer einfachen Genauigkeit (32-Bit Floating-Point bzw. Float-32) umgewandelt und gespeichert werden.

Für zwei Regionen wurden zufällig ausgewählte DKM-Blätter verwendet. Ein DKM-Blatt hat eine Größe von 1250 × 1000 Meter. Nur für den Bereich um Mariazell wurde ein gesamtes ÖLK-Blatt (österreichische Luftbildkarte) klassifiziert, welches eine Ausdehnung von 5000 × 5000 Meter hat. Die manuelle Segmentierung und Klassifikation von einem ÖLK-Blatt dauert in etwa drei bis vier Wochen, weshalb im Anschluss nur mehr ausgewählte DKM-Blätter verwendet wurden. Bei der Auswahl der Orthofotos wurde darauf geachtet, dass alle im Convolutional Neural Network verwendeten Klassen (siehe Tabelle 3) auf den Bildern vorkommen. Dadurch soll verhindert werden, dass ausschließlich ein paar wenige Klasse (z. B.: Nadel- und Laubwald) überrepräsentiert sind. Folgende Orthofotos wurden zum Training des Modells verwendet:

Tabelle 2: Verwendete DKM-Blätter (eigene Erstellung)

DKM-Blattnummer	Befliegungsjahr	Region	Meridianstreifen
4727-01	2018	Gosau, Hallstatt und Dachstein	M-31
4727-12			
4727-35			
4727-38			
6631-41 bis 6631-44	2017	Mariazell	M-34
6631-49 bis 6631-52			
6631-57 bis 6631-50			
6631-65 bis 6631-68			
6631-73 bis 6631-76			
7635-17	2018	Wien und Wienerwald	M-34
7635-19			
7535-51			

4.1.2. Geländemodelle

Auch die Geländemodelle werden für ganz Österreich vom BEV bereitgestellt. Dazu stehen digitale Oberflächenmodelle und digitale Höhenmodelle zur Verfügung. Beide Geländemodellvarianten besitzen eine Auflösung von 0,5 Metern pro Pixel und werden ebenso als GeoTIFF bereitgestellt. Bei den Geländemodellen ist keine Umwandlung in Gleitkommazahlen notwendig, da diese bereits in dem für *TensorFlow* richtigen Format vorliegen.

Das digitale Oberflächenmodell wird durch eine automatische Bildkorrelation (*Image Matching*) abgeleitet. Dadurch kann es insbesondere bei filigranen oder hohen Objekten zu Ungenauigkeiten kommen. Auch bei Gewässern und bewegten Objekten können Probleme auftreten. Zu einem großen Teil besitzt das digitale Oberflächenmodell jedoch eine Höhengenauigkeit von ungefähr einem Meter. Das digitale Geländemodell hingegen wird auf Basis von Laserscanning-Daten der verschiedenen Bundesländer erstellt. Auch für diese Daten gilt eine ähnliche Genauigkeit, wobei es insbesondere im Hochgebirge zu stärkeren Abweichungen von bis zu zehn Metern kommen kann (vgl. BEV 2019b, BEV 2019c).

4.1.3. Digitales Landschaftsmodell (DLM)

Auch das digitale Landschaftsmodell (DLM) des BEV steht als Datengrundlage zur Verfügung. Es handelt sich dabei um ein lagerichtiges Abbild der Natur, das die Objekte und die dazugehörigen Informationen in einem Vektorformat abbildet. Derzeit werden sieben Objektbereiche im digitalen Landschaftsmodell geführt: Verkehr, Siedlung, Gebietsnutzung, Gewässer, Bodenbedeckung, Gelände und Namen. Für das digitale Landschaftsmodell stehen verschiedene Datenquellen zur Verfügung. Diese umfassen

unter anderem die digitalen Orthofotos, die digitale Katastralmappe, GPS-Vermessungen und auch externe Informationsquellen wie die Gemeinden. Die Genauigkeit des DLMs ist von der Erfassungsmethode abhängig, sie beträgt im Schnitt jedoch in etwa drei Meter (vgl. BEV 2012: 4−6).

Viele Objektbereiche liegen bereits für das gesamte Bundesgebiet vor und werden auch laufend aktualisiert. Manche Objektbereiche wie die Bodenbedeckung sind noch nicht komplett erfasst und stehen in gewissen Gebieten noch nicht zur Verfügung. Je nach Objektbereich werden die Daten als Polygone (z. B. die Bodenbedeckung, Gewässer), Linien (z. B. Verkehr) oder Punkte (z. B. Namen) bereitgestellt (vgl. BEV 2012: 6−14).

4.2. Klassifikation und Export der Trainingsdaten

Aktuell verwendet das BEV eine objektbasierte Methode zur Ableitung von Bodenbedeckungsinformationen. Damit werden zurzeit die Klassen ‚Gebäude', ‚Wald', ‚Buschwerk', ‚niedere Vegetation', ‚Gewässer' und ‚vegetationslose Bodenflächen' abgeleitet (siehe Kapitel 2.1). Das Ziel dieser Masterarbeit ist es, dass die oben genannten Klassen mithilfe eines CNNs ergänzt oder auch weiter differenziert werden. Da im ersten Schritt die zu klassifizierenden Klassen festgelegt werden müssen, wurden für diese Masterarbeit folgende Klassen festgelegt:

Tabelle 3: Klassen des CNNs (eigene Erstellung)

Bodenbedeckungsklasse	Neuerung
Gebäude	Die Klassifizierung der Gebäude funktioniert mit dem aktuellen Algorithmus gut. Trotzdem wird diese Klasse in das CNN-Modell aufgenommen, da sich hiermit die Performance des CNNs gut evaluieren lässt, indem es mit den Ergebnissen des aktuellen Algorithmus verglichen wird.
Nadelwald	Alle großen und weitläufigen Nadelwälder werden dieser Klasse zugeordnet. Eindeutig erkennbare einzeln stehende Nadelbäume werden dieser Klasse ebenso zugeteilt.
Laubwald	Diese Klasse beinhaltet alle großen Laubwälder. Kleine einzeln stehende Bäume, die eindeutig als Laubbaum erkennbar sind, werden auch dieser Klasse zugeordnet.
Gewässer	Zurzeit erfolgt die Klassifikation von Gewässern unter der Einbeziehung des DLMs. In Zukunft soll das CNN diese Aufgabe übernehmen können, wodurch die Informationen aus dem DLM für die Ableitung der Bodenbedeckungsinformationen nicht mehr benötigt werden. Diese Klasse beinhaltet alle Flüsse, Seen, Bäche und auch Teiche.
Versiegelte Flächen	Diese Klasse beinhaltet alle versiegelten Flächen. Es werden somit auch versiegelte Parkplätze und andere versiegelte Flächen wie Häusereinfahrten dieser Klasse zugeordnet

Bodenbedeckungsklasse	Neuerung
Unversiegelte Flächen	Bei dieser Klasse werden, analog zu der vorherigen Klasse, alle unversiegelten Flächen klassifiziert.
Sonstiges	Diese Klasse beinhaltet alle anderen Klassen, die nicht für diese Masterarbeit relevant sind.

Das Softwarepaket ArcGIS Pro von ESRI bietet mittlerweile eine Vielzahl an Hilfswerkzeugen für Methoden im Bereich des Deep-Learnings an. Dies umfasst das Kennzeichnen von Daten mit Labels und den anschließenden Export dieser Daten bis hin zur Anwendung eines bereits trainierten Deep-Learning-Modells auf neue Daten innerhalb der Umgebung von ArcGIS Pro (vgl. ESRI o. J.).

Aufgrund der vielfältigen Möglichkeiten im Bereich des Deep-Learnings und der guten Performance in Bezug auf große Datenmengen wird für die Erstellung der Labels auf dieses Softwarepaket zurückgegriffen. Da sich die Orthofotos in unterschiedlichen Meridianstreifen befinden, müssen zunächst zwei Feature-Datasets mit den beiden Koordinatensystemen erstellt werden. Eine Transformation in ein Koordinatensystem für ganz Österreich, zum Beispiel Web Mercator Auxiliary Sphere (EPSG 3857) oder Austria Lambert (EPSG 31287), wäre grundsätzlich möglich. Dabei erfolgt jedoch immer eine Interpolation der Daten, wodurch es zu einer Veränderung kommt. Da eine Manipulation der Datengrundlage vermieden werden sollte und das Training des CNN mit den Originaldaten geschehen soll, werden die Koordinatensysteme der Orthofotos im jeweiligen Meridianstreifen für das Projekt übernommen. Für das CNN spielt das Koordinatensystem keine Rolle mehr und kann ignoriert werden.

Für die Erstellung der Segmentierungen und der Klassifikation muss in den jeweiligen Feature-Datasets jeweils ein Polygon-Feature erstellt werden. Die Attributtabellen dieser Features müssen eine bestimmte Formatierung aufweisen, da ArcGIS Pro diese sonst nicht für den späteren Export verwenden kann. Folgende Attribute werden deshalb benötigt:

Tabelle 4: Attributtabelle der Klassifizierungspolygone (eigene Erstellung)

Attributname	Datentyp	Beispiel für das Attribut
RED	Integer	128
GREEN	Integer	0
BLUE	Integer	0
SHAPE	Text	Polygon M
Classcode	Text	Code_0
Classname	Text	Gebaeude
Classvalue	Integer	0

Theoretisch wäre es möglich, die Gebäudepolygone aus dem Grenzkataster und dem Grundsteuerkataster zu übernehmen. In der Praxis sind die Abweichungen zwischen den

sichtbaren Dachtraufen auf den Orthofotos und den Gebäudepolygonen des Katasters zu groß, als dass diese ohne eine Kontrolle bzw. Überarbeitung übernommen werden könnten.

Wenn solche Fehler in die Trainingsdaten übernommen werden, könnte dies zu schlechteren oder falschen Ergebnissen beim Training des CNNs führen. Daher sollte für die Erstellung der Trainingsdaten auf keinen Fall eine automatische Klassifikation von Daten vorgenommen werden. Wenn eine solche automatische Klassifikation gewählt wird, müssen die Ergebnisse genauestens kontrolliert und überarbeitet werden. Um Fehler zu vermeiden, wurde für den praktischen Teil dieser Masterarbeit eine vollkommen manuelle Klassifikation der Orthofotos vorgenommen. Wie viel Zeit für das Einzeichnen der Polygone benötigt wird, hängt im Wesentlichen von der Komplexität des Bildinhaltes ab. Bei sehr komplexen Bildern mit vielen Details, zum Beispiel in dicht besiedelten Bereichen, beträgt die Dauer pro DKM-Blatt in etwa eine Woche. Für weniger dicht besiedelte Gebiete reichen im Normalfall drei Tage. Durch die manuelle Klassifizierung kann sichergestellt werden, dass wenige Fehler vorkommen und die Polygone lagegenau liegen.

Dieser Prozess wird für alle Klassen im Bild durchgeführt. Die Unterscheidung zwischen befestigten und unbefestigten Straßen erfolgt mithilfe der Straßenattribute des digitalen Landschaftsmodells, jedoch zum größten Teil auf visueller Basis. Auch die Unterscheidung bei den Waldtypen (Nadel- und Laubwald) findet ausschließlich auf visueller Basis statt, da es hierzu keine Daten gibt. Der Vorteil dabei liegt darin, dass sich Laub- und Nadelwald an der Form der Baumkronen in den meisten Fällen eindeutig voneinander unterscheiden lassen. Nur bei einzeln stehenden Bäumen ist es oft schwer, den richtigen Typ zu erkennen.

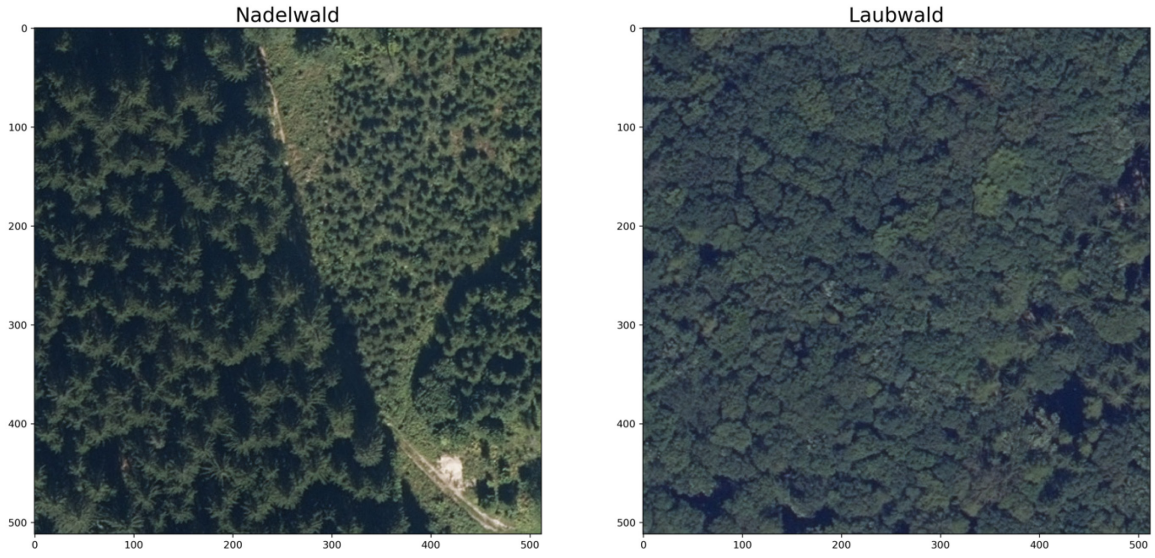


Abbildung 14: Optischer Vergleich zwischen Nadel- und Laubwald (Orthofotos BEV, eigene Erstellung)

In vielen Fällen ist eine eindeutige Abgrenzung zwischen den beiden Baumklassen zu einer anderen Klasse, zum Beispiel ‚Sonstiges', nicht oder nur schwer möglich. Dies ist oft der Fall, wenn sich die Wälder zum Rand hin ausdünnen und zunehmend einzelne

Bäume zu finden sind. Die Klassifikation dieser Bereiche ist zeitaufwendig und mit vielen Ungenauigkeiten verbunden. Insbesondere bei einzeln stehenden Bäumen ist die Erstellung solcher Polygone mit großen Ungenauigkeiten verbunden und kann oft nur geschätzt werden. Erschwerend kommt bei den Bäumen hinzu, dass diese oft von Bildstürzen im Orthofoto betroffen sind (siehe Abbildung 15). Dies ist auf das verwendete Geländemodell bei der Erstellung der Orthofotos zurückzuführen und kann insbesondere bei Bäumen nicht verhindert werden. Deshalb werden die Bereiche, in denen sich keine eindeutige Abgrenzung vornehmen lässt, der Klasse ‚Sonstiges' zugeordnet.

Abbildung 15: Schwierige Abgrenzung zwischen einzelnen Bäumen und anderen Klassen (Orthofotos BEV)

Bei manchen Klassen wurde zudem das digitale Landschaftsmodell zur Hilfe herangezogen. Die Gewässerpolygone können jedoch nicht direkt übernommen werden, da sie nicht genau genug sind und oft auch kleine Teile von Vegetation beinhalten. Die Daten des DLMs werden deshalb ausschließlich zur Findung von Gewässeroberflächen, zum Beispiel Seen, Teichen oder Flüssen, verwendet.

Auf diese Weise werden alle Klassen, die auf den gesamten Orthofotos vorkommen, digitalisiert. Dabei erhält jedes Polygon einen zuvor definierten und passenden Klassenwert (Attribut ‚Classvalue', siehe Tabelle 4). Folgende Klassenwerte wurden dafür ausgewählt:

Tabelle 5: Klassenwerte (Classvalues) der jeweiligen Klassen (eigene Erstellung)

Klasse	Klassenwert
Gebäude	0
Versiegelte Flächen	1
Unversiegelte Flächen	2
Gewässer	3

Klasse	Klassenwert
Nadelbäume	4
Laubbäume	5
Sonstiges	6

Nachdem alle Orthofotos auf diese Weise mit den Labels bzw. einer Maske versehen wurden, müssen die Polygone zu einem thematischen Rasterbild umgewandelt werden. Dies kann in ArcGIS Pro mit dem Werkzeug ‚Polygon in Raster' erfolgen. Dazu werden die Polygone als Input ausgewählt und das Feld, über das die Features eindeutig identifiziert werden können, sind die zuvor definierten Klassenwerte. Als Rasterzellengröße wird 0,2 gewählt, damit die Labels dieselbe Pixelgröße wie die Orthofotos besitzen. Bei den Labels handelt es sich um Graustufenbilder mit 256 Farbabstufungen (Unsigned Integer 8 – Uint8). Jede Graustufe repräsentiert eine Klasse, wobei nicht alle Farbabstufungen verwendet werden müssen. Bei mehr als 256 Klassen müsste ein anderes Format, zum Beispiel ein Unsigned Integer 16 (UInt16), verwendet werden, der 65 536 Graustufen ermöglicht (vgl. Geospatial Harris 2020).

Bevor der Export der Trainingsdaten vorgenommen werden kann, müssen die Orthofotos und die Geländemodelle mithilfe des ArcGIS-Werkzeugs ‚Bänder zusammensetzen' (‚Composite Bands') kombiniert werden. Dafür wird zunächst aus den beiden zur Verfügung stehenden Geländemodellen ein normalisiertes Geländemodell (nDSM) berechnet. Dieses wird aus der Differenz des digitalen Geländemodells und des digitalen Oberflächenmodells berechnet. Daraus ergeben sich die Höhen der Landschaftsobjekte, zum Beispiel jene von Gebäuden oder der Vegetation, vom Boden ausgehend. Theoretisch beträgt der Boden in einem normalisierten Geländemodell immer die Höhe Null. Durch Abweichungen zwischen dem digitalen Geländemodell und dem digitalen Oberflächenmodell ist dies normalerweise nicht der Fall. Die Abweichungen entstehen durch die verschiedenen Erfassungsarten der beiden verwendeten Geländemodelle. Insbesondere bei steilen Hängen und Gewässern kann es dadurch zu negativen Höhen oder auch zu extrem hohen Werten kommen. Die negativen Werte können mithilfe des ArcGIS-Tools ‚Reklassifizieren' (‚Reclassifiy') beseitigt werden. Die sehr hohen Werte hingegen können nicht verhindert und müssen verwendet werden.

Durch das Zusammensetzen der Bilder ergibt sich ein neues Bild im TIFF-Format, das sowohl die vier Kanäle des Orthofotos im Gleitkommaformat (Float 32) als auch einen fünften Kanal mit dem normalisierten Geländemodell beinhaltet. Dieser Vorgang wird für jedes zu klassifizierende Orthofoto aus Tabelle 2 wiederholt.

Auf Basis dieser neu erstellten Bilder und der soeben erstellten Labels können nun die eigentlichen Trainingsdaten exportiert werden. Dafür wird das ArcGIS-Werkzeug ‚Export Training Data for Deep Learning' verwendet. Dieses Tool konvertiert die maskierten Orthofotos in Trainingsdatensätze um. Dabei handelt es sich um kleine Ausschnitte des Orthofotos und des dazugehörigen Labels. Folgende Parameter müssen für das Werkzeug angegeben werden:

Tabelle 6: Erklärung der verwendeten Parameter des Werkzeuges ‚Export Training Data for Deep Learning' (eigene Erstellung)

Parameter	Erklärung
Input-Raster	Der Raster, der als Trainingsbasis dient. In dem Fall ist der Input das zuvor erstellte Bild (Orthofoto und nDSM).
Output-Ordner	Der Ordner, in dem die Trainingsdaten gespeichert werden sollen. In diesem Ordner werden dann Subfolder für die Bilder und die Labels erstellt.
Input class data	Der Raster mit den Labels. Es ist zwingend ein thematischer Raster notwendig (mit dem Tool ‚Raster Dataset Properties' festlegbar).
Tile-Size	Die Größe der Bildausschnitte. Die Größe kann beliebig groß sein, wird jedoch vom Arbeitsspeicher der Grafikkarte stark begrenzt. Je größer die Bildausschnitte sind, desto mehr Kontext kann in einem Bild vorhanden sein. Für diese Masterarbeit wurde eine Tile-Size von 256 × 256 verwendet
Image-Chip-Format	Das Dateiformat der Trainingsdaten. Es kann zwischen JPEG, PNG und TIFF ausgewählt werden. Wenn mehr als vier Kanäle verwendet werden sollen, muss das TIFF-Format ausgewählt werden.
Stride	Der Abstand zwischen den einzelnen Bildausschnitten. Damit kann festgelegt werden, ob und um wie viele Pixel sich die Bildausschnitte überlappen sollen. Für diese Masterarbeit wurde ein Stride von 128 Pixeln gewählt.
Metadata-Format	Bei diesem Punkt wird festgelegt, wie die Labels formatiert werden sollen. Es stehen mehrere Methoden zur Auswahl, die je nach der Klassifizierungsmethode ausgewählt werden müssen. Für die Segmentierung von Bildern muss das RCNN-Masks-Format ausgewählt werden.

Insgesamt entstehen auf diese Weise pro DKM-Blatt in etwa 5000 kleine Bildausschnitte und zusätzlich dazu passende Labels auf Basis der zuvor beschriebenen und erstellten Klassifizierungen. Diese Klassifizierung wird in den meisten Fällen als Segmentation oder Maske bezeichnet.

4.3. Preprocessing der Trainingsdaten

Theoretisch können die zuvor erstellten Daten bereits zum Training eines CNNs herangezogen werden. In der Praxis gibt es jedoch noch Methoden, um zusätzliche Daten zu generieren und diese anschließend in ein effizientes Format für *TensorFlow* umzuwandeln. Diese Methoden werden im folgenden Kapitel beschrieben. Das Python-Skript zum zählen der Pixel im Datensatz ist in Anhang I nachzulesen. Jenes für die Image-Augmentation ist in Anhang II zu finden und das für die Umwandlung in die *TFRecords*, ist in Anhang III zu finden.

4.3.1. Methoden der Image-Augmentation

Im Bereich des maschinellen Lernens stehen in vielen Fällen nur unzureichend viele Daten zur Verfügung. Ein ebenso relevantes Problem ist jenes der unausgewogenen Anzahl der im Datensatz vorhandenen Klassen. Bei beiden Problemstellungen kann mithilfe der Image-Augmentation gegengesteuert werden. Dabei werden aus den bestehenden Bildern und Masken neue abgewandelte Bilder generiert. Folgende Augmentations stehen laut SHIJIE et al. (2017: 4165−4166) für Bilder zur Verfügung:

- **Flip (Umdrehen, Wenden):** Bei einem Flip werden die Bilder entweder horizontal oder vertikal gewendet. Anzumerken ist, dass bei einem korrekt ausgeführten vertikalen Flip das Bild zunächst um 180° gedreht und anschließend horizontal umgedreht wird.

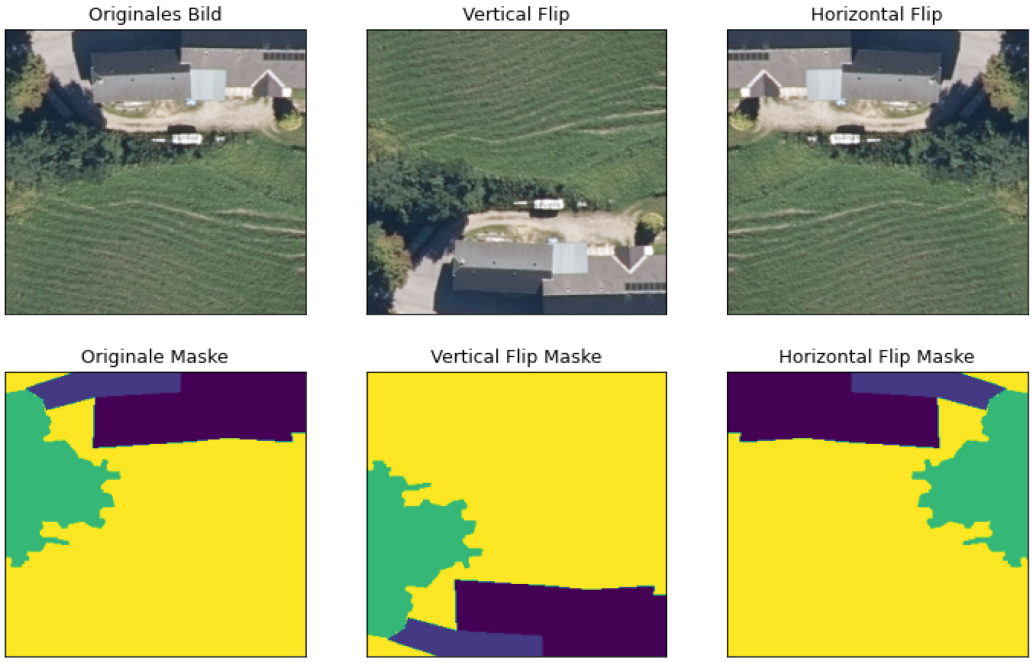


Abbildung 16: Beispiele für einen Flip (eigene Erstellung)

- **Rotation (Drehung, Rotation):** Bei der Rotation wird das Bild um einen bestimmten Winkel gedreht. Bei einer Drehung von 90° oder 180° können die ursprünglichen Bilddimensionen problemlos beibehalten werden. Bei einer Drehung um einen anderen Winkel müssen die nicht mehr vorhandenen Bildbereiche interpoliert werden. Dazu stehen verschiedenen Interpolationsmethoden zur Verfügung, die in Kapitel 4.3.2 beschrieben werden.

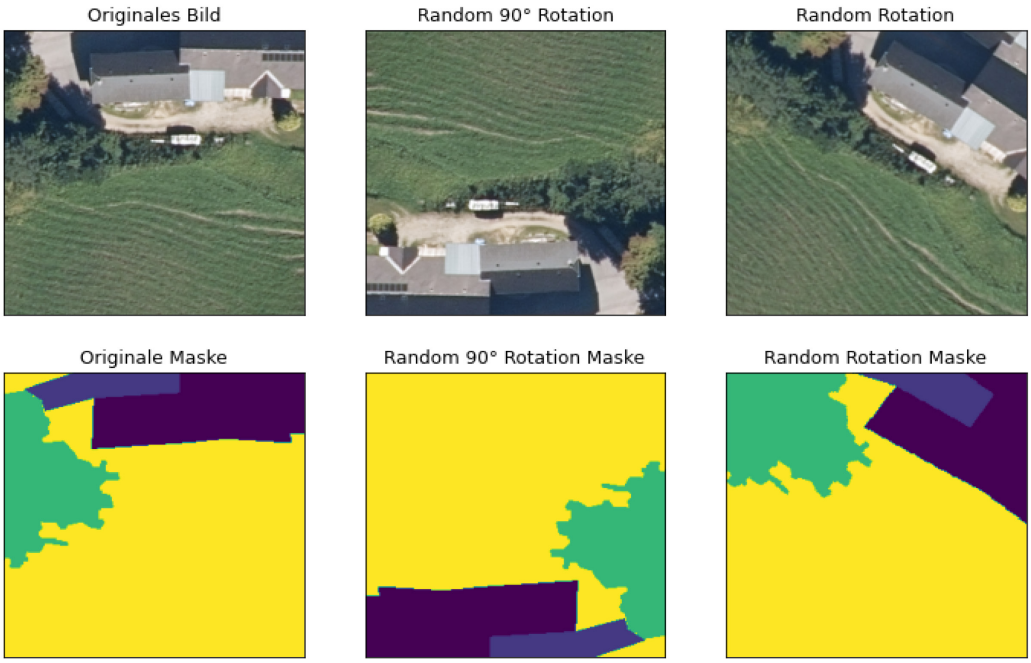


Abbildung 17: Beispiele für eine Rotation (eigene Erstellung)

- **Crop (Beschneiden):** Bei der Beschneidung der Bilder wird ein Teil des Bildes herausgeschnitten. Damit das Bild im Anschluss dieselbe Anzahl an Pixel wie zuvor hat, werden die Bilder zu einer gewünschten Größe interpoliert.

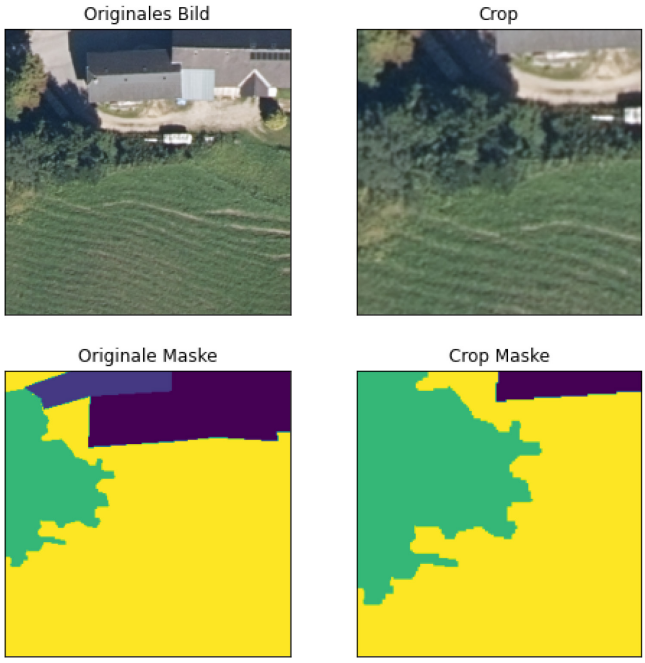


Abbildung 18: Beispiel für ein Crop (eigene Erstellung)

- **Translation (Verschiebung):** Bei der Verschiebung wird das Bild in eine zufällige Richtung verschoben. Jene Bereiche, die im Anschluss keinen Bildinhalt mehr aufweisen, müssen erneut interpoliert werden.

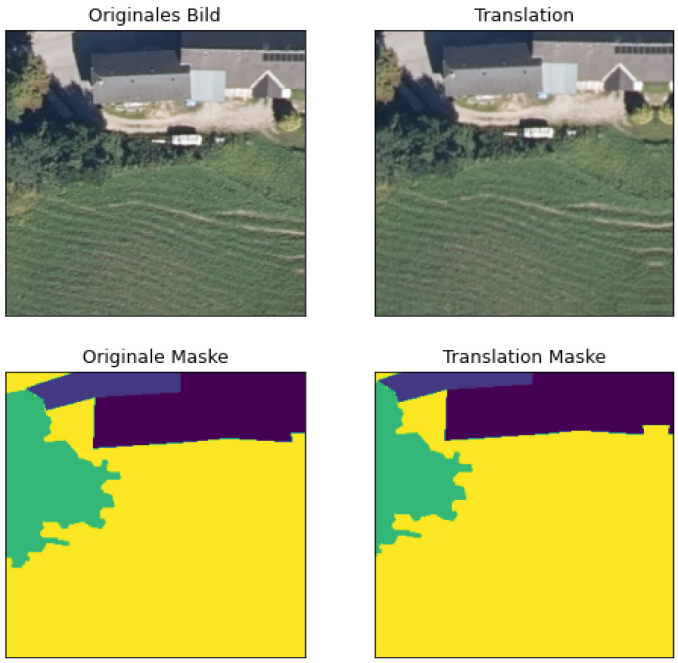


Abbildung 19: Beispiel für eine Translation (eigene Erstellung)

- **Blur (Verwischen):** Die Bildinhalte werden verwischt bzw. verschwommen. Hier gibt es verschiedenen Methoden wie den Gaussian Blur, Median Blur oder den Motion Blur. Bei dem Vorgang wird nur das Bild verändert, jedoch nicht die Maske.

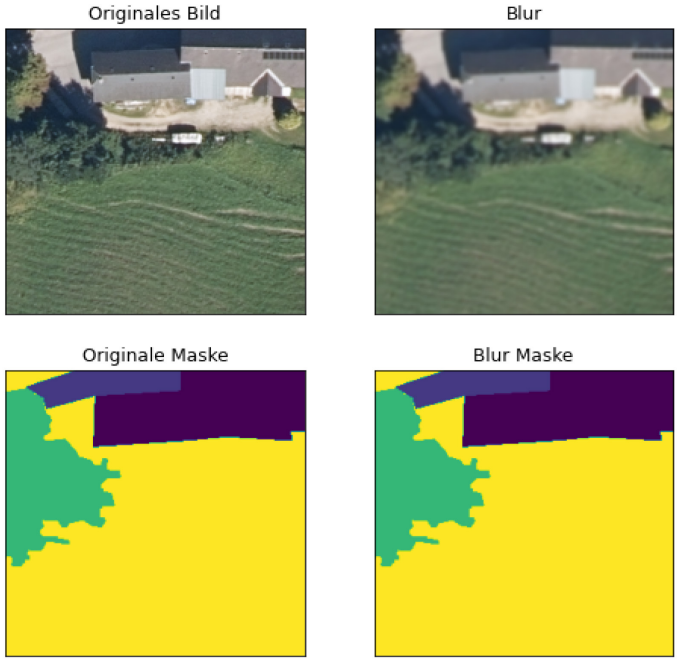


Abbildung 20: Beispiel für ein Blur (eigene Erstellung)

- **Kontrast und Helligkeit:** Bei dem Vorgang werden der Kontrast und die Helligkeit des Bildes verändert. Dadurch kann das Modell nach dem Training auch Bilder klassifizieren, die einen anderen Kontrast bzw. eine andere Helligkeit aufweisen. Auch hier bleibt die Maske durch den Vorgang unberührt.

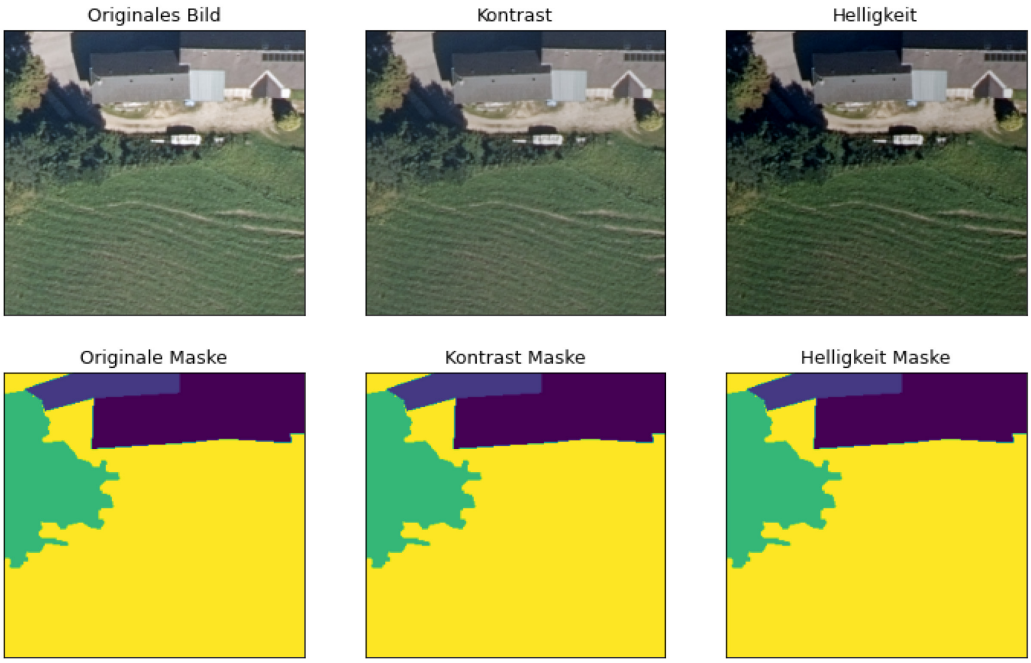


Abbildung 21: Beispiel für eine Veränderung der Helligkeit und des Kontrastes (eigene Erstellung)

- **Noise (Rauschen):** Beim Rauschen werden zufällige Bildpunkte im Bild verändert, um ein Rauschen in den Daten zu erzeugen. Diese Unruhe kann in manchen Fällen dazu führen, dass das Modell besser lernen kann. Das Rauschen kann unterschiedlich beschaffen sein. Es reicht von einzelnen schwarzen und weißen Punkten bis zu einem großflächigen Ausfall von Bildinhalten. Diese Augmentation wurde für diese Masterarbeit jedoch nicht verwendet.

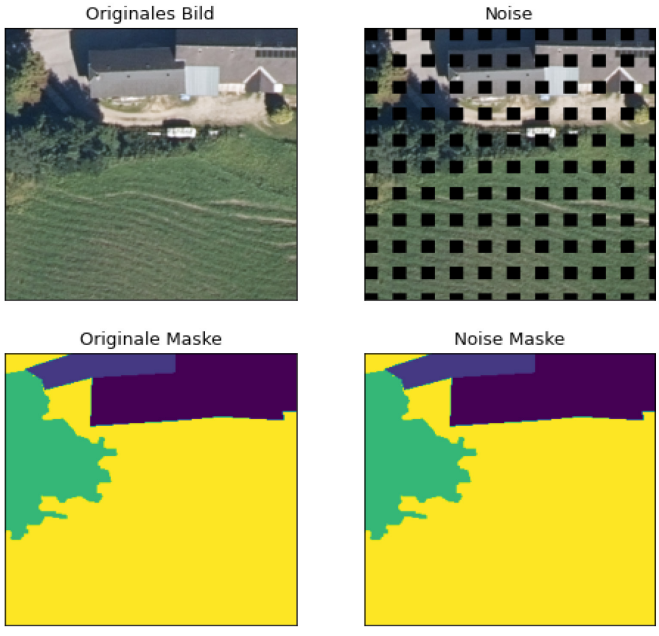


Abbildung 22: Beispiel für ein Noise (eigene Erstellung)

Es gibt eine große Anzahl an weiteren Methoden, die für die Image-Augmentationen verwendet werden können, die aber nur Abwandlungen der oben genannten Varianten sind (z.B. Gaussian Blur und Median Blur). Die Methoden müssen nach dem

Einsatzzweck ausgewählt werden und auch die Parameter der Veränderungsmethoden müssen entsprechend angepasst werden. Mit den Parametern kann der Benutzer auswählen, wie stark die Veränderungen ausfallen. Bei Orthofotos sind alle der oben genannten Methoden sinnvoll, jedoch müssen Werte wie der Kontrast oder das Verwischen der Bilder nur leicht angepasst werden. Leichte Kontrast- und Helligkeitsunterschiede bei Orthofotos sind durchaus normal, weshalb der Einsatz dieser Methode sinnvoll ist. Ein Blur, also das Verwischen der Bilder, sollte nur in wenigen Fällen benutzt werden, jedoch sollte die Ausprägung dieses Parameters nur sehr schwach gewählt werden (vgl. BUSLAEV et al. 2020: 12−13).

4.3.2. Interpolationsmethoden bei der Image-Augmentation

Bei vielen der oben genannten Augmentation-Methoden (z.B. Rotation, Scale, Translation) müssen unbekannte Bildinhalte interpoliert werden. Wenn zum Beispiel ein Bild um 45° gedreht wird, entstehen in den Ecken des Bildes mehrere Bereiche ohne Inhalt. Diese müssen mithilfe einer Interpolation gefüllt werden. Nach KAEHLER und BRADSKI (2016: 254) können unterschiedliche Interpolationsmethoden verwendet werden:

- **Constant Values (Konstante Zahlen):** Dabei handelt es sich um die einfachste Interpolationsmethode. Der unbekannte Bildbereich wird mit einem zuvor festgelegten Wert gefüllt. Für natürliche Bilder wie Orthofotos ist diese Methode nicht geeignet.
- **Edge (Kanten):** Die Kanten der Bilder werden bis zum Rand des neuen Bildes weitergeführt, jedoch wird die Methode selten benutzt.
- **Reflect (Reflektion):** Die Bildwerte werden ab der Kante reflektiert und bis zum Ende geführt. Diese Interpolationsmethode kommt häufig zum Einsatz und kann auch für Orthofotos verwendet werden.
- **Symmetric:** Die Methode ist der Reflect-Methode sehr ähnlich. Der einzige Unterschied ist, dass auch die Werte der Kante einbezogen werden und diese ebenfalls reflektiert wird. Im Normallfall lässt sich kaum ein Unterschied zu der Reflect-Methode erkennen.
- **Wrap:** Das Bild wird im unbekannten Bereich wiederholt, es entsteht somit im Prinzip ein Kachelbild. Die Methode wird kaum eingesetzt und ist deshalb von keiner großen Relevanz.

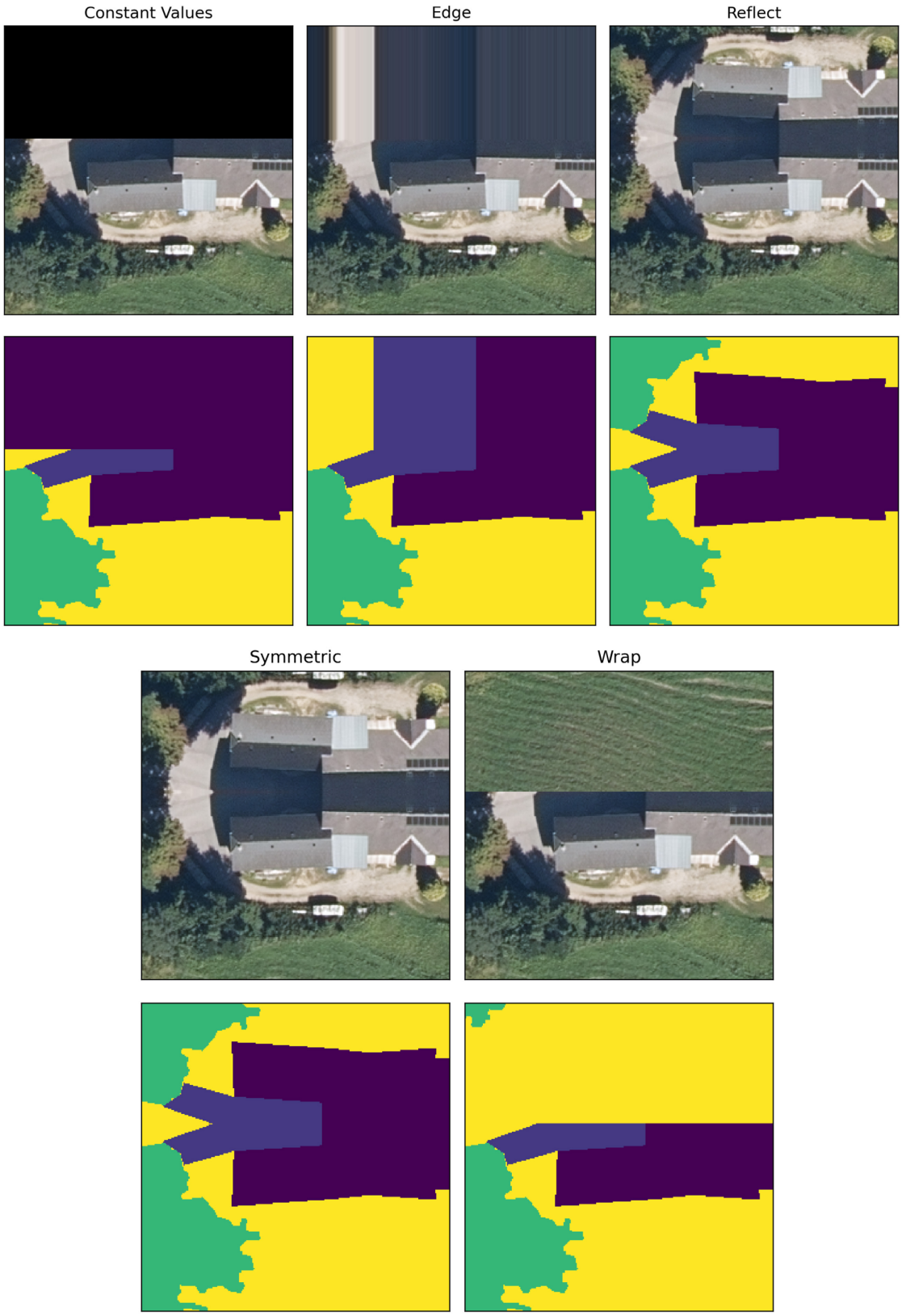


Abbildung 23: Beispiele für die fünf verfügbaren Interpolationsmethoden (eigene Erstellung)

4.3.3. Ergebnisse der Image-Augmentation

Da die Ergebnisse der Image-Augmentation nichts mit dem eigentlichen Training des CNNs zu tun haben, werden sie bereits in diesem Kapitel präsentiert. Durch die Image-

Augmentation werden zusätzliche Bilddaten generiert. Im Fall dieser Masterarbeit sind zwar von Beginn an ausreichend Daten vorhanden, jedoch gibt es hier eine ungleiche Verteilung der jeweiligen Klassen. Um die Verteilung der Klassen zur visualisieren, muss die gesamte Anzahl der Pixel pro Klasse im gesamten Datensatz gemessen werden. Dies kann mit einem einfachen Python-Skript (siehe Anhang III) erfolgen und führt zu folgender Verteilung:

Tabelle 7: Anzahl der Pixel nach den Klassen und deren Gesamtanteil (eigene Erstellung)

Klasse	Anzahl der Pixel	Anteil am gesamten Datensatz
Gebäude	23 581 481	0,74 %
Versiegelte Flächen	42 305 833	1,33 %
Unversiegelte Flächen	17 629 462	0,55 %
Gewässer	168 429 198	5,29 %
Nadelbäume	1 801 024 875	56,62 %
Laubbäume	181 785 453	5,71 %
Sonstiges	945 771 314	29,73 %

Wie in Tabelle 7 zu sehen ist, sind die Klassen ‚Nadelbäum' und ‚Sonstiges' im Datensatz stark überrepräsentiert. Insgesamt sind in etwa 57 % der Pixel der Klasse ‚Nadelbäume' und 30 % der Klasse ‚Sonstiges' zuzuordnen. ‚Gebäude', ‚Versiegelte Flächen' und ‚Unversiegelte Flächen' hingegen sind im Datensatz kaum vorhanden. Das würde beim Training des CNNs dazu führen, dass das Modell schlechte Klassifizierungsergebnisse liefert. In diesen Fall würde das Modell zu viele Bereiche als ‚Nadelbäume' und ‚Sonstiges' klassifizieren und die unterrepräsentierten Klassen nur schlecht oder gar nicht richtig klassifizieren. Um dem Problem entgegenzuwirken, kann eine Image-Augmentation für Bilder, in denen die unterrepräsentierten Klassen vorkommen, sinnvoll sein. Das Ziel besteht darin, dass die Anzahl der Pixel pro Klasse am Ende dieses Prozesses ausgewogener ist.

Für die Bilder werden all jene Methoden angewendet, die bereits in Kapitel 4.3.1 vorgestellt wurden. Dies erfolgt mithilfe eines Python-Skripts (siehe Anhang II) und der Python-Bibliothek *Albumentations,* die auf den bekannten Bibliotheken *NumPy*, *Scikit-Image*, und *OpenCV* aufbaut. Mit *Albumentations* lassen sich verschiedene Methoden der Image-Augmentation miteinander kombinieren und anwenden. Der Fokus dieses Vorgangs liegt auf jenen Klassen, die im gesamten Datensatz unterrepräsentiert sind. Dazu werden explizit die Bilder genommen, in denen die zusätzlich benötigten Klassen bis zu einem gewissen Anteil vorkommen. Anschließend werden nur jene Bilder für die Image-Augmentation verwendet, in denen der Anteil der unterrepräsentierten Klassen einen zuvor definierten Grenzwert überschreitet. Dadurch entstehen für die gesuchten Klassen zusätzliche Bilder und jene Bilder, in denen ausschließlich die überrepräsentierten Klassen vorkommen, werden für die Image-Augmentation ignoriert.

Damit der Datensatz nach dem Prozess in Hinsicht auf die Anzahl der Pixel pro Klasse, ausgewogener ist, muss zunächst auch festgelegt werden, wie viele abgewandelte Bilder aus einem einzelnen Bild erzeugt werden sollen. Da die benötigte Anzahl für jede Klasse unterschiedlich ist, muss für alle Klassen ein individueller Multiplikationsfaktor festgelegt werden. Dieser kann auf der Basis der zuvor berechneten Anteile (siehe Tabelle 6) berechnet werden. Dabei ergeben sich folgende Faktoren:

Tabelle 8: Faktoren für die jeweiligen Klassen zur Herstellung eines Gleichgewichts im Trainingsdatensatz (eigene Erstellung)

Klasse	Faktor	Theoretischer neuer Anteil am gesamten Datensatz
Gebäude	32	14,39 %
Versiegelte Flächen	16	12,91 %
Unversiegelte Flächen	32	10,76 %
Gewässer	4	12,85 %
Nadelbäume	0,5	17,17 %
Laubbäume	4	13,86 %
Sonstiges	1	18,03 %

Die in der Tabelle 8 genannten Anteile basieren auf einer theoretischen Rechnung, mit der versucht wird, ein Gleichgewicht herzustellen. Ein ausgewogener Datensatz ist bei der Problemstellung der Bildsegmentierung nicht möglich. Zudem kommen die Klassen ‚Sonstiges' und ‚Nadelbäume' in vielen Bildern vor, in denen auch die unterrepräsentierten Klassen vorhanden sind. Diese sind insbesondere im Umfeld um die Klassen ‚Gebäude', ‚Versiegelte Straßen' und ‚Unversiegelte Straßen' vorzufinden. Dadurch kommt es bei der Image-Augmentation auch zu einer Zunahme der Anzahl der Pixel dieser beiden Klassen, wodurch diese im Datensatz immer überrepräsentiert sein werden. Dieses Problem kann im Datensatz nicht verhindert werden und es kann nur versucht werden, dass die Anzahl der Pixel ausgeglichener wird.

Für die Bilder, die zu einem großen Teil die Klassen ‚Gebäude' und ‚Unversiegelte Straßen' beinhalten, werden immer 32 zusätzliche Bilder erzeugt. Für die versiegelten Flächen werden immer 16 zusätzliche Bilder generiert und für die Gewässer und Laubbäume werden vier neue Bilder erzeugt. Für die Anwendung der Modifikationen an den Bildern muss zuvor ein Workflow festgelegt werden, in dem angegeben wird, welche Transformationen durchgeführt werden sollen.

Bei den Transformationen wird zwischen ‚Pixel-Level Transformations' und ‚Spatial-Level Transformations' unterschieden. Die Pixel-Level Transformations verändern ausschließlich die Bilder, jedoch nicht die Masken. Solche Prozesse sind zum Beispiel der Blur oder die Anpassung des Kontrastes. Da die Veränderung des Kontrastes beim normalisierten Geländemodell (nDSM) sinnlos ist, muss dieses zunächst vom Bild getrennt und wie eine Maske behandelt werden. Die Spatial-Level Transformations

verändern auch die Masken und somit auch das nDSM. Beispiele für solche Transformationen sind das Umdrehen, die Drehung, das Wenden und das Beschneiden (vgl. BUSLAEV et al. 2020: 8).

Jede Bildoperation wird dabei mit einer gewissen Wahrscheinlichkeit durchgeführt. Bei manchen Operationen kann von mehreren verfügbaren Transformationen lediglich eine einzelne ausgewählt werden. Dieser Schritt dient dazu, dass zwei ähnliche Operationen, zum Beispiel ein Blur, nicht zweimal hintereinander durchgeführt werden. Nach dem Prozess werden das abgewandelte RGBI-Bild und das nDSM wieder miteinander kombiniert und können gemeinsam mit der Maske als TIFF-Bild abgespeichert werden. Durch die Augmentation der Bilder haben die jeweiligen Klassen den folgenden Anteil am Trainingsdatensatz:

Tabelle 9: Anzahl der Pixel nach den Klassen und deren Gesamtanteil nach der Image-Augmentation (eigene Erstellung)

Klasse	Anzahl der Pixel	Anteil am gesamten Datensatz
Gebäude	626 656 853	6,9 %
Versiegelte Flächen	665 896 450	7,4 %
Unversiegelte Flächen	152 821 473	1,7 %
Gewässer	876 862 437	9,7 %
Nadelbäume	2 448 591 413	27,0 %
Laubbäume	1 101 177 497	12,2 %
Sonstiges	3 181 857 813	35,1 %

Aus Tabelle 9 wird ersichtlich, dass sich die Anteilswerte der jeweiligen Klassen durch die Image-Augmentation stark verbessert haben. Sie haben zwar keinen Idealwert erreicht, jedoch sind die unterrepräsentierten Klassen zumindest stärker im Gesamtdatensatz vertreten. Zur Beseitigung der Unausgeglichenheit kann beim Training eine zusätzliche Methode angewandt werden. Diese wird in Kapitel 4.4.2 beschrieben.

Diese generierten Daten könnten direkt zum Trainieren des Modells verwendet werden. In der Praxis ist es jedoch effizienter, die Bilder zusammen mit den Masken in das sogenannte *TFRecord*-Format umzuwandeln. Dieser Prozess wird im nachfolgenden Kapitel beschrieben.

4.3.4. TFRecord-Dateien

Wie die Bilddaten zum Training eines CNNs verwendet werden, hängt von der Anzahl der Bilder und der benötigten Speichergröße ab. Sind nur wenige Daten vorhanden, können sie direkt in den Arbeitsspeicher (RAM) eingelesen werden und stehen von dort dem Modell zum Training zur Verfügung. Mit einer mittleren Zugriffszeit von 20 bis 70

Nanosekunden ist dies die schnellste und effizienteste Methode (vgl. THORMÄHLEN 2018, TensorFlow 2020a).

Da die benötigten Speichermengen bei Bilddaten im Normalfall größer sind als der vorhandene Arbeitsspeicher, müssen die Daten direkt von der Festplatte eingelesen werden. Die Zugriffszeiten sind hier von der Festplattentechnologie abhängig und variieren ebenso innerhalb der jeweiligen Technologie. Eine normale rotierende Festplatte (HDD – Hard Disk Drive) hat eine mittlere Zugriffszeit von 3 bis 20 Millisekunden und ein Flashspeicher (SSD) hat eine mittlere Zugriffszeit von drei bis 20 Mikrosekunden. Eine ebenso große Rolle spielt die Art des Zugriffes. Muss oft auf viele verteilte Dateien zugegriffen werden, kommt es zu einem schlechten Datendurchsatz. Wird auf wenige größere Dateien, die in einem einzelnen Ordner gespeichert sind, zugegriffen, wird ein wesentlich größerer Datendurchsatz erreicht. Bei einer HDD wird im Durchschnitt einen Datendurchsatz von 0.3 Gigabyte pro Sekunde (ungefähr 300 Megabyte) erreicht (vgl. THORMÄHLEN 2018).

An dieser Problematik setzt auch das *TFRecord*-Format an und versucht hier, das Einlesen der Daten zu beschleunigen. Es handelt sich dabei um ein binäres Speicherformat, das gemeinsam mit *TensorFlow* im Jahr 2015 veröffentlicht wurde. Beim binären System werden Informationen, zum Beispiel Zahlen oder Buchstaben, als Abfolge von zwei verschiedenen Symbolen (im Normalfall sind es die Symbole ‚0' und ‚1') dargestellt. Im Bereich der Informatik wird der Binärcode aufgrund der großen Effizienz und Zuverlässigkeit verwendet. Diese beiden Faktoren macht sich auch das *TFRecord*-Format zunutze und speichert die Zahlenwerte (Gleitkommazahlen und ganze Zahlen) in einem binären Format ab (vgl. TensorFlow 2020b).

TFRecords basieren auf den von Google entworfenen Protocol Buffers. Dabei handelt es sich um ein programmiersprachen- und plattformneutrales Format, mit dem verschiedenste Daten serialisiert werden können. Bei einer Serialisierung werden strukturierte Daten (z.B. mehrere Bilder und dazu passende Labels) zu einer einzelnen Datei zusammengefasst. Dabei werden sie in ein binäres Format umgewandelt und dreidimensionale Daten wie Orthofotos werden eingeebnet. Später können diese immer wieder in den Originalzustand gebracht werden. Ein weiterer Vorteil dieser Zusammenfassung von Daten ist, dass der benötigte Speicherplatz reduziert werden kann und beim Einlesen der Daten nur auf wenige Daten zugegriffen wird (vgl. REITZ und SCHLUSSER 2016: 255−257).

Darüber hinaus besitzen diese Dateitypen den Vorzug, dass sie bezüglich der Kompatibilität problemlos eingelesen werden und perfekt in *TensorFlow* integriert werden können. Der einzige Nachteil bei *TFRecords* ist, dass die TIFF-Bilder einmalig in das Format umgewandelt werden müssen.

Bevor diese Umwandlung in Python vorgenommen werden kann, muss die Struktur der Daten angegeben werden. Dazu muss zunächst festgelegt werden, welche Datentypen (z.B. Float32, UInt8, usw.) in der *TFRecord*-Datei gespeichert werden sollen. Dazu wird eine Art Beispiel angegeben, wie die Daten formatiert sind und wie sie im *TFRecord*-Format gespeichert werden sollen (Binär, Integer oder Float). Auf Basis dieser

Informationen werden alle vorhandenen Bilder und die dazugehörigen Informationen serialisiert. Die Bilder mit den Labels werden zu einem Textobjekt (String) umgewandelt und als Byteobjekt geführt. Im Anschluss werden die Daten in ein Binärformat umgewandelt. Andere Informationen, zum Beispiel die Bilddimensionen, können zusätzlich zu den Bilddaten als Integerobjekte in das *TFRecord*-Format gebracht werden (vgl. TensorFlow 2020b).

Um die Performance der *TFRecord*s zu steigern, sollte nicht eine einzelne große Datei erstellt werden. Grundsätzlich kann ein einzelnes *TFRecord* auch in mehrere Stücke, sogenannte Shards (Fragment, Bruchstück), aufgeteilt werden. Jedes dieser Shards sollte zwischen 100 und 200 Megabyte haben, damit eine optimale Performance erreicht werden kann. Zudem können die Daten bereits während der Konvertierung in das *TFRecord*-Format verlustfrei komprimiert werden, wodurch zusätzlich Speicherplatz gespart werden kann (vgl. TensorFlow 2020c).

Damit *TFRecord*-Dateien erneut geöffnet werden können, muss der Nutzer unbedingt wissen, welchen Dateityp die Daten ursprünglich hatten (Float, Integer, usw.). Da die Bilder sowie die Masken als String im Binärformat gespeichert wurden, müssen sie auch wieder als String eingelesen werden. Damit das Bild wieder in seine ursprüngliche Form umgewandelt werden kann, müssen die binären Strings zunächst zu einer Gleitkommazahl (Float32) umgewandelt werden. Nach dem Vorgang muss das Bild wieder in seine ursprünglichen Dimensionen umgeformt werden. Dies geschieht über eine einfache Umformung der Daten in das ursprüngliche Format. All diese Vorgänge können mit *TensorFlow* durchgeführt werden (vgl. REITZ und SCHLUSSER 2016: 255−257, TensorFlow 2020b).

TFRecords können für das Training direkt als Batches eingelesen werden. Je nach gewählter Batchgröße (siehe Kapitel 3.5) werden aus den einzelnen Bildern Batches mit der vom Benutzer gewählten Größe erstellt. Damit die Labels für das Training verwendet werden können, müssen sie noch in die sogenannte One-Hot-Encoding-Form umgewandelt werden (vgl. Steadforce 2020). Dabei wird jeder Grauwert der Maske (bei 7 Klassen somit 7 Grauwerte) in einen eigenen Bildkanal umgewandelt. Jeder dieser sieben Kanäle enthält dabei ausschließlich die Werte Null und Eins. Zum Beispiel haben nach dem Vorgang alle Gebäude im ersten Kanal den Wert ‚1' und die restlichen den Wert ‚0'. Im zweiten Kanal hingegen haben alle versiegelten Flächen den Wert ‚1' und die übrigen wiederum den Wert ‚0'. Dies wird automatisch für jede Klasse berechnet und kann dann zusammen mit den Bilddaten als Batch zum Trainieren des CNNs bereitgestellt werden. Dieser Vorgang ist wesentlich effizienter, als wenn die Daten direkt aus den TIFF-Bildern eingelesen werden.

4.4. Modell

Um ein CNN-Modell trainieren zu können, muss zunächst eine Modellarchitektur gewählt werden. Eine solche Architektur kann selbst konzipiert werden oder es kann auf bestehende und bekannte Architekturen zurückgegriffen werden (siehe Kapitel 3.7). Im Anschluss muss das Modell mit gewissen Parametern kompiliert bzw. erstellt werden.

Dabei müssen bereits verschiedene Parameter, zum Beispiel die Lernrate oder die Loss-Funktion, angegeben (vgl. Kapitel 3.5) werden.

4.4.1. Modellarchitektur

Die Wahl des richtigen Modells ist für das Training entscheidend. In vielen Fällen gibt es keine richtige oder falsche Wahl, da sich die Modellarchitekturen stark voneinander unterscheiden und in vielen Fällen nicht miteinander verglichen werden können. Bekannte Segmentierungsmodelle sind das U-Net, Linknet oder PSPNet, wobei das am meisten genutzte Modell das U-Net ist. Das originale Paper zu U-Net von RONNEBERGER et al. (2015) wurde laut Google Scholar mittlerweile nahezu 13 000-mal zitiert. Das U-Net wurde ursprünglich für die Segmentierung von medizinischen Bildern entwickelt, kann jedoch auch für andere Problemstellungen eingesetzt werden. Aufgrund der intensiven Verwendung des U-Nets wird dieses Modell auch für die vorliegende Masterarbeit eingesetzt (vgl. YAKUBOVSKIY 2019, RONNEBERGER et al. 2015).

Das Modell gibt jedoch nur das Grundgerüst an und nicht, welche Layer (Convolutional Layer, Pooling Layer usw.) verwendet werden. Diese Faktoren werden durch einen sogenannten Backbone festgelegt. Auch hier steht eine Vielzahl von Backbones zur Verfügung, die sich grundlegend voneinander unterscheiden. Beispiele dafür sind ResNet, DenseNet, Inception oder EfficientNet. Mit solchen Modellen lassen sich Bilder klassifizieren, aber nicht segmentieren. Deswegen müssen diese Backbones in die richtige Form, zum Beispiel in die Form des U-Nets, gebracht werden (siehe Kapitel 3.4) (vgl. YAKUBOVSKIY 2019).

Als Backbone wurde für diese Masterarbeit das ResNet ausgewählt. Dieses Modell ist aufgrund der vielen berechenbaren Parameter zwar rechenintensiv und umfangreich, hat bei Wettbewerben in den letzten Jahren immer die besten Ergebnisse erzielt. Die Entwickler des ResNets haben im Laufe der Zeit mehrere Modelle entworfen, die immer mehr berechenbare Paramater beinhalten. Beim bisher größten ResNet, dem ResNet-152, können 60 Millionen Parameter berechnet werden. Das kleinste Modell der ResNet-Familie ist das ResNet-18 mit wesentlich weniger berechenbaren Parametern (vgl. FUNG 2017).

Das ResNet wurde von HE et al. (2015) entwickelt und insbesondere mit dem Hintergedanken entworfen, das Problem des *Vanishing Gradient* (siehe Kapitel 3.6) zu beseitigen. Dieses Problem tritt besonders dann auf, wenn das Modell immer mehr Schichten hintereinander besitzt und immer tiefer wird. Damit ein solches Modell trotzdem lernen kann, nutzen die Autoren sogenannte *skipping connections*. Dabei werden ein oder mehrere

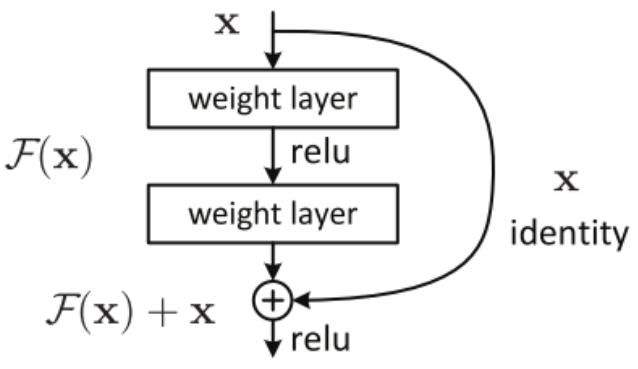


Abbildung 24: Beispiel für einen Residual-Block (SAHOO 2018)

Layer während der Berechnung übersprungen. Ein solcher Block wird auch als *Residual Block* bezeichnet (siehe Abbildung 24). Dieser Block überführt Gewichte, also Informationen, aus einem vorherigen Layer in einen nachfolgenden und überspringt dabei mehrere Layer. Dadurch kann ein solches Modell wesentlich mehr Schichten besitzen, da kontinuierlich Informationen aus vorherigen Schichten übernommen werden. Es gibt mehrere Arten von *Residual Blocks* (z.B. mit oder ohne Normalisierung oder auch eine oder zwei Aktivierungen), die im ResNet verwendet werden (vgl. SAHOO 2018, HE et al. 2015).

Aufgrund des großen Bekanntheitsgrades und der mittlerweile regelmäßig bestätigten zuverlässigen Ergebnisse (z. B. durch die Image Net Challenge) wird das ResNet als Backbone für diese Masterarbeit verwendet. Dabei wird auf das ResNet-101 zurückgegriffen, das ungefähr 40 Millionen berechenbare Parameter besitzt. Allgemein sind Modelle mit vielen Parametern für komplexe Problemstellungen bestimmt und sie neigen bei wenigen Klassen oder zu einfachen Problemen zu ungenauen Ergebnissen.

4.4.2. Modellkompilierung

Nachdem ein Modell ausgewählt wurde, muss es zunächst erstellt bzw. kompiliert werden. Zur Kompilierung muss die Loss-Funktion, die zur Berechnung des gemachten Fehlers herangezogen wird, angegeben werden (siehe Kapitel 3.5). Diese muss mit Bedacht gewählt werden, da jede Funktion Vor- und Nachteile hat. Für diese Masterarbeit wird ausschließlich der Cross-Entropy-Loss verwendet. Dieser Fehlerwert wird häufig für das Training von CNNs verwendet und ist quasi der Standard unter den Loss-Funktionen (vgl. NIERADZIK 2018).

Da die Daten noch immer im Ungleichgewicht sind, müssen Gewichtungen für die Loss-Funktion berechnet werden. Das Ziel ist es, dass die überrepräsentierten Beispiele einen geringeren Einfluss auf den Fehler haben als die unterrepräsentierten Trainingsdaten. Folgende Gewichtungen ergeben sich aus der Anzahl der Pixel pro Klasse:

Tabelle 10: Gewichtungen für den Loss (eigene Erstellung)

Klasse	Gewichtung
Gebäude	1
Versiegelte Flächen	1
Unversiegelte Flächen	2
Gewässer	1
Nadelbäume	0.5
Laubbäume	1
Sonstiges	0.5

Ohne diese Gewichtung kann es im Extremfall dazu kommen, dass das Modell während des Trainings lernt, dass es in den Fotos mehrheitlich die überrepräsentierte Klasse gibt, und dieser Klasse zu viele Pixel zuordnet. Durch eine Gewichtung der Klassen, kann dies

verhindert werden. Damit eine Gewichtung angewandt werden kann, wird der gewichtete Cross-Entropy-Loss verwendet (vgl. NIERADZIK 2018). Je niedriger der Loss wird, desto weniger Fehler macht das Modell bei der Klassifizierung. Deshalb ist darauf zu achten, dass dieser Loss-Wert in Bezug auf die Validierungsdaten während des Trainings möglichst minimiert wird.

Ebenso müssen die zu evaluierenden Genauigkeitsmaße (Metrics) definiert werden. Bei der Aufgabenstellung der Bildsegmentierung muss dazu auf spezielle Genauigkeitsmaße zurückgegriffen werden, da die berechneten Vorhersagen des Modells und die wahren Labels pixelweise miteinander verglichen werden müssen. Für diese Masterarbeit werden dazu zwei Genauigkeitsmaße verwendet: der *Intersection over Union-Score* (IoU-Score) und der F-Score (F-Maß).

Der IoU-Score gibt den Quotienten der Schnittfläche und der Vereinigung zwischen der wahren und der berechneten Segmentierung an. Mithilfe folgender Formel kann dieser Index berechnet werden (vgl. MITSCHKE und HEIZMANN 2019: 656).

$$IoU(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

Je höher der Wert des IoU-Scores ist, desto besser und genauer ist die Überlappung. Bei einem Wert von 1 ist eine zu 100 % lagerichtig geschätzte Segmentierung gemacht worden.

Das F-Maß hingegen ist ein rein statistisches Maß. Es ergibt sich aus dem sogenannten Recall (Trefferquote) und der Precision (Genauigkeit). Der Recall ist der Anteil der korrekt berechneten Pixel, gemessen an allen tatsächlich korrekten Pixeln. Die Precision hingegen gibt den Anteil der korrekt berechneten Pixel an allen berechneten, somit auch allen falsch berechneten Pixeln an. Mit folgender Formel kann der F1-Score berechnet werden (vgl. CLEMATIDE 2011).

$$F_1 = 2 * \frac{precision \times recall}{precision + recall}$$

Auch beim F1-Score erreicht der beste Wert 1. Dieser Wert würde bedeuten, dass alle berechneten Klassen mit den tatsächlichen Klassen übereinstimmen (vgl. CLEMATIDE 2011). Allgemein gilt somit: Je höher die Metrics und je kleiner der Loss, desto genauer und besser ist das Modell.

Zusätzlich kann bei der Kompilierung des Modells ein *Optimizer* definiert werden. Dabei handelt es sich um verschiedene Methoden, die das Training beschleunigen bzw. optimieren sollen und somit schneller zu besseren Ergebnissen führen. Hierfür gibt es mehrere Methoden, für die Masterarbeit wurde jedoch die Adam-Methode gewählt. Sie wurde im Jahr 2015 von KINGMA und BA veröffentlicht und hat sich mittlerweile zu einer weit verbreiteten Optimierungsmethode entwickelt. Bei der Adam-Methode erhält jedoch jeder berechenbare Paramater eine individuelle und während des Trainings anpassbare Lernrate. Bei einem CNN ohne die Adam-Methode oder auch bei anderen Optimierungsmethoden besitzt das Modell eine fixe Lernrate für alle Parameter. Durch diese individuelle Lernrate können die Parameter schneller optimiert werden und es

benötigt weniger Epochen bis zum besten Ergebnis. Einer der wesentlichen Vorteile der Adam-Methode ist zudem, dass sie einfach zu konfigurieren ist und im Normalfall die Standardparameter der Optimierungsmethode gut funktionieren (vgl. KINGMA und BA 2015: 1−3, BROWNLEE 2017).

Mithilfe aller oben genannten Elemente kann das Modell anschließend kompiliert werden. Darauffolgend kann das Modell bereits mit den zuvor erstellten *TFRecords* trainiert werden. Zusätzlich können auch sogenannte *Callbacks* definiert werden. Es handelt sich dabei um ein Set von Funktionen, die immer an einem gewissen Punkt während des Trainings einsetzen. Der wichtigste Callback ist jener, der das Modell und die antrainierten Gewichtungen nach jeder Epoche speichert, damit diese auch im Nachhinein wieder geladen und verwendet werden können. Da nicht jede Epoche zu einer Verbesserung des Losses oder der Genauigkeitsmaße führt, können auch nur jene Gewichtungen einer Epoche gespeichert werden, die die beste Performance in Bezug auf ein spezifisches Maß (Loss oder Metrics) haben (vgl. Keras 2019).

Ein anderer Callback, der von großem Nutzen ist, ist das vorzeitige Stoppen des Trainingsvorganges. Bei diesem kann eines der beiden Maße durch die Funktion beobachtet werden und er stoppt bei keiner Verbesserung den Trainingsvorgang. Dadurch wird überflüssige Rechenzeit eingespart. Dabei kann auch festgelegt werden, wie lange die Funktion bis zum Abbruch warten soll (vgl. Keras 2019).

Ein für das Training weniger relevanter Callback, der auch für diese Masterarbeit verwendet wurde, ist das TensorBoard-Callback. Bei TensorBoard handelt es sich um ein Visualisierungswerkzeug, das ebenso von Google entwickelt wurde. Mit diesem können unter anderem bereits während des Trainings die Loss- und Metrics-Werte verfolgt und visualisiert werden. Ebenso kann die Modellarchitektur visuell betrachtet und der zeitliche Verlauf der Histogramme der Gewichte visualisiert werden. Das TensorBoard ist ein umfangreiches Werkzeug, das dem Nutzer bei der Verbesserung des Modells helfen kann (vgl. GÉRON 2019: 317-320, TensorFlow o. J.)

4.5. Training

Die Trainingsphase kann in zwei Phasen aufgeteilt werden: das Suchen und Finden von geeigneten Hyperparametern, die sogenannte Hyperparameteroptimierung oder auch das Hyperparametertuning durch eine der in Kapitel 3.8.2 genannten Methoden, und das anschließende eigentliche Training. Die Findung von passenden Hyperparametern kann übersprungen werden, für ein gutes Klassifizierungsmodell sollte die Methode jedoch zunächst angewendet werden. Das Python-Skript zur Hyperparametersuche ist in Anhang IV nachzulesen und jenes zum eigentlichen Training kann in Anhang V nachgelesen werden.

4.5.1. Hyperparametersuche

Wie bereits in Kapitel 3.8.2 beschrieben, können verschiedene Methoden zur Findung der richtigen Hyperparameter herangezogen werden. Im Falle der vorliegenden Masterarbeit wird die Bayesian Optimization verwendet, da mit dieser die Optimierungszeit stark

verkürzt wird und gute Ergebnisse erzielt werden können. Da kein individuell erstelltes Modell verwendet wird, sondern auf das ResNet zurückgegriffen wird, muss ausschließlich die Lernrate (Learning-Rate) bestimmt werden (vgl. GÉRON 2019: 322).

Die anderen anpassbaren Hyperparameter, zum Beispiel die Anzahl der Zwischenschichten oder die Art der Aktivierung, sind bereits durch das ResNet vorgegeben und können nicht mehr optimiert werden. Die Größe der Batches (Batchsize) könnte zwar ausgetestet werden, jedoch können aufgrund der verwendeten Hardware ausschließlich Batchgrößen zwischen 2 und 16 getestet werden. Da im Normallfall eine Batchgröße, die kleiner als 16 ist, zu stark ausgeprägten Instabilitäten im Modell führt (starke Schwankungen der Klassifizierungsgenauigkeit bei unbekannten Beispielen), sollte keine kleinere Größe verwendet werden. Eine größere Batchsize von 32 oder 64 würde zwar zur Stabilität des Modells beitragen, jedoch fehlt es hierfür bei der verwendeten Hardware an Platz im Videospeicher (vgl. GÉRON 2019: 326).

Die Suche nach passenden Parametern erfolgt mit denselben Trainingsdaten wie beim anschließenden Training. Für die Implementierung der Bayesian Optimization wird die Programmbibliothek *Keras-Tuner* verwendet. Damit können die Methoden aus Kapitel 3.8.2 zur Suche nach passenden Hyperparametern in *TensorFlow* implementiert werden. Vor der Suche müssen bei den numerischen Werten des Modells die (Lernrate-)Wertebereiche angegeben werden, in denen sich die zu optimierenden Werte befinden können. Dieser Merkmalsraum wird auch als ‚Hyperparameter Space' bezeichnet und ist für die Optimierung unverzichtbar. Die Lernrate darf maximal eine Höhe von 0,1 annehmen und die Untergrenze liegt bei 0,0001.

Aus dem Merkmalsraum werden zufällige Werte ausgewählt. Mit diesen werden anschließend immer fünf Epochen durchgerechnet und am Ende wird das Ergebnis evaluiert. Die Evaluierung kann entweder auf Basis des Loss-Wertes oder der Genauigkeitswerte vorgenommen werden. Im Anschluss daran wird ein neues Modell kompiliert, wodurch auch alle bisherigen Ergebnisse und Gewichtungen auf den ursprünglichen Zustand gesetzt werden. Die Hyperparameter werden anhand der vorherigen Evaluierung angepasst und mit den neuen Werten wird dann erneut ein neues Modell für fünf Epochen berechnet. Dieser Vorgang wird 25-mal durchgeführt und am Ende des Vorgangs kann die beste gefundene Lernrate für das eigentliche Training verwendet werden.

Da verschiedene Bildkanäle zur Verfügung stehen, können auch diese überprüft werden. Folgende Kombinationen von Bildkanälen werden getestet:

- RGB-Orthofotos
- RGBI-Orthofotos
- RGBI-Orthofotos und das nDSM

Für jede dieser Kombinationen muss eine eigene Suche nach Hyperparametern durchgeführt werden. Dabei steht für alle Kombinationen der oben genannte Merkmalsraum zur Verfügung. Am Ende werden alle drei Modelle für das eigentliche

Training verwendet. Damit soll überprüft werden, welche Kombination die besten Ergebnisse liefert. Die Ergebnisse werden in Kapitel 5.1 beschrieben.

4.5.2. Trainingsphase

Während des Trainings hat der Nutzer kaum eine Möglichkeit, in den Vorgang einzugreifen. Es läuft weitgehend automatisch ab und hört von allein wieder auf, wenn ein entsprechendes Callback gesetzt wurde. Während des Trainings bekommt der Nutzer jedoch kontinuierlich ein Feedback über den Fortschritt in der jeweiligen Epoche. Dabei handelt es sich um Informationen zu dem Loss und den Genauigkeitsmaßen in Bezug auf die Trainings- und Validierungsdaten.

Bevor die Daten trainiert werden, müssen sie gemischt werden. Diese Funktion wird bereits durch *TensorFlow* bereitgestellt und kann direkt mit den *TFRecords* erfolgen. Dazu wird eine genau definierte Anzahl an Bildern in den Arbeitsspeicher eingelesen. Aus diesem Datenpool, dem sogenannten *Shufflebuffer*, werden im Anschluss zufällig Bilder und die dazugehörigen Masken ausgewählt. Diese werden dann zu einem Batch zusammengefügt und dienen als Trainingsbeispiele für das CNN. Während des Trainings werden kontinuierlich die restlichen Daten eingelesen, wodurch es zu keinen Pausen innerhalb einer Epoche kommt. Der Datenpool leert sich somit erst am Ende einer Epoche und belegt während des Trainings den Arbeitsspeicher.

Am Ende einer Epoche werden die Validierungsdaten wie oben beschrieben eingelesen. Der Unterschied liegt darin, dass für die Validierungsbeispiele keine Fehlerrückführung gemacht wird und die Berechnung dadurch wesentlich schneller funktioniert. Bei der Validierung werden ebenso der Loss und die Genauigkeitswerte berechnet, damit die Performance des Modells überprüft werden kann.

Die Trainingsphase dauert im Normalfall mehrere Tage bis Wochen. Die Dauer hängt im Wesentlichen von der Datenmenge, der Größe des Modells und der Geschwindigkeit der Grafikkarte ab. In wenigen Fällen benötigt ein Modell unter 50 Epochen, um den optimalen Zustand zu erreichen. In vielen Fällen sind über 100 Epochen nötig, um einen optimalen Loss-Wert zu erlangen. Am Ende bzw. mit TensorBoard bereits während des Trainings kann der Verlauf von den Loss- und den Genauigkeitswerten betrachtet werden.

Wie bereits im Kapitel (siehe Kapitel 4.5.1) beschrieben, müssen unterschiedliche Modelle auf der Basis verschiedener Bildkanäle trainiert werden. Nicht immer führt es zu einer Verbesserung, wenn möglichst viele Kanäle in einem CNN verwendet werden. Dabei werden die im vorherigen Kapitel genannten Kombinationen ausgetestet und trainiert.

Es werden ausschließlich die besten Modelldurchläufe, also jene mit den besten Ergebnissen in Bezug auf den Loss, gespeichert. Dabei wird das gesamte Modell inklusive der berechneten Gewichtungen behalten. Dadurch kann das Modell auch zu späteren Zeitpunkten geladen und für eine Klassifikation von neuen unbekannten Orthofotos verwendet werden. Die Ergebnisse dieses Prozesses werden in Kapitel 5.2 beschrieben.

4.6. Klassifikation

Die Klassifikation wird im Zuge dieser Masterarbeit in zwei Teile aufgeteilt: die eigentliche Klassifikation, die durch *TensorFlow* gemacht wird, und die Bewertung der Ergebnisse. Im ersten Schritt wird mithilfe des CNNs eine automatische Segmentierung und Klassifizierung der unbekannten Orthofotos abgeleitet. In einem zweiten Schritt müssen diese Daten dann auf ihre Genauigkeit geprüft werden. In folgendem Kapitel werden die Prozesse, wie eine Klassifikation entsteht und wie die anschließende Bewertung durchgeführt wird, genauer erläutert. Das Python-Skript zur Klassifikation ist im Anhang VI zu finden. Eine Abbildung des verwendeten ArcGIS-Modells zur Evaluierung der Ergebnisse ist in Anhang VII abgebildet.

4.6.1. Klassifikation in TensorFlow und *NumPy*

Die Klassifikation erfolgt durch eine sogenannte *Prediction* durch das Modell mit den zuvor errechneten Gewichtungen. Für die Vorhersage können nur jene Bildkanäle verwendet werden, die bereits während des Trainings verwendet wurden. Wurden zum Beispiel zum Trainieren ausschließlich RGB-Bilder verwendet, können auch nur RGB-Bilder klassifiziert werden. Wie bereits in Kapitel 3.4 erwähnt, ist die Bildgröße aufgrund des Fully Convolutional Networks irrelevant. Das einzige Kriterium, das dabei immer erfüllt werden muss, ist, dass die Größe eines Bildausschnittes zwingend ein Vielfaches der Zahl 2 sein muss.

Für die Klassifikation werden Orthofotos im ÖLK-Blattschnitt, da die Landcover-Klassifikation des BEV auch auf diesen Blattschnitt setzt. Damit lassen sich die Endergebnisse besser miteinander vergleichen und es wird ein großer Bereich klassifiziert. Dazu wird das Orthofoto zunächst in Python mithilfe der Programmbibliothek *RasterIO* eingelesen. Mit dieser können GeoTIFF-Bilder mit einer beliebigen Anzahl von Kanälen eingelesen und in einen *NumPy*-Array umgewandelt werden. Ein solcher Array kann beliebig viele Dimensionen besitzen und ähnelt einer Matrize, mit der Zahlenwerte in einer gitterförmigen Anordnung gespeichert werden. Die Arrays werden im Arbeitsspeicher hinterlegt, wodurch für ein Orthofoto schnell acht Gigabyte an Arbeitsspeicher benötigt wird. Der wesentliche Vorteil ist jedoch die schnelle Zugriffszeit aus dem Arbeitsspeicher.

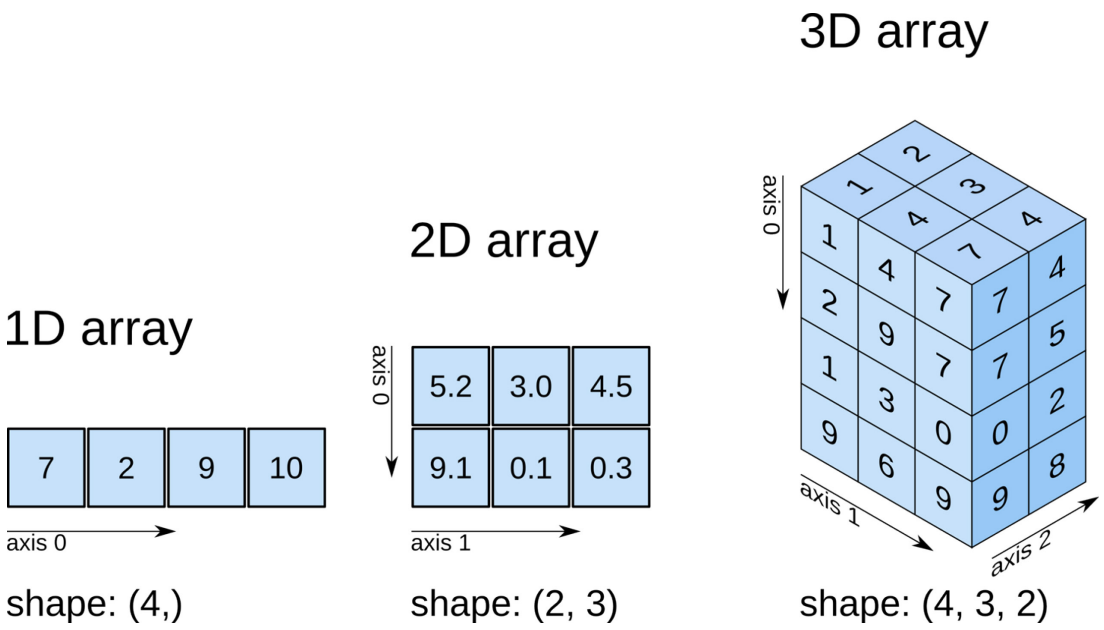


Abbildung 25: Beispiele für verschiedene *NumPy*-Arrays (https://fgnt.github.io/python_crashkurs_doc/include/numpy.html)

Aufgrund der limitierten Kapazitäten der Grafikkarte kann die Prediction für das Orthofoto nicht in einem einzelnen Rechenschritt durchgeführt werden. Deswerden müssen Bildausschnitte aus dem zuvor geladenen Bild herausgeschnitten werden (Array-Indexing), die anschließend für die Prediction verwendet werden. Die Bildgröße für die Prediction kann so groß gewählt werden, wie sie maximal im Videospeicher der Grafikkarte bearbeitet werden kann. Dies kann ausschließlich durch das Austesten von verschiedenen Größen herausgefunden werden. Im Falle dieser Masterarbeit werden immer Bildausschnitte mit einer Größe von 2560 × 2560 Pixeln klassifiziert.

Dieser Bildausschnitt wird über das Orthofoto geschoben (*Sliding Window*) und an jeder Position wird die Klassifikation berechnet. Generell sollten sich die Positionen überlappen, da am Rand der Bildausschnitte ein großer Teil der für das CNN relevanten Kontextinformationen fehlen. Durch diese Überlappung der Bildbereiche kann der Verlust des Kontextes vermindert werden und es kommt zu deutlich besseren Ergebnissen. Für die vorliegende Masterarbeit wurde daher eine Schrittweite von 256 Pixeln gewählt. Der Ausschnitt wird somit sowohl horizontal als auch vertikal um diese Schrittweise über das Bild geschoben.

Da ein Orthofoto im ÖLK-Blattschnitt insgesamt 25 000 × 25 000 Pixel besitzt, kann mit einer Ausschnittsgröße von 2 560 × 2 560 Pixeln und einer Schrittweite von 256 Pixeln keine gerade Anzahl an Schritten erzeugt werden. Da keine halben Schritte möglich sind, müssen an jeder Seite des Orthofotos jeweils 300 Pixel hinzugefügt werden. Dabei werden die Kanten des Bildes ebenso wie bei der Data-Augmentation gespiegelt und es entsteht ein Orthofoto mit 25 600 × 25 600 Pixeln. Dadurch kann das gesamte Orthofoto mit der gewählten Ausschnittsgröße und der Schrittweite klassifiziert werden.

Das Ergebnis dieser Berechnung ist ein Array mit sieben Kanälen, wobei jeder Kanal eine bestimmte Klasse repräsentiert. Die Pixelwerte dieser Kanäle geben die

Wahrscheinlichkeit an, mit der der Pixel einer bestimmten Klasse zugeordnet wird. Dieser Wert liegt zwischen 0 und 1 und kann auch als Prozentwert dargestellt werden. Je höher der Wert ist, desto wahrscheinlicher ist er einer bestimmten Klasse zuzuordnen. Ein Pixel kann jedoch nicht mehreren Klassen zugeordnet werden, da der Wert sich über alle Kanäle auf 1 summiert. Wenn zum Beispiel ein Pixel mit einer Wahrscheinlichkeit von 0.55 der ersten Klasse zugeordnet wird, dann verteilen sich die restlichen 0.45 auf die anderen verfügbaren Klassen. Dadurch kann es zu keiner Überlappungen zwischen den Klassen kommen, da es immer nur einen einzelnen höchsten Wert gibt.

Da eine solche *Prediction* immer nur für den jeweiligen Bildausschnitt ist und mit jedem Schritt überschrieben wird, müssen die Ergebnisse in einem separaten Array (dem Klassifikations-Array) abgespeichert werden. Die Speicherung dieser Werte erfolgt mithilfe eines dreidimensionalen *NumPy*-Arrays. Der Array hat dieselbe Größe wie das Orthofoto, da für jedes Pixel des Orthofotos ein Wahrscheinlichkeitswert gespeichert werden soll. Die Anzahl der Kanäle dieses Klassifikations-Arrays orientiert sich jedoch an der *Prediction*, da alle sieben Kanäle der Vorhersage gespeichert werden sollen. Da ein Array im Float32-Format und zudem mit einer solchen Dimension beinahe 20 Gigabyte an Arbeitsspeicher benötigt, sollte zudem das Float16-Format verwendet werden. Dieses Format benötigt nur die Hälfte des Arbeitsspeichers und ist im Hinblick auf die Präzision (um die Hälfte weniger Nachkommastellen) ausreichend. Da ein Array vor der Verwendung immer mit Zahlen gefüllt werden sollte, wird dieser zunächst mit Nullen gefüllt.

Mithilfe von zwei einfachen Schleifen in Python kann der Bildausschnitt über das Orthofoto geschoben werden. Eine Schleife zieht das Fenster horizontal über das Bild und die andere vertikal. An jeder Position wird ein Ausschnitt aus dem originalen Orthofoto-Array ausgelesen und eine Klassifikation für diesen Ausschnitt gemacht. Die Ergebnisse der Klassifikation werden dann in dem Klassifikations-Array gespeichert. Dabei werden sie an dieselbe Position geschrieben, an der sich auch der dazugehörige klassifizierte Bildausschnitt befindet. Damit die bereits bestehenden Werte im Klassifikations-Array, die wegen der Überlappung an jeder Position enthalten sind, nicht überschrieben werden, werden nur jene Werte ersetzt, die größer als die ursprünglichen Werte an der jeweiligen Position sind.

Nachdem das ganze Orthofoto auf diese Weise klassifiziert wurde, muss der Klassifikations-Array in ein sinnvolles Format gebracht werden. Theoretisch könnten auch die Wahrscheinlichkeitswerte als ein TIFF mit sieben Kanälen abgespeichert werden, jedoch ist dies nicht wirklich hilfreich. Deswegen wird mithilfe von *NumPy* geprüft, in welchem Kanal des Klassifikations-Arrays sich der höchste Wahrscheinlichkeitswert befindet. Dieser Vorgang wird für jeden einzelnen Pixel durchgeführt und anschließend werden die Positionen der höchsten Wahrscheinlichkeitswerte (also der Kanal) in einen neuen Array mit der Größe des Orthofotos geschrieben. Dieser Array besitzt nur mehr einen Kanal und kann als Graustufenbild dargestellt werden. Jede Graustufe entspricht dabei wieder den Klassen aus Kapitel 4.2.

Die Daten müssen im Anschluss im GeoTIFF-Format abgespeichert werden und können dann mithilfe eines Geoinformationsprogrammes wie ArcGIS Pro eingelesen werden. Die Speicherung der Daten erfolgt erneut mithilfe von *RasterIO*. Dabei werden die Daten bereits georeferenziert abgespeichert und können direkt mit einem Geoinformationssystem eingelesen werden.

4.6.2. Bewertung der Ergebnisse

Die Qualität der Ergebnisse wird zunächst optisch bewertet. Dazu wird kontrolliert, wie gut die Abgrenzungen der Polygone mit den Bildinhalten übereinstimmen und wie genau diese sind. Ebenso wird nach etwaigen Fehlern gesucht, damit diese evaluiert werden können. Damit die Genauigkeit der Ergebnisse jedoch auch quantitativ bestimmt werden kann, werden zwei unterschiedliche Varianten angewendet. Im ersten Schritt wird auf den für die Klassifikation verwendeten ÖLK-Blättern ein zufälliges Gebiet mit einer Ausdehnung von 1 000 × 1 000 Metern manuell klassifiziert und dient als Referenz (die Ground-Truth-Daten) für die quantitativen Messungen. Dabei wird insbesondere die Kontrolle der Gebäude-, Gewässer- und Straßenpolygone von großer Bedeutung sein, da diese auf den Orthofotos gut zu erkennen sind und sich eindeutig von den anderen Klassen unterscheiden lassen.

Wie bereits bei der Klassifikation der Trainingsdaten lassen sich die Bäume bzw. Wälder teilweise nur schwer von den anderen Klassen unterscheiden (siehe Kapitel 4.2). Deshalb wird bei den Klassen ‚Nadelbäume', ‚Laubbäume' und ‚Sonstiges' auf eine quantitative Messung verzichtet. Die Ergebnisse würden bereits durch die manuelle Klassifikation der Ground-Truth-Daten ungenau werden, was zu unbrauchbaren Ergebnissen führen würde. Aus diesem Grund werden die drei oben genannten Klassen nur visuell bewertet. Dabei wird überprüft, ob die Klassen zu dem Bildinhalt des Orthofotos passen und ob die Abgrenzung dieser der Realität entsprechen.

Die Überprüfung erfolgt für jede der oben genannten Klassen einzeln, da die Aufarbeitung so wesentlich verständlicher ist. Im ersten Schritt wird ausschließlich die Gebäudeklasse untersucht und erst im Anschluss werden die Gewässer und die Straßen überprüft. Dazu müssen die Klassifizierungen des CNNs, die als Raster vorliegen, mithilfe von ArcGIS Pro in Polygone umgewandelt werden. Im Anschluss können diese Klassifikationsergebnisse der jeweiligen Modelle mit den Ground-Truth-Daten auf verschiedene Arten (Intersect und Erase) miteinander verschnitten werden (siehe Anhang VII). Am Beispiel des Gebäudes ergeben sich folgende Maße:

- ‚True-Positive'-Pixel: Dabei handelt es sich um jene Pixel, bei denen das Modell die korrekte Klasse (Gebäude) zugeteilt hat, also um eine richtige Klassifikation.
- ‚False-Positive'-Pixel: Das Modell hat einen Pixel einer Klasse (Gebäude) zugeteilt, obwohl dieser einer anderen Klasse entspricht. Dabei handelt es sich somit um eine falsche Klassifikation.
- ‚True-Negative'-Pixel: Das Modell hat erkannt, dass es sich bei dem Pixel um kein Gebäude handelt, und hat es somit richtig klassifiziert

- ‚False-Negative'-Pixel: Das CNN hat nicht erkannt, dass es sich bei einem Pixel um ein Gebäude handelt, und hat das Pixel einer anderen Klasse zugeordnet.

Die Berechnung erfolgt immer für eine binäre Klassifizierung. Es wird somit eine Klasse allen anderen Klassen gegenübergestellt. Die Identifikation dieser Pixel erfolgt mithilfe eines Modells in ArcGIS-Pro. Damit können für jede Klassifikation die oben beschriebenen Maße berechnet werden. Die ‚False-Positive'-Pixel können zwar auch damit ausgegeben werden, jedoch erfüllt dies keinen weiteren Zweck, weshalb darauf verzichtet wird. Für ein besseres Verständnis wurden die oben beschriebenen Maße als Abbildung visualisiert (siehe Abbildung 26).

		Wahre Klasse	
		Gebäude	Andere
Vorhergesagte Klasse	Gebäude	‚True-Positive'-Pixel	‚False-Positive'-Pixel
	Andere	‚False-Negative'-Pixel	‚True-Negative'-Pixel

Abbildung 26: Confusion Matrix am Beispiel von Gebäuden und den anderen Klassen (eigene Erstellung)

Die Ergebnisse dieses Vorgangs sind Polygone, die die oben beschriebenen Pixeltypen repräsentieren und in ArcGIS visualisiert werden können. Im Idealfall sind dies einzelne zusammenhängende Polygone, damit zum Beispiel ein Polygon pro Haus entsteht. Für einen besseren Vergleich werden die Flächen dieser Polygone in Quadratmeter ausgerechnet. Aus diesen Flächen kann anschließend der IoU-Score berechnet werden (siehe Kapitel 4.4.2). Zudem kann auch der Anteil der Flächen zur Gesamtfläche des Referenzgebietes berechnet und für die Analyse herangezogen werden.

In einem zweiten Schritt werden die Ergebnisse der Klassifikation des CNNs mit den Ergebnissen der Landcover-Klassifikation vom BEV verglichen. Diese stehen bereits für viele Flugblöcke von Österreich zur Verfügung und werden vom BEV bereitgestellt. Die Klasse ‚Gebäude' wird eines der zentralen Kriterien zur Evaluierung der Genauigkeit zwischen den beiden Methoden sein, da sie exakt denselben Bildinhalt klassifizieren. Eine weitere relevante Vergleichsklasse wird die Klasse ‚Gewässer' sein, bei der dieselben Inhalte klassifiziert werden. Die Gewässerklasse ist insbesondere deshalb so spannend, weil dieser Vorgang im BEV nicht automatisiert funktioniert und auf die manuell erfassten Daten des digitalen Landschaftsmodells zurückgegriffen werden muss. Auf einen quantitativen Vergleich zu den anderen Klassen in der Klassifikation des BEV wird verzichtet, da diese in vielen Fällen nicht mit den Kategorien des CNNs verglichen werden kann. Bei den Daten des BEV werden beispielsweise keine Straßen klassifiziert, sondern diese Klasse fällt in die Kategorie ‚Bodenflächen'. Diese umfasst, wie der Name bereits verrät, nicht nur Straßen, sondern auch andere Bereiche, die keine Straßen beinhalten.

Zudem werden die Landcover-Klassifikationen des BEV nach der automatischen Klassifikation erneut manuell überarbeitet, während das CNN nicht bearbeitet und direkt für den Vergleich herangezogen wird. Die Bewertung erfolgt ebenso über die Referenzdaten und die oben erwähnten Pixelarten.

5. Ergebnisse der Klassifikation mit einem Convolutional Neural Network

In diesem Kapitel werden die Ergebnisse der zuvor beschriebenen Methoden präsentiert. Zudem sollen die Fragestellungen aus der Einleitung beantwortet werden.

Da die Ergebnisse der Data-Augmentation bereits in Kapitel 4.3.1 erläutert wurden und es auch nicht Teil des eigentlichen Trainings ist, wird dieser Vorgang nicht mehr in diesem Kapitel behandelt. Dieses Kapitel beschäftigt sich ausschließlich mit den Ergebnissen aus den eigentlichen Trainingsphasen. Diese umfassen zu einem geringen Teil die Suche nach geeigneten Hyperparametern, das eigentliche Training und der anschließenden Klassifikation. Im ersten Unterkapitel werden zunächst die Ergebnisse der Hyperparametersuche beschrieben. Dieses Kapitel wird nur kurz sein, da der Fokus nicht auf diesem Vorgang liegen soll. Anschließend werden die Resultate aller trainierten Modellvarianten beschrieben. Als letztes Unterkapitel werden die Ergebnisse der Klassifikationen mit dem trainierten Modell erläutert. Dies wird auf Basis der zuvor beschriebenen Methoden passieren.

5.1. Ergebnisse der Hyperparametersuche

Dieses Kapitel beschäftigt sich mit der Suche nach passenden Hyperparametern. Wie bereits in Kapitel 4.5.1 erwähnt, kann damit ausschließlich die Lernrate für jedes Modell gefunden werden. Es wurden drei Modellvarianten getestet: ein Modell mit RGB-Daten (Modellvariante 1), eines mit RGBI-Daten (Modellvariante 2) und eines mit RGBI- und nDSM-Daten (Modellvariante 3). Andere Kombinationen könnten zwar einen Mehrwert haben, können jedoch aufgrund der langen Trainingsdauer nicht getestet werden. Die Bilder, die für das Training bereitstehen, sind für jede Modellvariante dieselben, da sie bereits im Vorhinein von den Validierungsdaten getrennt wurden. Durch die Verwendung derselben Bilder für jede Variante des Modells können die verschiedenen Modelle gut miteinander verglichen werden. Die Bilder werden dem CNN lediglich in einer zufälligen Reihenfolge präsentiert, was jedoch keinen großen Unterschied bei den Endergebnissen macht.

Insgesamt stehen im Datensatz 110 512 segmentierte Ausschnitte aus den verschiedenen Orthofotos zur Verfügung. Dem gegenüber stehen 11 052 segmentierte Bildausschnitte, die für die Validierung des Modells verwendet werden. Für alle Modelle wurde derselbe Merkmalsraum für den einen Hyperparameter ausgewählt. Dieser konnte für die Lernrate einen Wert zwischen 0,0001 und 0,1 annehmen. Der erste Wert wird zufällig aus dem Merkmalsraum ausgewählt. Der Vorgang benötigte mehrere Wochen, da pro Modell insgesamt 125 Epochen gerechnet werden. Da es drei Modellvarianten gibt, wurden somit 375 Epochen durchgerechnet.

Das Endergebnis ist überraschenderweise für jede Modellvariante nahezu gleich und hat eine optimale Lernrate im Bereich von 0,001 eruiert. Dies ist vermutlich darauf zurückzuführen, dass die Adam-Optimierungsmethode verwendet wurde (siehe Kapitel 4.4.2). Bei dieser hat jeder Parameter eine individuelle Lernrate und kann somit optimal

angepasst werden, was die gesamte Trainingsdauer wesentlich verkürzt. Die Lernrate bei der Adam-Methode stellt jedoch nur eine Obergrenze für die Lernrate dar und kann auch wesentlich kleiner sein.

Da die Trainingsdaten für jede Modellvariante ähnlich sind, wird sich dieser Wert bei der Höhe von 0,001 eingependelt haben. Auch die Autoren und Erfinder der Adam-Methode geben an, dass dies im Normalfall die optimale Lernrate für den Optimizer ist und damit die besten Ergebnisse erzielt werden. (vgl. KINGMA und BA 2015: 2). Obwohl sich die Werte nicht exakt bei 0,001 befinden, wird dieser der Einfachheit wegen genommen. Die Lernraten befinden sich im ungefähren Bereich um den gefundenen Wert. Für die optimalen Werte müsste die Hyperparametersuche wesentlich länger durchgetestet werden, was aus zeitlichen Gründen aber nicht möglich war.

5.2. Ergebnisse des Trainings

Im folgenden Kapitel werden die verschiedenen Modellvarianten trainiert, und zwar so lange trainiert, bis keine Verbesserung des Loss-Wertes und der Genauigkeitswerte mehr feststellbar ist. Das Training wird automatisch abgebrochen, wenn die Bedingungen, also keine Verbesserung der erwähnten Werte, dafür erfüllt werden. Im Fall dieser Masterarbeit kann das Modell maximal für 300 Epochen trainiert werden, jedoch wird das Training im Normalfall schon wesentlich früher abgebrochen.

Für die Evaluierung der Modelle wird die Höhe der Genauigkeitsmaße (IoU-Score und F1-Score) verwendet. Der Loss-Wert kann ebenso für die Evaluierung eingesetzt werden. Keiner dieser Werte gibt eine Prozentzahl an, sondern es sind ausschließlich bestimmte Maßzahlen, die für klar definierte Einsatzzwecke konzipiert sind. Hier gilt: Je geringer der Loss, desto weniger Fehler macht das Modell bei der Klassifizierung und desto genauer klassifiziert das Modell, wenn die beiden Genauigkeitsmaße hoch sind. Alle Maße in diesem Kapitel beziehen sich ausschließlich auf die Validierungsdaten und nicht auf die Trainingsdaten. Da das Modell während des Trainings ständig eine Fehlerrückführung zur Verbesserung des Losses macht, wird dieser Wert bei den Trainingsdaten kontinuierlich besser. Im Normalfall verbessern sich entsprechend auch die Genauigkeitsmaße. Deshalb kann mit den Trainingsdaten keine Aussage über die Genauigkeit des Modells gemacht werden. Daher wurden für die Evaluierung ausschließlich die Validierungsdaten genutzt.

5.2.1. RGB-Modell

Das RGB-Modell wurde insgesamt für 52 Epochen trainiert, bevor es zu einem automatischen Abbruch kam. Der finale Loss-Wert in der letzten Epoche betrug 0,00981. Der optimale Wert ist jedoch bei der 42. Epoche zu finden. Diese Tatsache ist auf den Vorgang des Early-Stoppings zurückzuführen (siehe Kapitel 4.4.2). Da das Early-Stopping immer zehn Epochen abwartet und kontrolliert, ob es noch zu einer Verbesserung der Werte kommt, ist der optimale Wert stets zehn Epochen vor der letzten berechneten Epoche zu finden. Dieser beträgt 0,00956 und ist somit etwas geringer als bei der 52. Epoche.

Auch im Hinblick auf die Genauigkeitsmaße können die finalen Zahlen überzeugen. Die IoU-Score hatte bei der 42. Epoche eine Höhe von 0,9066 und der F1-Score erreichte mit einer Höhe von 0,9468 einen ebenso hohen Wert. Der F1-Score ist somit nahe seinem Maximalwert, wodurch das Modell eine nahezu perfekte Precision und einen optimalen Recall (siehe Kapitel 4.4.2) besitzt. Mit einem IoU-Score dieser Höhe kann zudem davon ausgegangen werden, dass die durch das CNN vorhergesagten Segmentierungen lagegenau sind. Die Verlaufskurven des Loss-Wertes und der Genauigkeitswerte über die Epochen hinweg sind in den nachfolgenden Abbildungen visualisiert (siehe Abbildung 27 und Abbildung 28). Die wiederholt erscheinenden Ausreißer, zum Beispiel bei der 22. und 35. Epoche, sind normal und können immer wieder auftreten. Gewöhnlich normalisieren sich die Werte jedoch kurz darauf und pendeln sich wieder ein.

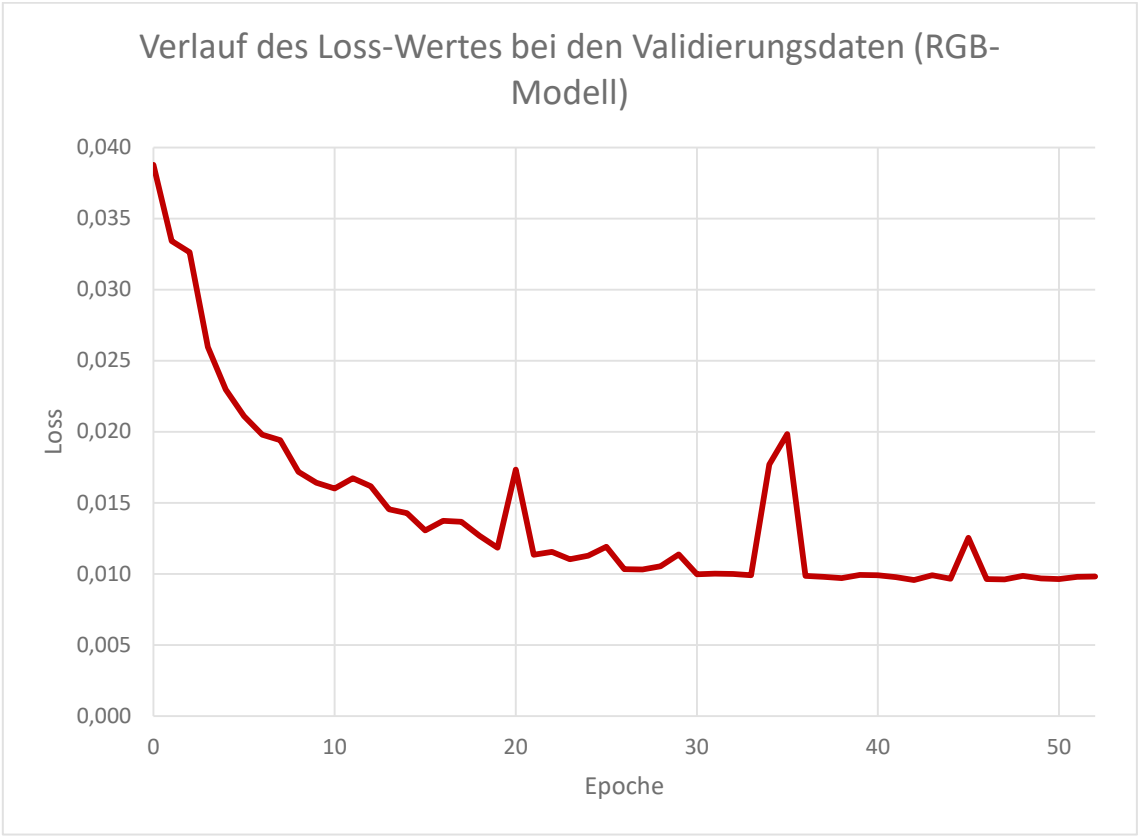


Abbildung 27: Verlauf des Loss-Wertes beim RGB-Modell (eigene Erstellung)

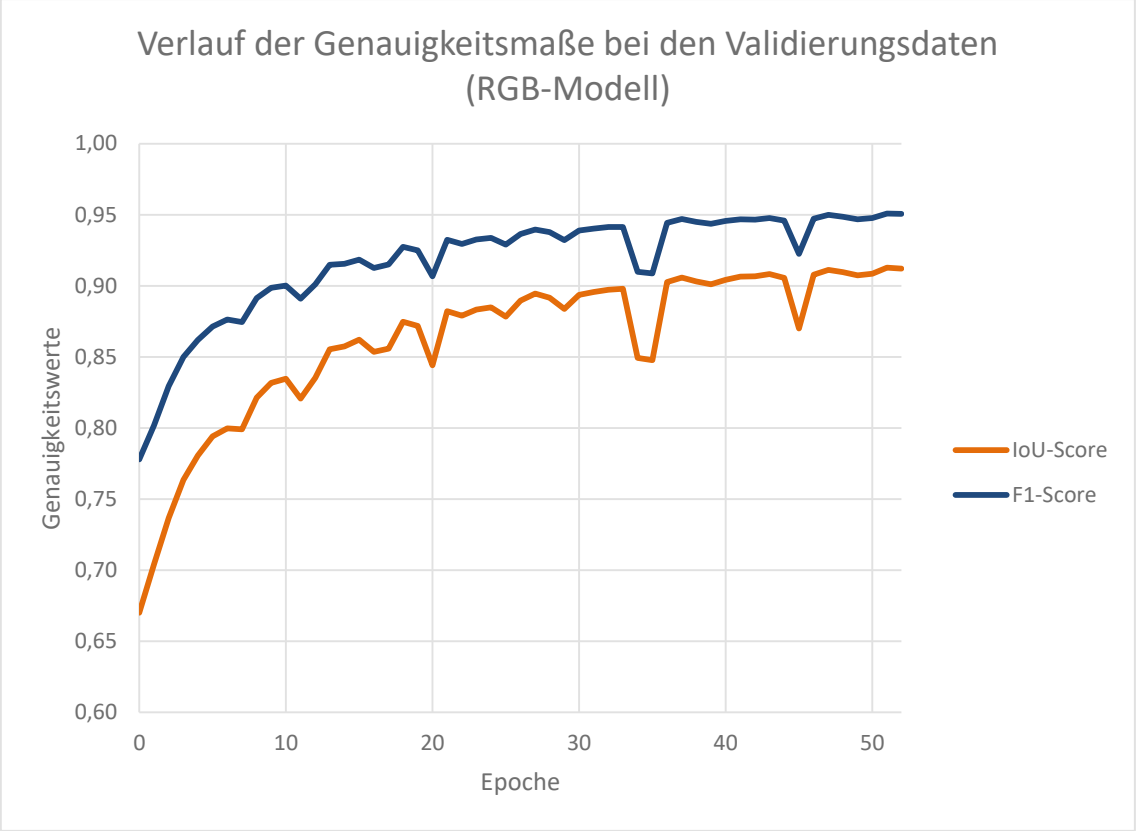


Abbildung 28: Verläufe der Genauigkeitswerte beim RGB-Modell (eigene Erstellung)

5.2.2. RGBI-Modell

Auch das Modell, das mit den RGBI-Daten trainiert wurde, weist einen ähnlichen Verlauf des Loss-Wertes wie das RGB-Modell auf. Das Modell wurde für 61 Epochen trainiert, jedoch wurde der beste Loss-Wert bereits bei der 51. Epoche erreicht. Er betrug bei dieser Epoche 0,00922 und ist somit etwas geringer als jener des RGB-Modells. Beim Loss-Wert spielen jedoch schon so kleine Unterschiede eine Rolle, also kann davon ausgegangen werden, dass das RGBI-Modell bei der Klassifikation weniger Fehler oder zumindest weniger stark ausgeprägte Fehler macht. In den Genauigkeitsmaßen spiegelt sich diese Verbesserung wider. Der IoU-Score beträgt bei der 51. Epoche 0,9117 und ist somit besser als jener des RGB-Modells. Auch der F1-Score ist höher und beträgt bei derselben Epoche 0,9502, wodurch auch dieser leicht gestiegen ist.

Allgemein kann daraus geschlossen werden, dass dieses Modell für die endgültige Klassifikation vermutlich besser geeignet ist als das Modell, das mit den RGB-Daten trainiert wurde.

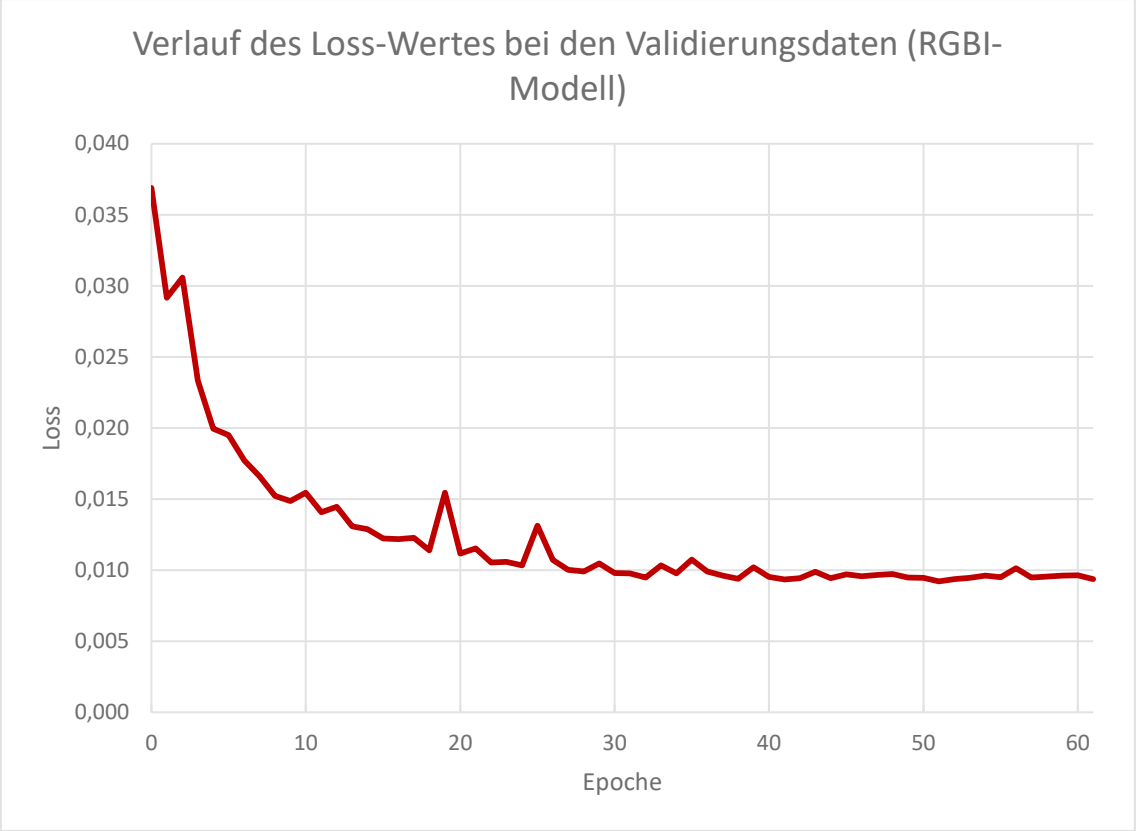


Abbildung 29: Verlauf des Loss-Wertes beim RGBI-Modell (eigene Erstellung)

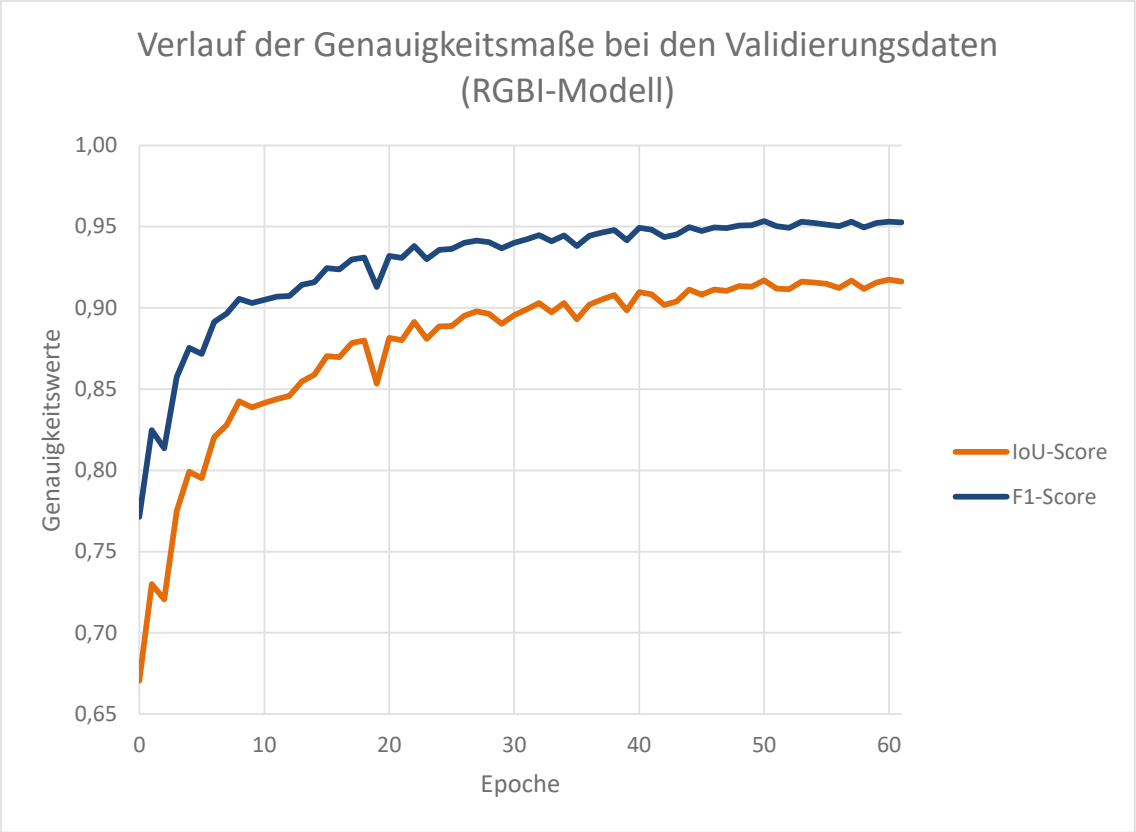


Abbildung 30: Verläufe der Genauigkeitswerte beim RGBI-Modell (eigene Erstellung)

5.2.3. RGBI- und nDSM-Modell

Das letzte Modell wurde auf Basis der RGBI-Daten und der Daten des nDSM trainiert. Es wurden somit alle Daten verwendet, die in den *TFRecord*-Dateien zur Verfügung standen. Insgesamt wurde dieses Modell für 68 Epochen trainiert, der beste Loss-Wert wurde jedoch bei der 58. Epoche erreicht. Bei dieser Epoche betrug der Loss-Wert 0,00767, im Vergleich zu den anderen beiden Modellen der geringste Wert. Dieser niedrige Wert wirkt sich auf die beiden Genauigkeitsmaße aus. Da der Fehler geringer ist, erreicht der F1-Score einen Wert von 0,956 und ist dadurch höher als beim RGBI-Modell. Auch der IoU-Score ist mit einem Wert von 0,921 um einiges höher als jener des vorherigen Modells.

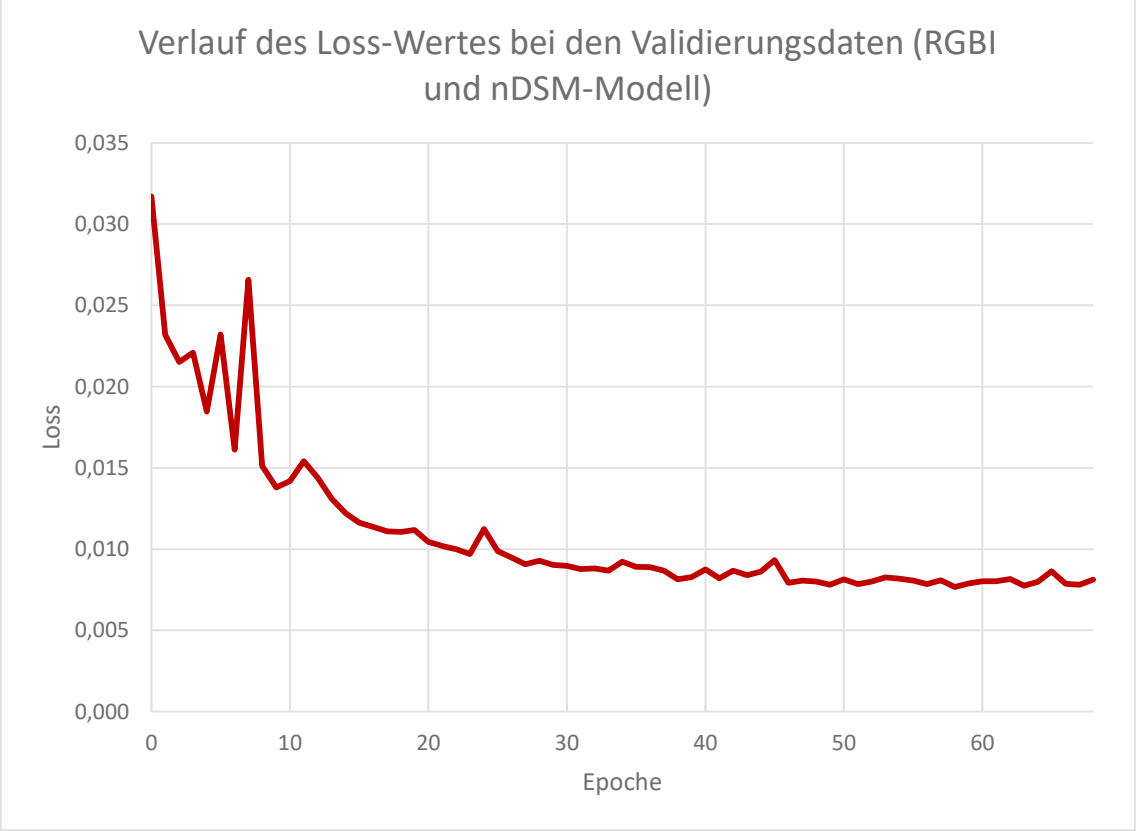


Abbildung 31: Verlauf des Loss-Wertes beim RGBI- und nDSM-Modell (eigene Erstellung)

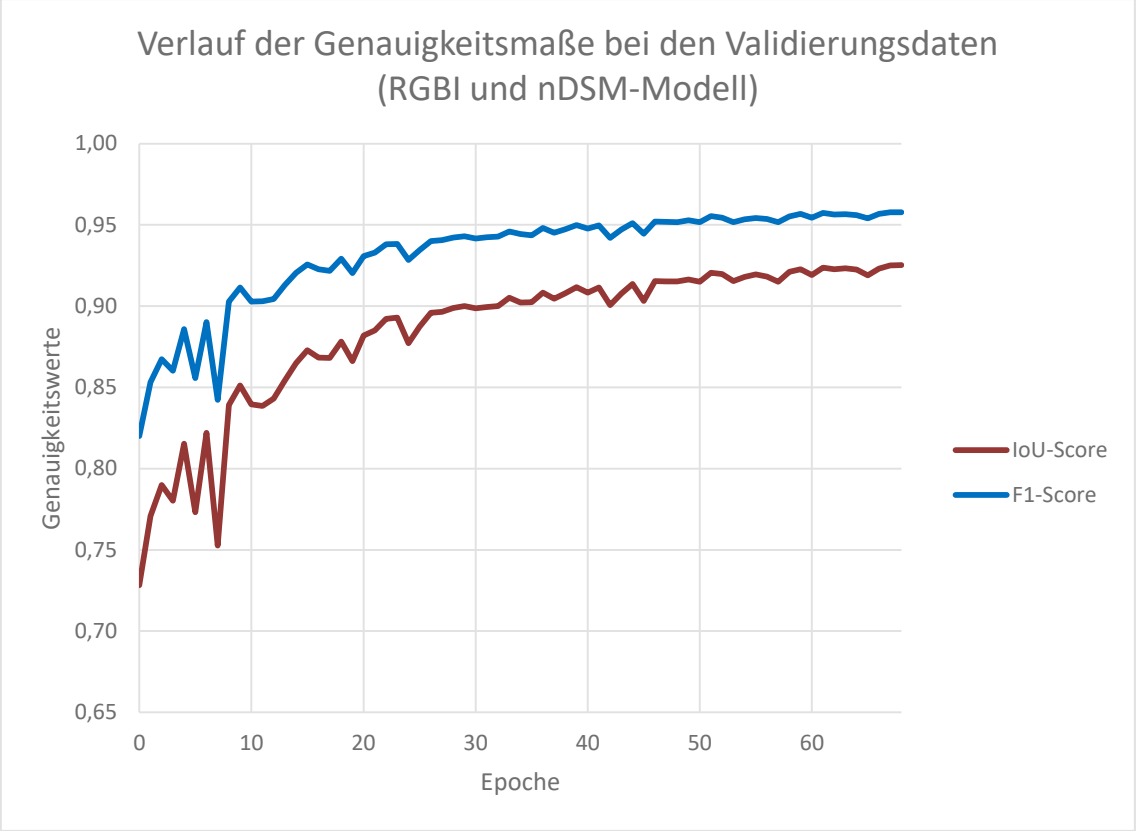


Abbildung 32: Verläufe der Genauigkeitswerte beim RGBI und nDSM-Modell (eigene Erstellung)

5.2.4. Zusammenfassung des Trainings

Die Zahlen zeigen, dass das zuletzt getestete Modell (RGBI- und nDSM-Modell) in der Theorie am besten für die Klassifikation von unbekannten Daten geeignet ist. Die eigentliche Performance kann jedoch erst gemessen werden, wenn eine Klassifikation auf unbekannten Daten durchgeführt wird. Im Falle der vorliegenden Masterarbeit werden Orthofotos aus einem anderen Gebiet Österreichs genommen, die in keinerlei Zusammenhang mit den Trainingsdaten stehen. Diese Tatsache ist wichtig, weil die Validierungsdaten in gewissem Zusammenhang mit den Trainingsdaten stehen, da diese aus dem gesamten zur Verfügung stehenden Datensatz entnommen wurden. Da der gesamte Datensatz auf denselben Orthofotos basiert, sind sich die Trainingsdaten und die Validierungsdaten in gewisser Weise ähnlich.

Die Überprüfung auf vollkommen unbekannten Daten erfolgt im nachfolgenden Kapitel.

5.2.5. Optische Bewertung des Trainings

Für die Bewertung der Performance der drei Modelle wird ein Orthofoto aus dem Bereich von Mariazell (6830-100) herangezogen. Dieses wird auch für die nachfolgende endgültige Klassifikation verwendet, jedoch zuvor für die optische Bewertung herangezogen. Für diesen Prozess wird es zunächst keine quantitative Überprüfung geben. Es wird lediglich geprüft, ob die Modelle für die Klassifikation von unbekannten Daten funktionieren und ob die Ergebnisse plausibel sind. Dazu wird immer ein kleiner

Bildausschnitt eines Orthofotos klassifiziert und visualisiert. Folgende Ergebnisse bilden sich daraus:

- **RGB-Modell:** Die Klassengrenzen sind scharf und genau abgegrenzt. Bei den Gewässern gibt es immer wieder Verwechslungen zwischen den Klassen, diese sind jedoch minimal. Versiegelte und unversiegelte Flächen können in vielen Fällen unterschieden werden und sind akkurat. Das größte Problem hat das RGB-Modell mit Gewässern, bei denen es oft zu Verwechslungen mit den Klassen ‚Sonstiges' und ‚Unversiegelte Flächen' kommt.
- **RGBI-Modell:** Bei dem Modell entstehen ähnliche Ergebnisse wie beim ersten Modell. Lediglich die Gewässer können mit diesem Modell besser und genauer klassifiziert werden. Die restlichen Klassen sind nahezu deckungsgleich mit denen des RGB-Modells.
- **RGBI- und nDSM-Modell:** Das Modell zeigt trotz der guten Ergebnisse beim Training keine zuverlässigen Klassifikationen bei unbekannten Daten. Dies liegt vermutlich daran, dass die Geländemodelle auf Basis von Laserscanndaten in Österreich von jedem Bundesland selbst erfasst werden und dann für das BEV bereitgestellt werden. Aufgrund der unterschiedlichen Erfassungen entsteht ein inhomogenes Geländemodell, womit das CNN nicht mehr umgehen kann.
  Mit dem Modell entstehen zwar Klassifikationsergebnisse, die verwendet werden könnten, die Qualitätsunterschiede zwischen den ersten Modellen sind jedoch sofort mit bloßem Auge erkennbar. Das Modell kann in vielen Fällen keine Häuser von Straßen unterscheiden und umgekehrt.

Daraus ergibt sich, dass das letzte Modell, das ebenso das Geländemodell einbezieht, nicht verwendet werden sollte. Dabei sollte eher auf Daten gesetzt werden, die von einer einzigen Quelle stammen und auf demselben Konzept (z. B. einer Kamera) basieren. Da alle Modelle funktionieren, werden die Ergebnisse aller drei Modelle auch für eine quantitative Analyse verwendet.

5.3. Ergebnisse der Klassifikation

Die Ergebnisse der Klassifikation sind die weitaus bedeutendsten dieser Masterarbeit. Sie betreffen die Segmentierung und die damit einhergehende Klassifikation der Orthofotos. Folgende drei Orthofotos wurden für die Klassifikation verwendet:

Tabelle 11: Verwendete ÖLK- und DKM-Blätter für die Klassifizierung und deren Besonderheiten (Quelle: eigene Erstellung)

Blattschnittnummer und Standort	Besonderheiten und topografische Gegebenheiten
6830-100 (ÖLK): Mariazell	Die Trainingsdaten stammen aus derselben Befliegung wie dieses Orthofoto. Es wurde somit von der gleichen Kamera und zu einem ähnlichen Zeitpunkt gemacht. Alle trainierten Klassen kommen darin vor, jedoch sind kaum Laubbäume vorhanden.

Blattschnittnummer und Standort	Besonderheiten und topografische Gegebenheiten
1926-100 (ÖLK): Imst	Die Landschaft auf dem zweiten Orthofoto befindet sich in der Nähe von Imst in Tirol und wurde von einem anderen Unternehmen aufgenommen als die anderen beiden Orthofotos. Es ist auch Gebirge zu finden, jedoch kein Hochgebirge. Ebenso ist der Nadelwald am meisten vertreten, jedoch sind alle Klassen zu finden.
3728-102 (ÖLK): Kufstein	Die Landschaft auf dem dritten Orthofoto befindet sich in einem dicht besiedelten Gebiet in Kufstein in Tirol und beinhaltet viele Gebäude und Straßen. Zudem kommt auch ein Fluss in dem Ausschnitt vor.
7535-68 (DKM): Wien	Die auf dem letzten und vierten Orthofoto abgebildete Landschaft befindet sich im Wienerwald. Es dient ausschließlich dazu, zu prüfen, ob die Klasse ‚Laubbäume' korrekt klassifiziert wird. Die anderen im Bild vorkommenden Klassen werden nicht evaluiert.

Die Klassifizierungsergebnisse der oben genannten Blattschnittnummern, sind im Anhang (siehe Anhang VIII bis XI) zu finden. Für jedes Blatt ist dazu ein Überblick über das Referenzgebiet vorhanden und im Anschluss können die Klassifizierungen mit den verschiedenen CNN-Modellen gefunden werden.

5.3.1. Ergebnisse des ersten Orthofotos (Mariazell)

Für das erste Orthofoto wird die quantitative Analyse zunächst mit allen drei Modellen durchgeführt. Dabei wird im Wesentlichen geprüft, ob die optische Bewertung mit den quantitativen Ergebnissen übereinstimmt. Die beiden ungenauesten Modelle werden anschließend verworfen und nicht mehr für eine Klassifikation von den anderen Orthofotos verwendet. Für die anderen drei Orthofotos wird die Klassifikation ausschließlich mit dem besten Modell durchgeführt. Zum Vergleich werden zudem die Klassifikationsergebnisse vom BEV herangezogen und auf dieselbe Weise quantitativ bewertet.

Insgesamt befinden sich im Referenzgebiete 87 Gebäude mit einer durchschnittlichen Fläche von 167 Quadratmetern. Im Referenzgebiet können 14 409 Quadratmeter der Gebäudeklasse zugeordnet werden. Wie in der nachfolgenden Abbildung gut zu erkennen ist, umfassen die meisten Gebäude im Referenzgebiet zwischen 100 und 200 Quadratmeter. An der gesamten Größe des Referenzgebiets (1 000 × 1 000 Meter) gemessen, besitzen die Gebäude einen Anteil von 1,44 %.

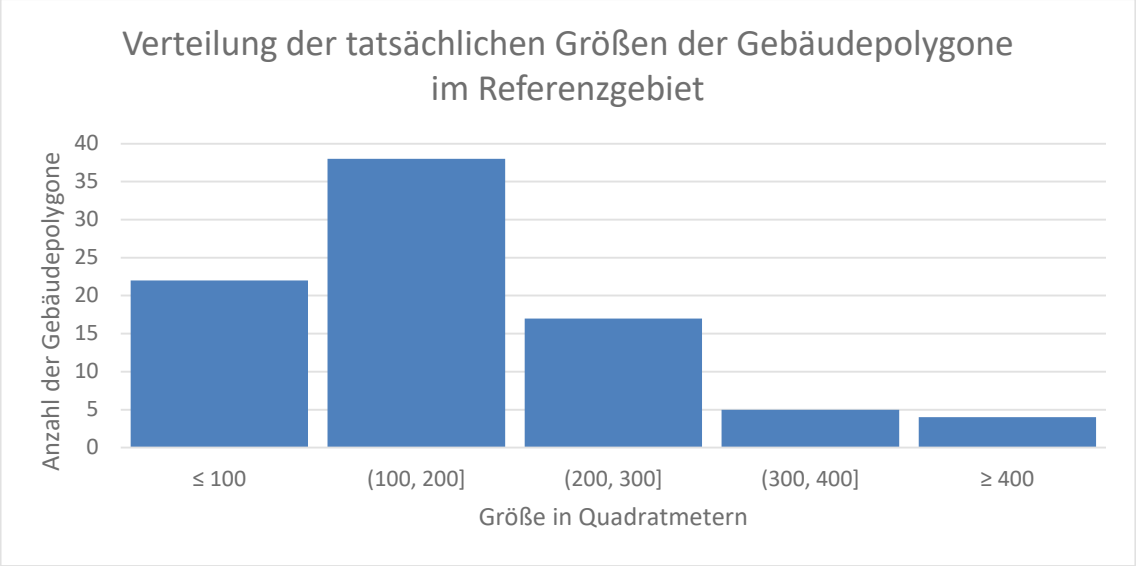


Abbildung 33: Verteilung der tatsächlichen Gebäudegrößen in Quadratmeter für das Referenzgebiet (eigene Erstellung)

Im nächsten Schritt werden diese Größen zunächst mit den ‚True-Positive'-Pixeln verglichen. Werden die nachfolgenden Abbildungen einander gegenübergestellt, ergibt sich bereits ein drastischer Unterschied, der auch die Ergebnisse der optischen Bewertung bestätigt.

Das RGB-Modell hat im Referenzgebiet insgesamt 87 Gebäudepolygone und somit exakt die Anzahl an Gebäuden gefunden, die auch bei den Referenzdaten vorhanden ist. Insgesamt machen die Gebäudepolygone des RGB-Modells einen Anteil von 1,35 % am gesamten Referenzgebietes aus, wodurch der Anteil sehr ähnlich zu den Referenzdaten ist. Mit einem IoU-Score von 0,88 ist die Genauigkeit der durch das Modell berechneten Segmentierungen sehr hoch bzw. nahezu perfekt. Die Verteilung der Polygongrößen (siehe Abbildung 32) ist beinahe identisch mit derjenigen bei den Referenzdaten.

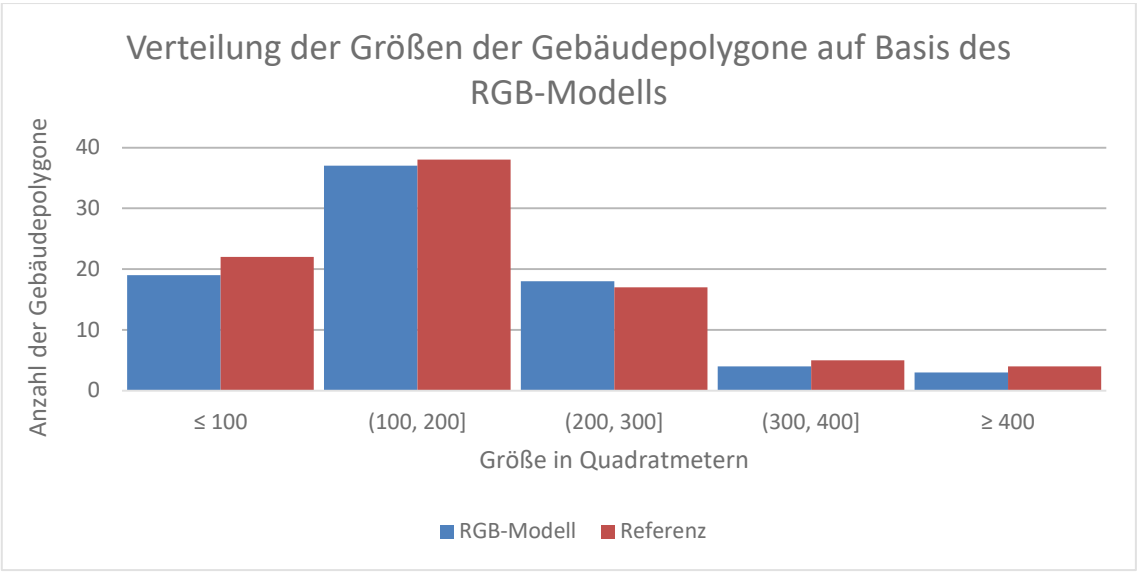


Abbildung 34: Verteilung der Gebäudegrößen in Quadratmeter auf Basis des RGB-Modells (eigene Erstellung)

Bei den RGBI-Daten sieht die Klassifizierung ähnlich aus. Insgesamt wurden vom RGBI-Modell 86 Gebäude gefunden und auch nahezu vollständig richtig klassifiziert. Die Gebäude haben eine durchschnittliche Grundfläche von 157 Quadratmetern, wodurch dieser Wert schon nahe am realen Durchschnitt liegt. Auch die Verteilung der Größen auf Basis des RGBI-Modells ist im Vergleich zu den tatsächlichen Größen nahezu identisch. Mit einem Anteil von 1,34 % liegt auch dieser Wert nahe bei demjenigen der Referenzdaten. Mit einem IoU-Score von 0,88 ist das Genauigkeitsmaß ebenso hoch wie beim RGB-Modell, was die guten Werte des Trainings erneut bestätigt. Die Genauigkeit der Überlappung zwischen den berechneten Gebäudepolygonen und den tatsächlichen manuell eingezeichneten Polygonen ist somit ebenso sehr hoch. Lediglich der Anteil der Gebäudepolygone zu der Gesamtfläche unterscheidet sich geringfügig von dem des RGB-Modells.

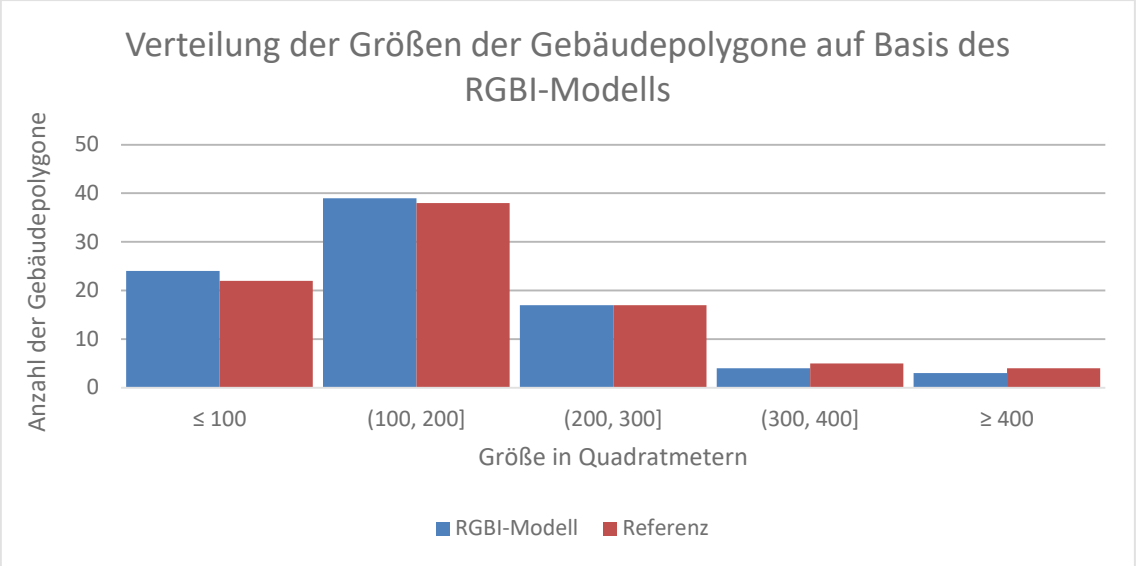


Abbildung 35: Verteilung der Gebäudegrößen in Quadratmeter auf Basis des RGBI-Modells (eigene Erstellung)

Bei den berechneten Gebäudegrößen auf Basis des RGBI- und nDSM-Modells ergibt sich ein völlig anderes Bild. Viele Gebäude sind aufgrund von fehlerhaften Einschätzungen in mehrere kleine Polygone aufgeteilt worden und bilden dadurch kein durchgehendes Polygon für jedes Gebäude. Insgesamt sind 116 Gebäudepolygone entstanden, wodurch der Wert stark von der tatsächlichen Anzahl der Gebäude abweicht. Bei der Betrachtung der Verteilung der Größen ergibt sich ein ebenso falsches Bild. Ein Großteil der Polygone hat unter 100 Quadratmeter und im Durchschnitt hat ein Gebäude 67 Quadratmeter. Im Gegensatz zu den anderen Klassifikationen hat kein einziges Gebäude über 400 Quadratmeter. An der gesamten Größe des Referenzgebiets gemessen haben die Gebäudeflächen lediglich einen Anteil von 0,8 %, was stark vom tatsächlichen Wert abweicht.

Wie bereits bei den beiden vorherigen Modellen wurde der IoU-Score berechnet. Dieser liegt bei 0,52 und ist damit ausgesprochen niedrig. Die Lage der berechneten Segmentierungen entsprechen damit kaum den tatsächlichen Gebäudepolygonen.

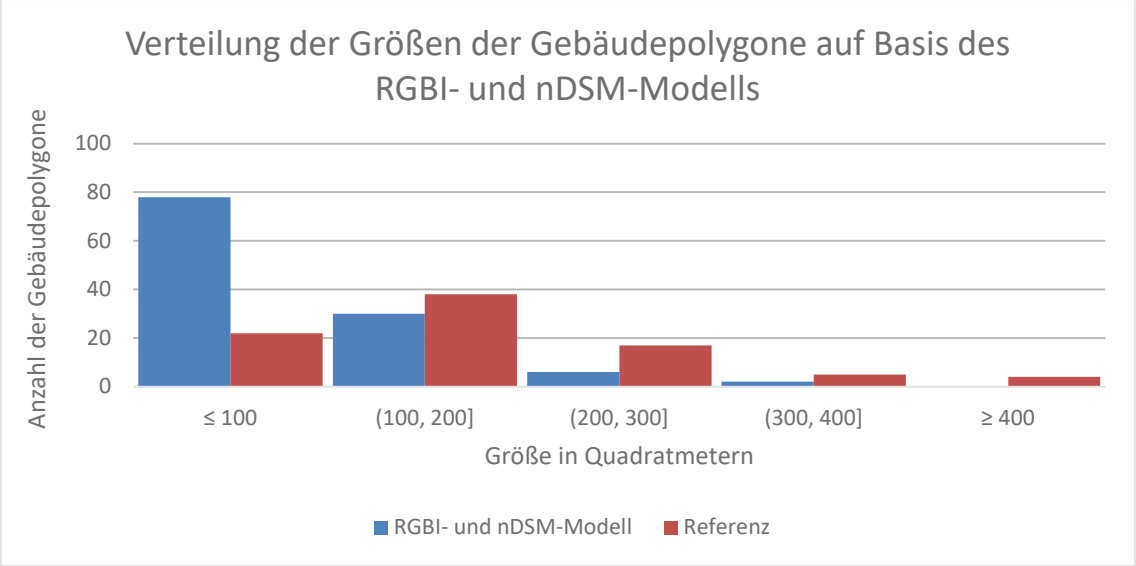


Abbildung 36: Verteilung der Gebäudegrößen in Quadratmetern auf Basis des RGBI- und nDSM-Modells (eigene Erstellung)

Anschließend können für dieses Orthofoto noch jene Flächen betrachtet werden, auf denen kein Gebäude erkannt wurde, aber eigentlich eines steht (‚False-Negative'-Pixel). Auch hier ergibt sich ein ähnliches Bild wie zuvor. Während beim RGBI-Modell im Wesentlichen nur kleine Flächen falsch klassifiziert wurden (der Großteil von ihnen hat eine Größe von unter 10 Quadratmeter und keines ist größer als 100 Quadratmeter), fallen die Ergebnisse beim RGBI- und nDSM-Modell wesentlich schlechter aus. Hier wurden teilweise ganze Gebäude mit über 400 Quadratmetern falsch klassifiziert.

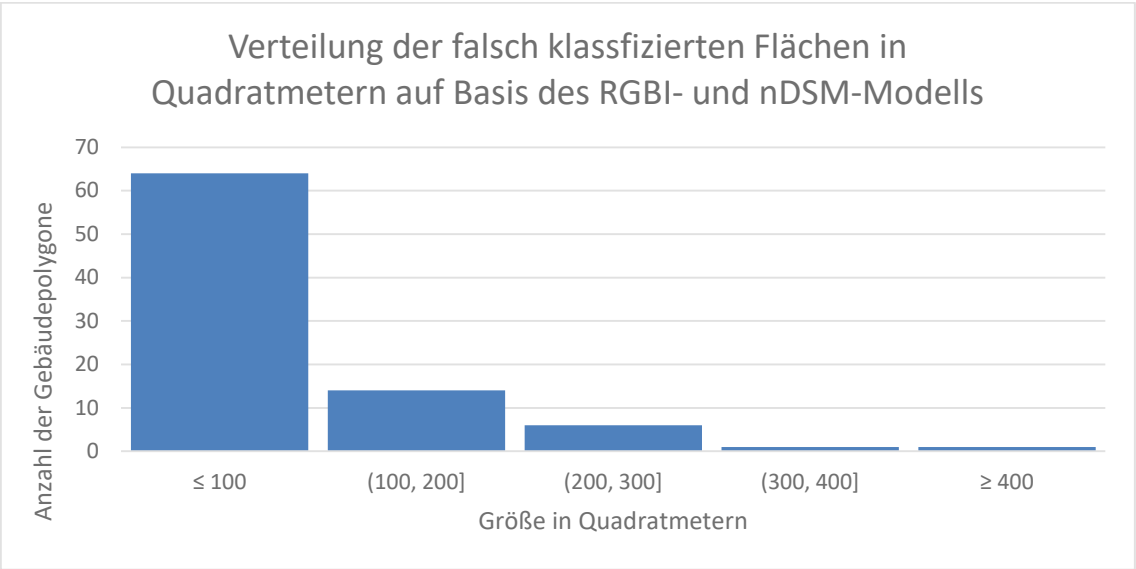


Abbildung 37: Verteilung der falsch klassifizierten Flächen in Quadratmetern auf Basis des RGBI- und nDSM-Modells (eigene Erstellung)

Abschließend kann für dieses Orthofoto festgehalten werden, dass die Modellwahl eindeutig ausfällt. Obwohl die Ergebnisse des RGBI- und nDSM-Modells während des Trainings vielversprechend wirkten, haben sie sich als vollkommen ungeeignet für neue unbekannte Daten erwiesen. Dabei war das Modell nicht nur geringfügig schlechter,

sondern es zeigte eindeutige Schwächen bei den Gebäuden. Obwohl das Modell bereits bei der Gebäudeklasse ungeeignet war und somit nicht verwendet werden kann, wird es weiterhin für die Überprüfungen verwendet. Damit soll geprüft werden, ob es mit anderen Klassen besser umgehen kann oder ob die Ergebnisse ähnlich wie bei den Gebäuden sind.

Die Wahl zwischen dem RGB- und dem RGBI-Modell ist komplizierter und verlangt nach einer weiteren Austestung mithilfe von anderen Referenzdaten. Dazu wird zunächst die Gewässerklasse herangezogen. Bei der Betrachtung derselben werden große Unterschiede ersichtlich.

Die manuell eingezeichneten Gewässerpolygone im Referenzgebiet haben einen Anteil von 2,43 % an der gesamten Fläche. Sie setzen sich aus einem See und einem Bach zusammen. Ein großer Teil des Baches liegt im Schatten oder ist von Bäumen überdeckt und ist dadurch kaum erkennbar. Insbesondere das RGB-Modell hat in diesen Bereichen große Probleme mit der Klassifizierung. Mit einem IoU-Score von 0,65 ist der Wert im Vergleich zu den anderen Ergebnissen sehr niedrig. Der Anteil an der Gesamtfläche beträgt 2,0 % und ist somit wesentlich geringer als bei den Referenzdaten. Das RGBI-Modell hat hier wesentlich besser abgeschnitten und einen IoU-Score von 0,89 erreicht, wodurch dieser erneut nahezu perfekt ausfällt. Auch der Anteil ist mit einer Höhe von 2,35 % wesentlich näher an den Referenzpolygonen als das RGB-Modell. In dem Fall profitiert das Modell offensichtlich von dem zusätzlichen Infrarotkanal.

Im Vergleich zu den Gebäudepolygonen hat bei den Gewässerpolygonen insbesondere das RGBI- und nDSM-Modell ausgezeichnet abgeschnitten. Es erreicht im Referenzgebiet einen IoU-Score von 0,91, der somit höher ist als derjenige des RGBI-Modells. Der Grund liegt wahrscheinlich darin, dass es bei Gewässern keine großen Höhenunterschiede gibt. Dadurch kann das Modell nicht nur von den RGBI-Daten, sondern zusätzlich von den nDSM-Daten profitieren und erzeugt eine genauere Klassifikation. Der Anteil ist mit 2,33 % etwas geringer als bei den Ergebnissen des RGBI-Modells.

Für die Gewässerpolygone kann zusammenfassend festgehalten werden, dass hier insbesondere das Modell mit den zusätzlichen nDSM-Daten gut abgeschnitten hat. Nichtsdestotrotz kann es aufgrund der ausgeprägten Schwächen bei den Gebäudepolygonen nicht verwendet werden.

Bei der Betrachtung der versiegelten Flächen ergibt sich ein ähnliches Bild wie bei den Gebäudepolygonen. Die manuell eingezeichneten Straßenpolygone (Versiegelte Flächen) haben einen Anteil von 3,09 % an der Fläche des Referenzpolygons. Während das RGB-Modell und das RGBI-Modell den gleichen IoU-Score von 0,82 haben, stürzt das RGBI- und nDSM-Modell auf einen Wert von 0,74 ab und ist somit wesentlich schlechter. Wird der Anteil an der Gesamtfläche betrachtet, haben das RGB- und das RGBI-Modell erneut dieselben Werte von 2,85 %. Das Modell, das auch das nDSM benutzt hat, hat einen Anteil von 2,73 %, der unter demjenigen der anderen beiden Modelle liegt.

Abschließend kann somit gesagt werden, dass das RGBI-Modell am besten für die Klassifizierung von Orthofotos geeignet ist. Es hat bei allen drei überprüften Klassen gute

und akkurate Ergebnisse produziert und weist keine instabilen Genauigkeitsmaße auf. Das RGB-Modell scheint für manche Klassen ebenso zu funktionieren, jedoch hat es Schwierigkeiten bei der Klassifizierung von Gewässern. Das RGBI- und nDSM-Modell hingegen hat nahezu keinen Verwendungszweck, da die Ergebnisse nicht zuverlässig genug sind. Das RGBI-Modell profitiert vom zusätzlichen nahen Infrarotkanal und kann diesen auch erfolgreich nutzen (z.B. bei Gewässern).

Werden nun die Ergebnisse des RGBI-Modells mit der Landcover-Klassifikation des BEV verglichen, ergeben sich zum Teil sehr unterschiedliche Ergebnisse. Generell tendiert das Modell des BEV eher dazu, dass es kleine Gebäude nicht als solche klassifiziert. Deswegen sind im Referenzgebiet auch nur 74 Gebäude insgesamt erkannt worden. Auch werden oft zu viele Pixel als Gebäude klassifiziert, die jedoch keine Gebäude sind. Daraus ergibt sich ein IoU-Score von 0,74. Das RGBI-Modell hat hingegen einen IoU-Score von 0,88. Anhand dieser Werte ist zu sehen, dass die Gebäudepolygone des CNNs im Referenzgebiet somit lagegenauer als die vom BEV sind.

Werden die Gewässerflächen betrachtet, ergibt sich ein ähnliches, aber nicht so deutliches Bild. Der IoU-Score des BEV ist ähnlich hoch wie beim RGBI-Modell (0,84 und 0,89). Lediglich bei dem Bach sind bei den Landcover-Daten des BEV Probleme zu finden und oft wird dort nicht die richtige Klasse gefunden. Dabei ist jedoch zu erwähnen, dass das Modell des BEV für die Klassifikation von Gewässern auf DLM-Daten zurückgreifen muss. Im Gegensatz dazu kommt das CNN ausschließlich mit RGBI-Daten zurecht und kann mithilfe dieser auch Gewässer klassifizieren.

Die Unterscheidung zwischen versiegelten und unversiegelten Flächen hat in den meisten Fällen funktioniert. In ein paar wenigen Fällen kann das Modell keine genaue Unterscheidung machen. Hierbei ist jedoch zu erwähnen, dass bereits eine visuelle Einteilung zwischen unversiegelt und versiegelt oft schwierig oder nicht möglich ist. Unter rein visuellen Aspekten erscheint die Unterscheidung jedoch legitim und in weiten Teilen zuverlässig. Derselbe Sachverhalt gilt für die Unterscheidung von Nadel- und Laubbäumen. Auch hier ist eine Differenzierung oft kompliziert. Zusätzlich sind im Referenzgebiet kaum Laubbäume vorhanden. Am Flussufer erkennt das Modell hin und wieder Laubbäume, die auch als solche erscheinen. Die Wälder hingegen werden alle als Nadelwälder klassifiziert, was auch der Wirklichkeit entspricht.

5.3.2. Ergebnisse des zweiten Orthofotos (Imst)

Im nächsten Schritt wurde das Orthofoto in Tirol mithilfe des RGBI-Modells klassifiziert. Auch wurde hier ein Referenzgebiet mit 1000 × 1000 Metern erstellt und die vorherigen Klassen manuell eingezeichnet. Daraus werden dieselben Maße wie beim ersten Orthofoto berechnet. Für einen abschließenden Vergleich werden im Anschluss erneut die Daten des BEV herangezogen und mit den Ergebnissen des CNNs verglichen. Im Unterschied zum ersten Orthofoto wurde das Modell nicht mit Daten aus dieser Gegend trainiert. Die Orthofotos wurden somit von einem anderen Unternehmen und zu einem anderen Zeitpunkt aufgenommen als jene in Mariazell.

Im Referenzgebiet sind insgesamt 124 Gebäude vorhanden, wobei die dazugehörigen Gebäudepolygone eine durchschnittliche Fläche von 228 Quadratmetern umfassen. Somit sind die Gebäude sowohl im Hinblick auf die Anzahl als auch bezüglich der durchschnittlichen Grundfläche größer als diejenigen in Mariazell. Sie haben an der Fläche des Referenzgebietes einen Anteil von 2,8 %, wodurch dieser Wert wesentlich höher als im Bereich von Mariazell ausfällt. Insgesamt betrachtet bedecken die Gebäude im Referenzgebiet eine Fläche von 28 387 Quadratmetern. Die Verteilung der Größen der Gebäudepolygone sieht ähnlich wie beim ersten Orthofoto aus.

Die Klassifikation des gesamten Orthofotos mit dem CNN erzeugt dabei ähnliche Ergebnisse. Insgesamt wurde eine Fläche von 26 663 Quadratmetern korrekt der Gebäudeklasse zugeordnet. Somit ist der Wert bereits ähnlich hoch wie bei den Referenzdaten. Mit einem Anteil von 2,6 % an der gesamten Fläche ist dieser Wert ebenso etwas geringer. Ein Problem bei dem Orthofoto sind Schattenbereiche auf Dächern. Obwohl diese Bereiche eigentlich zum Gebäude gehören und somit der Klasse ‚Gebäude' zugeteilt werden sollten, werden sie als ‚Sonstiges' klassifiziert. Dadurch entstehen keine durchgehenden Gebäudepolygone, sondern zwei oder mehrere getrennte Polygone pro Gebäude. Insgesamt sind durch den Klassifizierungsvorgang und ohne eine anschließende Überarbeitung 139 Gebäudepolygone entstanden. Der IoU-Score für das CNN beträgt 0,77 und ist somit schlechter als im Bereich von Mariazell (0,88). Die Lagegenauigkeit der Gebäudepolygone ist also geringer als zuvor. Allerdings ist der neue IoU-Score noch immer hoch und zeugt davon, dass das CNN gut generalisieren und mit unbekannten Daten umgehen kann.

Werden die falsch klassifizierten Ergebnisse betrachtet, lassen sich große Fehler in manchen Bildbereichen im Referenzgebiet entdecken. Durchschnittlich wurde eine Fläche von 25 Quadratmetern irrtümlich der Gebäudeklasse zugeordnet, obwohl der Pixel nicht diese Klasse repräsentiert (‚False-Positive'-Pixel). Insgesamt wurden 5863 Quadratmeter der falschen Klasse zugeordnet. Insbesondere in den Randbereichen der Gebäude hat das CNN Probleme und tendiert dazu, zu viele Pixel als Gebäude zu klassifizieren. Ebenso große Schwierigkeiten hatte das Modell bei einem Tennisplatz und bei einem großen asphaltierten Parkplatz. Werden die ‚False-Negative'-Pixel betrachtet, also das Gegenstück zu der oben genannten Art, entsteht ein weniger fällt das Ergebnis weniger hoch aus. Insgesamt 1754 Quadratmeter wurden einer anderen Klasse als der Gebäudeklasse zugeordnet, obwohl diese die richtige Kategorie gewesen wäre. Im Durchschnitt beträgt die Fläche dieser Polygone 14,14 Quadratmeter und ist somit kleiner als bei den ‚False-Positive'-Pixeln.

Im Vergleich zur Landcover-Klassifikation des BEV schneidet die Klassifikation des CNNs schlechter ab. Die Klassifikation des BEV erreicht einen IoU-Score von 0,81 und ist somit höher als beim CNN. Während beim CNN zu viele Gebäudepolygone entstanden sind, wurden bei den Daten des BEV insgesamt 105 Gebäudepolygone im Referenzbereich gefunden. Im Durchschnitt haben die Gebäudepolygone eine Größe von 255 Quadratmetern. Wie bereits bei dem vorherigen Orthofoto tendiert das Modell des BEV dazu, kleine Gebäude nicht als solche zu klassifizieren. Dabei ist jedoch

anzumerken, dass es manuell überarbeitet und kontrolliert wird. Die Ergebnisse des CNNs hingegen sind die Originaldaten, die durch das Modell berechnet wurden und somit den Daten des BEV ähnlich. Die Klassifikation des BEV hat ebenso Probleme mit den Randbereichen des Gebäudes und tendiert ähnlich wie das CNN dazu, zu viele Pixel der Gebäudeklasse zuzuordnen (‚False-Positive'-Pixel). Die Probleme beim Parkplatz und Tennisplatz hingegen sind bei den Daten des BEV nicht vorhanden. In Summe wurden 4637 Quadratmeter der Gebäudeklasse zugeordnet, obwohl die Pixel keine Gebäude sind. Somit ist der Wert um mehr als 1000 Quadratmeter geringer als bei der Klassifizierung des CNNs. Die ‚False-Negative'-Pixel sind ähnlich wie beim CNN, sie fallen jedoch geringer aus.

Für die Gebäudeklasse kann somit zusammenfassend festgestellt werden, dass die Ergebnisse überraschend gut sind. Obwohl das CNN-Modell mit vollkommen anderen Daten trainiert wurde, hat es erfolgreich alle Gebäude gefunden. Im Gegensatz zum ersten Orthofoto fallen die Ergebnisse für die Gebäudeklasse bei der Klassifikation des BEV geringfügig besser aus.

Abschließend wurden für dieses Orthofoto noch eine Überprüfung der Genauigkeit bei den Gewässern im Referenzgebiet und zusätzlich ein Vergleich zu den Daten des BEV durchgeführt. Bei einer visuellen Überprüfung ist schnell zu sehen, dass das Modell insbesondere in den Schattenbereichen noch großes Verbesserungspotenzial besitzt. Hier kommt es, wie bereits bei der Gebäudeklasse, oftmals zu einer Verwechslung mit der Klasse ‚Sonstiges'. Auf der nachfolgenden Abbildung sind mehrere solcher Bereiche zu sehen. Die rot eingerahmten Flächen sind Bildbereiche, die nicht der Klasse ‚Gewässer' zugeordnet wurden, obwohl diese in den Bereichen eindeutig vorhanden ist. Mit der Klassifikation wurde ein IoU-Score von 0,80 erreicht, was bereits ein guter Wert für die unbekannten Daten ist.

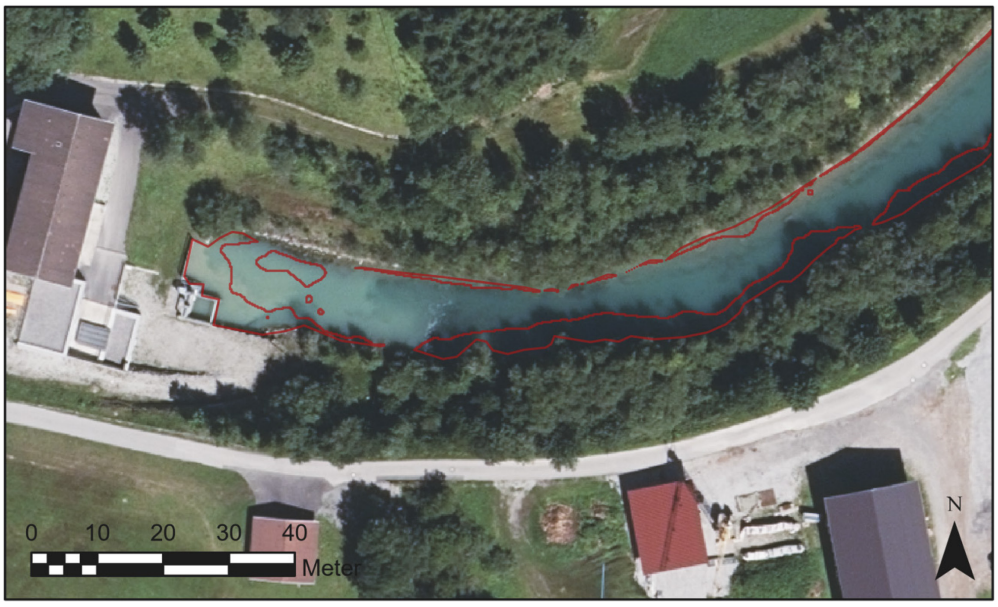


Abbildung 38: Durch das CNN falsch klassifizierte Bereiche (eigene Erstellung, Orthofoto BEV)

Die Ergebnisse des BEV hingegen sind wesentlich genauer. Wie bereits im vorherigen Kapitel beschrieben, basiert die Klassifikation der Gewässer im BEV auf den Daten des digitalen Landschaftsmodells. Diese Polygone werden bis zu einem gewissen Detailgrad manuell erfasst und dann zur Klassifikation der Gewässer übernommen. Dadurch wird ein IoU-Score von 0,89 erreicht, also eine fast lagerichtige Klassifizierung der Gewässer. Im Gegensatz dazu kommt das CNN ohne Zusatzinformationen aus und erreicht dennoch gute Ergebnisse. Wie in der nachfolgenden Abbildung zu sehen ist, sind die Flächen für denselben Bereich wie in der vorherigen Abbildung, die falsch klassifiziert wurden, wesentlich kleiner und geringer in der Anzahl.

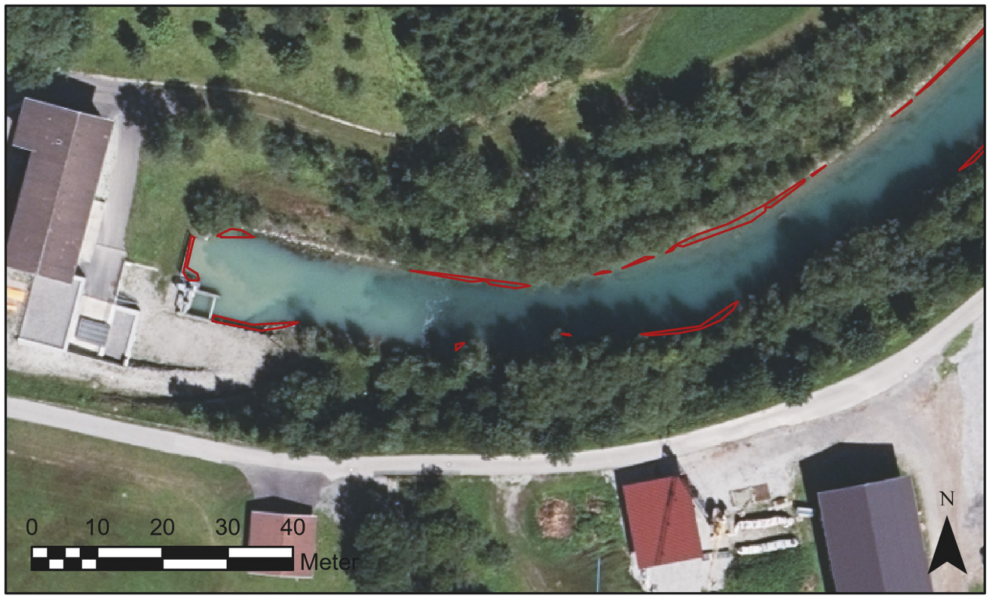


Abbildung 39: Falsch klassifizierte Bereiche bei den Daten des BEV (eigene Erstellung, Orthofoto BEV)

Die versiegelten Straßen werden in diesem Fall nicht mehr quantitativ bewertet, da die visuellen Ergebnisse sehr gut aussehen und in den meisten Fällen korrekt sind. Die Unterscheidung zu den unversiegelten Flächen ist zum größten Teil stimmig. Zum Beispiel werden alle Forstwege als unversiegelte Straßen erkannt, was auch der Wahrheit entspricht. Alle großflächigen Wälder werden als Nadelwälder erkannt, was ebenso der Wahrheit entspricht. Mit Laubbäumen scheint das Modell jedoch Probleme zu haben. Hier werden kaum welche erkannt und zu viele als Nadelwald klassifiziert.

Abschließend kann für das zweite Orthofoto gesagt werden, dass die Ergebnisse trotz der neuen und unbekannten Daten gut ausfallen. In vielen Fällen kann das CNN mit dem Algorithmus des BEV mithalten und erzeugt Ergebnisse, die nur ein wenig schlechter sind. Das ist vor allem auf das fehlende Training in der Umgebung zurückzuführen und könnte durch ein erweitertes Training verbessert werden. Der Fokus beim Training sollte dabei insbesondere auf den schattigen Bereichen liegen, damit das CNN lernt, wie es damit umgehen soll.

5.3.3. Ergebnisse des dritten Orthofotos (Kufstein)

Bei diesem Orthofoto wird auf dieselben Methoden wie bereits bei den ersten beiden Orthofotos gesetzt. Der einzige Unterschied liegt darin, dass aufgrund der dichten

Besiedlung ein Referenzgebiet mit der Größe von 500 × 500 Metern gewählt wird. Der Grund dafür ist, dass die genaue manuelle Klassifikation eines Gebietes mit einer dichten Besiedlung viel Zeit für in Anspruch nimmt. Zudem sind in dem Gebiet bereits ausreichend viele Gebäude vorhanden und es kommen Gewässer, Straßen und wenige Bäume vor.

Die Gebäude stehen dicht aneinander und bilden große zusammenhängende Gebäudeblöcke. Bei den Gebäuden sind kaum Grünflächen vorzufinden und nur vereinzelt gibt es Bäume. Die Gebäudepolygone im Referenzgebiet haben dabei eine durchschnittliche Größe von 923 Quadratmetern und sind somit deutlich größer als in den bisherigen klassifizierten Orthofotos. Obwohl die Fläche des Referenzgebietes wesentlich kleiner als bei den anderen Beispielen ist, sind insgesamt 92 Gebäudepolygone im Referenzgebiet. Auch die Dachstrukturen unterscheiden sich deutlich von denen auf den anderen Orthofotos. Dabei sind viele Flachdächer mit Sonnenkollektoren und Photovoltaikanlage ausgestattet und einige der Hausdächer sind begrünt. Durch die größeren Polygone bzw. Häuser ist die Verteilung der Größen der Polygone im Vergleich zu den ersten beiden Orthofotos eine vollkommen andere. Wie man in Abbildung 40 erkennen kann, sind ein Großteil der Gebäudepolygone unter 1000 Quadratmeter groß. Es gibt jedoch ein paar wenige Polygone, die wesentlich größer sind und sogar drei, die über 5000 Quadratmeter umfassen.

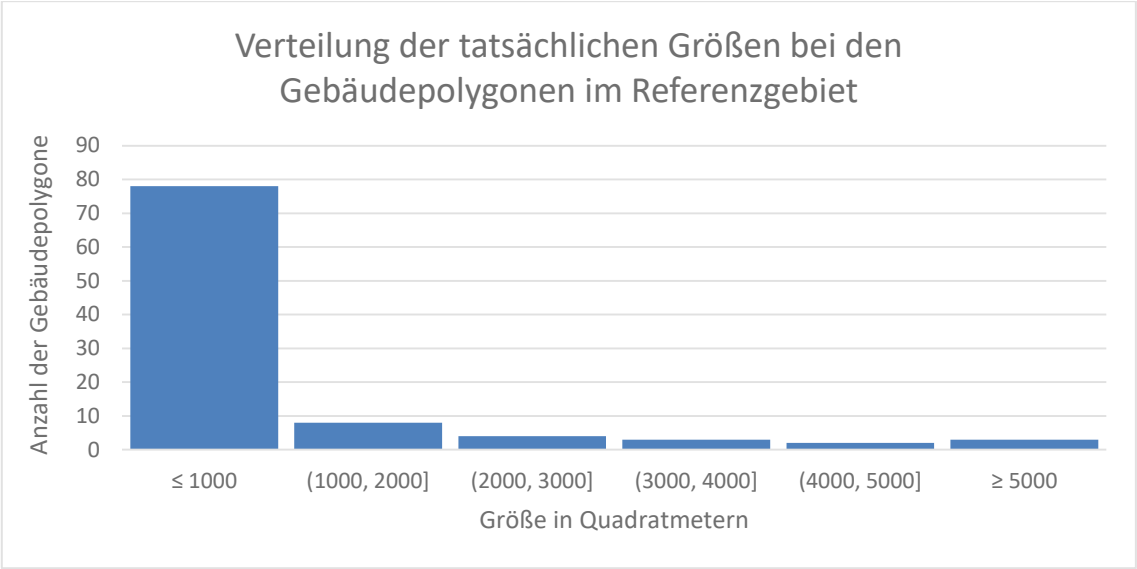


Abbildung 40: Verteilung der Größen der Gebäudepolygone auf Basis der Referenzdaten (eigene Erstellung)

Wird nun die Klassifikation des CNNs betrachtet, ergibt sich ein vollkommen anderes Bild. Insbesondere bei den großen Gebäudeblöcken und den begrünten Dächern gibt es große Probleme bezüglich der Klassifikation. In den meisten Fällen werden diese Flächen mit der Klasse ‚Sonstiges' oder ‚Versiegelte Flächen' verwechselt. Zudem erkennt das neuronale Netzwerk oft keine ganzen Gebäude, sondern nur einzelne kleine Flächen, die es als solche klassifiziert. Dadurch entstehen viele kleine Polygone, wodurch sich insgesamt 259 Gebäudepolygone ergaben. Im Durchschnitt sind diese 227 Quadratmeter groß, weshalb sie wesentlich kleiner als bei den Ground-Truth-Daten sind. Keines der Polygone ist dabei größer als 2900 Quadratmeter. Daraus lässt sich schließen, dass das

Modell große Probleme mit diesen Gebäuden hat, da sie in den Trainingsdaten nicht vorgekommen sind. Zudem wurden Teile von Straßen und Gewässern als Gebäude klassifiziert (‚False-Positive'-Pixel), wobei die Größe dieser mit durchschnittlich 45 Quadratmetern wesentlich geringer ausfällt. Bei der Betrachtung der ‚False-Negative'-Pixel bestätigt sich erneut, dass das Modell mit großen Gebäuden erhebliche Probleme hat, da die meisten von ihnen in diese Klasse fallen.

Mit einem IoU-Score von 0,55 fällt der Wert wesentlich geringer als bisher aus und spiegelt auch die oben genannten Zahlen wider. Die Überlappung ist somit ausgesprochen gering und das Modell sollte in diesem Fall nicht verwendet werden.

Im Gegensatz dazu steht die Klassifikation des BEV. Bei dieser fallen die Ergebnisse wesentlich präziser aus und erreichen einen IoU-Score von 0,82. Somit ist der Wert deutlich höher als bei der Klassifizierung durch das CNN. Die Verteilung der Gebäudepolygone, die beim BEV korrekt klassifiziert wurden, ist nahezu ident zu den Referenzdaten. Die Gebäudepolygone haben im Referenzgebiet dabei eine durchschnittliche Größe von 911 Quadratmetern, wodurch das Ergebnis bereits sehr nah an den Referenzdaten liegt.

Abschließend wird noch die Klasse ‚Gewässer' betrachtet und bewertet. Im Referenzgebiet sind zwei große Gewässerpolygone zu finden. Bei der Klassifizierung durch das CNN entsteht ein ähnliches Bild wie bei den Gebäuden. Das Modell kann nahezu keinen einzigen Pixel richtig zuordnen. In diesem Fall versagt das Modell und auf Basis der Flächen entsteht ein IoU-Score von 0,002. Es wurde somit keine Überlappung erzeugt.

Im Vergleich dazu steht erneut die Klassifizierung des BEV. Diese hat aufgrund der Daten aus dem digitalen Landschaftsmodell eine gute Klassifizierung erzeugt und erreicht einen IoU-Score von 0,98. Dieser Wert bedeutet eine nahezu idente Klassifizierung im Vergleich zu den Referenzdaten und stellt somit einen optimalen Wert dar, der kaum mehr verbessert werden kann. Im Vergleich zum CNN hat die Klassifikation des BEV somit Ergebnisse hervorgebracht, die verwendet werden können.

Die einzeln stehenden Bäume wurden oft als Laubbäume erkannt, jedoch wurden hier ebenso oft falsche Ergebnisse erzeugt. Dadurch wurden die Laubbäume oft mit der Klasse ‚Sonstiges' verwechselt. Andere Klassen wie ‚Unversiegelte Flächen' oder ‚Nadelbäume' kamen im Referenzgebiet nicht vor und wurden auch nicht erkannt.

5.3.4. Ergebnisse des vierten Orthofotos (Wienerwald)

Wie bereits in der Einleitung zu diesem Kapitel erwähnt, wird hier nur eine visuelle Überprüfung vorgenommen. Dies dient im Wesentlichen zur Überprüfung, ob das Modell auch Laubwälder bzw. Laubbäume klassifizieren kann, da diese in den anderen Orthofotos kaum vorgekommen sind. Die anderen Klassen werden nur kurz zusammengefasst und visuell bewertet.

Das auf dem Orthofoto abgebildete Gebiet befindet sich im Osten von Wien im Wienerwald und ist mit vielen Einfamilienhäusern bebaut. Im Gegensatz zum vorherigen

Orthofoto kann das CNN hier erfolgreich Gebäude klassifizieren. Auch die versiegelten Flächen werden eindeutig erkannt. Beide Klassen werden in den meisten Fällen pixelgenau eingezeichnet und es werden gute Ergebnisse erzielt. Im Orthofoto befinden sich viele einzeln stehende Laubbäume und Laubwälder, die auch als solche erkannt werden. Die Bäume werden teilweise pixelgenau verortet und ebenso wie die Wälder sehr genau eingezeichnet. Lichtungen im Waldgebiet werden ebenso erfolgreich der Klasse ‚Sonstiges' zugeteilt und sind eindeutig als solche zu erkennen.

Abschließend kann zu dem Orthofoto festgehalten werden, dass die Klassifikation von Laubwäldern in dem Gebiet gut funktioniert. Das kann jedoch auch wieder darauf zurückgeführt werden, dass die Trainingsdaten aus einem Orthofoto in der Nähe dieses Orthofotos entstanden sind. Dadurch war es für das Modell, wie beim ersten Orthofoto, kein Problem, die Klassen erfolgreich zu erkennen.

5.4. Zusammenfassung der Ergebnisse

Für die Zusammenfassung werden ausschließlich die Ergebnisse des RGBI-Modells herangezogen, da es als einziges auf alle vier Orthofotos angewandt wurde. Aufgrund der schlechteren Ergebnisse der beiden anderen Modelle, wurden diese nach dem ersten Orthofoto nicht mehr weiterverwendet. Folgende Ergebnisse sind mit dem RGBI-Modell erzielt worden:

Tabelle 12: Überblick über die Ergebnisse des RGBI-Modells bei den vier verwendeten Orthofotos (eigene Erstellung)

	Mariazell	Imst	Kufstein	Wien
Gebäude	Genaue Ergebnisse	Größtenteils genaue Ergebnisse. Seltene Verwechslungen mit versiegelten Flächen.	Ungenaue Ergebnisse. Nur wenige Gebäude wurden richtig erkannt.	Genaue Ergebnisse
Versiegelte Flächen	Genaue Ergebnisse	Größtenteils genaue Ergebnisse.	Ungenaue Ergebnisse	Genaue Ergebnisse
Unversiegelte Flächen	Genaue Ergebnisse. Einzelne Verwechslungen mit versiegelten Flächen.	Genaue Ergebnisse. Einzelne Verwechslungen mit versiegelten Flächen.		
Gewässer	Genaue Ergebnisse	Größtenteils genaue Ergebnisse. Im Schatten kommt es aber oft zu Verwechslungen mit anderen Klassen.	Ungenaue Ergebnisse. Alle Gewässer wurden mit anderen Klassen verwechselt.	Genaue Ergebnisse

	Mariazell	Imst	Kufstein	Wien
Nadelbäume	Genaue Ergebnisse	Genaue Ergebnisse		
Laubbäume				Genaue Ergebnisse
Sonstiges	Genaue Ergebnisse	Genaue Ergebnisse	Ungenaue Ergebnisse	Genaue Ergebnisse

6. Zusammenfassung

In der vorliegenden Masterarbeit wurde die Landcover-Klassifikation von Orthofotos mithilfe einer künstlichen Intelligenz, genauer gesagt eines CNNs, erläutert. Dabei wurde nicht nur evaluiert, ob ein solcher Prozess damit ermöglicht werden kann, sondern auch, ob die Ergebnisse mit anderen aktuellen Methoden vergleichbar sind. Als Datengrundlage dienten Orthofotos und Geländemodelle, die vom BEV bereitgestellt wurden. Für den Vergleich der Ergebnisse des CNNs mit einer anderen Klassifikationsmethode wurden ebenfalls Daten des BEV verwendet. Dort wird zurzeit auf eine objektbasierte Klassifikationsmethode gesetzt, bei der viele verschiedene Daten, zum Beispiel Orthofotos, Indizes auf Basis der Orthofotos, Gelände- und Höhenmodelle, für eine Landcover-Klassifikation benutzt werden. Zusätzlich wurden für die Kontrolle eigene Referenzdaten erstellt.

Für die Programmierung wurde Python verwendet, wobei für die Programmierung des CNN auf das Framework *TensorFlow* gesetzt wurde, das als Open-Source-Paket zur Verfügung steht. Weiters wurden häufig die Programmbibliotheken *RasterIO* und *NumPy* benutzt. Für den abschließenden Vergleich der Ergebnisse wurde aufgrund der Effizienz und der Visualisierungsmöglichkeiten auf die Software ArcGIS Pro von ESRI zurückgegriffen.

Folgende Fragestellungen sollten durch diese Masterarbeit beantwortet werden:

I. Ist der Einsatz eines CNNs für die Klassifizierung von Orthofotos geeignet und kann das Modell auch auf große Gebiete ausgeweitet werden?
II. Welche Parameter sind für den Einsatz eines CNNs von Relevanz und wie sehen diese aus?
III. Wie viele Trainingsdaten sind notwendig und welche zusätzlichen Klassen können daraus gewonnen werden?
IV. Führt der Einsatz eines CNNs zu einer genaueren Klassifikation?

Alle vier Fragen wurden in eigenen Kapiteln behandelt und der Reihe nach beantwortet

6.1. Landcover-Klassifikation mithilfe von CNNs

Bei CNNs handelt es sich um künstliche neuronale Netzwerke, die sich insbesondere für die Klassifikation von Bildern eignen. Dazu müssen sie zunächst mit Trainingsbildern und dazu passenden Labels oder Masken trainiert werden. Während des Trainings werden Filter erzeugt und so angepasst, dass sie bestimmte Bildinhalte erkennen und klassifizieren können. Dabei kann nicht nur das gesamte Bild klassifiziert werden, sondern auch mehrere Klassen auf einem einzigen Bild. Im Zuge dieses Vorgangs können die Bilder auch bereits segmentiert werden. Die Segmente können abschließend bestimmten Klassen zugeteilt werden.

Dieser stark verkürzt beschriebene Vorgang wurde auch für diese Masterarbeit genutzt. Zunächst wurde auf Basis von Orthofotos eine manuelle Klassifizierung in ArcGIS Pro durchgeführt. Auf dieser Grundlage wurden die Labels erzeugt, die für das spätere Training gebraucht wurden. Die Orthofotos und die dazu passenden Labels wurden

anschließend in kleine Bilder zerteilt und zum Trainieren des CNNs verwendet. Im Anschluss konnten mit dem trainierten Netzwerk neue, unbekannte Orthofotos klassifiziert werden.

Die Daten des BEV eignen sich gut für das Training, da sie hochaufgelöst sind und ein großer Datensatz für ganz Österreich vorhanden ist. Mit den Daten kann ein CNN trainiert und auch angewandt werden. Da ein gut trainiertes Modell in der Lage ist, zu verallgemeinern, können die Ergebnisse des Trainings auch auf andere Gebiete ausgeweitet werden. Dadurch können auch andere Orthofotos erfolgreich klassifiziert werden. Problematisch wird dies jedoch, wenn das Modell neue Bildinhalte erkennen soll, die es während des Trainings noch nicht gesehen hat.

Bei einem der Orthofotos kam dieser Fall vor (dicht besiedeltes Gebiet; siehe Kapitel 5.3.3), wodurch ausgesprochen schlechte Ergebnisse entstanden sind. In diesem Fall fehlte es schlichtweg an Trainingsdaten in genau solchen Bereichen. Da die manuelle Klassifikation eines Orthofotos äußerst umständlich ist, war dies im Zuge dieser Masterarbeit nicht möglich. Die Klassifikation für ein dicht besiedeltes Gebiet ist noch zeitaufwendiger, da viele Details vorhanden sind, die ebenso klassifiziert werden müssen. Die Klassifikation eines CNNs ist letztlich immer nur so genau, wie es der Erstellter in den Trainingsdaten präsentiert. Entsprechend zeitaufwendig ist die Klassifikation eines Orthofotos.

Zusammenfassend kann somit festgehalten werden, dass ein CNN durchaus zur Klassifikation von Orthofotos geeignet ist. Der Zeitaufwand ist anfänglich hoch und es müssen viele Ressourcen investiert werden. Dabei reicht es nicht, nur zwei Orthofotos zu klassifizieren. Damit zum Beispiel ganz Österreich klassifiziert werden kann, müssen Orthofotos über das ganze Land verteilt klassifiziert und als Trainingsbilder für das CNN bereitgestellt werden. Dabei ist es vermutlich nicht notwendig, ein ganzes Orthofoto zu klassifizieren, sondern es könnten auch nur Ausschnitte bearbeitet werden. Jedoch gilt in diesem Bereich grundsätzlich: Je mehr Daten es gibt und je unterschiedlicher die Beispiele sind, desto genauer sind die Ergebnisse und umso besser kann das Modell am Ende generalisieren.

6.2. Parameter in einem Convolutional Neural Network

In einem CNN gibt es mehrere relevante Parameter. Die ausschlaggebenden für diese Masterarbeit waren die Modellwahl und die Lernrate. Bei den Modellen fiel die Wahl auf eine ResNet-Modellarchitektur, da diese in der Wissenschaft häufig angewendet und in vielen Studien Konsens über die Effizienz dieser Architektur herrscht. Von der ResNet-Architektur gibt es verschiedene Varianten, wobei für die Masterarbeit das ResNet-101 gewählt wurde. Dies wird für mittelgroße Datenmengen bevorzugt und konnte für diese Masterarbeit erfolgreich eingesetzt werden. Ob die Verwendung einer anderen Architektur sinnvoll ist, müsste in einem weiteren Schritt geprüft werden.

Einen Parameter, der von besonders großer Bedeutung ist, stellt die Lernrate dar. Je höher sie ist, desto schneller kann das Modell lernen. Dabei muss aber darauf geachtet werden, dass das Modell tatsächlich lernt, da es in vielen Fällen zu Problemen bezüglich des

Lernerfolges kommt. Bei zu hohen Raten kann es in passieren, dass kein Lernerfolg erkennbar ist. Bei einer langsamen Lernrate kommt es im Normallfall immer zu einem Erfolg, jedoch kann es bei einer zu niedrigen Rate sehr lange dauern, bis die optimalen Werte erreicht werden.

Mittels einer Hyperparametersuche wurde die optimale Lernrate gefunden. Da sich jedoch alle Werte im Umfeld des empfohlenen Standardwertes von 0,001 befanden, wurde dieser für alle Modellvarianten gewählt. Insgesamt wurden drei Modellvarianten trainiert, eines mit RGB-Daten, eines mit RGBI-Daten und das letzte mit RGBI- und nDSM-Daten. Interessanterweise schnitt das Modell mit den zusätzlichen nDSM-Daten während des Trainings am besten ab, es scheiterte aber anschließend bei den Klassifikationen der neuen Daten. Hier führte dieses Modell zu den ungenausten Ergebnissen.

Dieser Sachverhalt ist vermutlich darauf zurückzuführen, dass die beiden Datensätze (Orthofotos und Geländemodelle) durch unterschiedliche Methoden erzeugt wurden und somit aus verschiedenen Quellen (Kamera und Laserscanner) stammten. Zudem wurden die Daten zu verschiedenen Zeitpunkten aufgenommen, wodurch sich Unterschiede zwischen dem Bildinhalt des Orthofotos und den Daten der Geländemodelle ergeben können. Deshalb sollte für ein CNN ausschließlich auf eine einzige Datenquelle zurückgegriffen werden. Es könnte zum Beispiel auch ein Vegetationsindex wie der NDVI (Normalized Difference Vegetation Index) aus den Orthofotos berechnet und für das Training des CNNs bereitgestellt werden. Ob dies tatsächlich eine Verbesserung bringt, müsste jedoch getestet werden, was den Umfang dieser Masterarbeit jedoch bei weiten überstiegen hätte.

6.3. Trainingsdaten bei einem Convolutional Neural Network

Die Anzahl und der Inhalt der Trainingsdaten sind für den Erfolg eines CNNs essenziell. Grundsätzlich gilt, dass die Anzahl der Trainingsdaten unbegrenzt ist. Je mehr Daten für das Training verwendet werden, umso besser wird die endgültige Klassifikation. Dabei muss jedoch darauf geachtet werden, dass die zu trainierenden Klassen möglichst gleichmäßig im Datensatz vorkommen. Dazu stehen verschiedene Methoden, zum Beispiel die Data-Augmentation oder die Gewichtung der Klassen während des Trainings, zur Verfügung.

Mit einem CNN kann grundsätzlich jede gewünschte Klasse unterschieden werden. Ob die Erkennung funktioniert, hängt davon ab, wie gut sich die Klassen durch das neuronale Netzwerk unterscheiden lassen. Dabei wird es nie eine hundertprozentig korrekte Klassifizierung geben, da sich bereits bei der manuellen Klassifizierung der Trainingsdaten Fehler einschleichen. Im Fall dieser Masterarbeit hat die Differenzierung der Klassen ‚Unbefestigte Flächen' und ‚Befestigte Flächen' gut funktioniert und in den meisten Fällen die richtigen Ergebnisse produziert. Auch die Unterscheidung zwischen ‚Nadelbäume' und ‚Laubbäume' hat für große und weitläufige Wälder gepasst. Bei einzeln stehenden Bäumen hat sich das CNN im Normalfall für ‚Nadelbäume' entschieden. Das ist vermutlich darauf zurückzuführen, dass diese Klasse im

Trainingsdatensatz überrepräsentiert war und dies nicht durch die oben genannten Methoden ausgeglichen werden konnte.

In diesem Bereich müsste weiter untersucht werden, wie zuverlässig die Unterscheidung von Subklassen funktioniert. Im Zuge der vorliegenden Masterarbeit erfolgte eine Differenzierung, die jedoch weiter ausgebaut werden könnte. So kann zum Beispiel geprüft werden, ob das CNN auch Mischwälder erkennt. Auch im Hochgebirge sind die Klassen noch ausbaubar. Hier ist insbesondere die Unterscheidung von Felsen oder Gletschern denkbar. Die unterscheidbaren Klassen sind unbegrenzt, jedoch muss dabei bedacht werden, dass Zeit und Geduld für die Erstellung der Trainingsdaten benötigt wird. Auch kann es nach dem Training passieren, dass eine Klasse aufgrund der unzuverlässigen Klassifizierung verworfen werden muss.

6.4. Qualität der Klassifizierung

Die Messung der Qualität der Klassifizierung stellte im Zuge dieser Masterarbeit ein zeitaufwendiges Unterfangen dar. Einerseits wurden dafür Referenzgebiete mit den passenden Klassen klassifiziert und andererseits wurde die Landcover-Klassifikation des BEV herangezogen. Ein entscheidender Wert für die Qualität ist der IoU-Score, mit dem die Lagegenauigkeit der Segmentierungen gemessen werden kann. Die Berechnung dieses IoU-Scores wurde sowohl für die Daten des CNN als auch für diejenigen des BEV gemacht.

Dabei haben sich große Unterschiede zwischen den zur Überprüfung verwendeten Orthofotos ergeben. Die Ergebnisse dieser Klassifikationen waren im Wesentlichen von den Orthofotos abhängig, die für die Trainingsdaten verwendet wurden. Bei der Klassifikation von Orthofotos unbekannter Gebiete, die jedoch in der Nähe von denen der Trainingsdaten gelegen sind, hat das neuronale Netzwerk ausgesprochen gute Ergebnisse erzielt. Im Hinblick auf die Lagegenauigkeit der Klassifikation, also den IoU-Score, hat es jene des BEV sogar überstiegen. So wurden zum Beispiel die Gebäude- und Straßenpolygone zum größten Teil pixelgenau eingezeichnet und es zeigen wesentlich klarer definierte Grenzen als bei den Landcover-Klassifikationen des BEV.

Bei anderen Orthofotos, etwa jenem von dem Gebiet in Kufstein, waren die Ergebnisse hingegen mittelmäßig oder sogar unbrauchbar. Aufgrund des fehlenden Trainings in dicht besiedelten Regionen Österreichs war dieses Resultat jedoch zu erwarten. Für solche Regionen braucht das Modell Trainingsdaten aus dicht besiedelten Gebieten, damit es auch in solchen Bereichen funktioniert und diese später richtig klassifizieren kann. Der Fokus sollte dabei insbesondere auf großen Gebäuden bzw. Gebäuden mit einer Dachbegrünung und Photovoltaikanlagen liegen. Generell mangelte es dem in dieser Masterarbeit trainierten Modell an Trainingsdaten in Städten, wodurch sich das schlechte Ergebnis in Kufstein erklären lässt.

Auch bei den Flüssen müssen die Trainingsdaten erweitert und zudem über ganz Österreich verteilt werden. Dabei sollten möglichst viele und unterschiedlich aussehende Flüsse für das Training benutzt werden. Nur so können gute Ergebnisse für ganz Österreich erzielt werden. Dies gilt im Grunde für alle trainierten Klassen. Je mehr

Regionen Österreichs für den Trainingsprozess verwendet werden, umso besser kann das Modell am Ende mit unbekannten Daten aus dem ganzen Land umgehen. Da die Aufnahmen der Orthofotos über einen Zeitraum von vier Jahren gemacht werden, muss auch darauf geachtet werden, dass ein möglichst großer Zeitraum in den Trainingsdaten abgedeckt ist. Dabei sollte berücksichtigt werden, dass möglichst viele Variationen im Hinblick auf die Jahreszeiten vorhanden sind. Auf diese Weise sollten die Ergebnisse des CNNs an Genauigkeit gewinnen.

7. Conclusio und Ausblick in die Zukunft

Zu Beginn dieser Masterarbeit war ungewiss, ob und wie ein CNN für die Landcover-Klassifikation funktioniert und wie es implementiert wird. Dabei gab es Unterstützung vom BEV, das nicht nur die Daten, sondern auch die dazu passende Hardware zur Verfügung stellte.

Die Ergebnisse zeigen, dass das Projekt als gelungen anzusehen ist. Obwohl manche Testungen zu keinen perfekten Ergebnissen führten, war die Landcover-Klassifikation im Schnitt zuverlässig und erzeugte brauchbare Ergebnisse. Diese können in weiteren Schritten die aktuelle Klassifikationsmethode entweder ergänzen oder sogar vollständig ersetzen. Bis die zurzeit im BEV verwendete Software (eCognition von Trimble) ausgetauscht werden kann, sind jedoch noch viele Trainingsstunden notwendig. Zudem müssen wesentlich mehr Trainingsdaten generiert werden, was zeitintensiv ist und sich in vielen Fällen mühsam gestaltet.

Nichtsdestotrotz werden CNNs in Zukunft eine wesentliche Rolle bei der Fernerkundung spielen, nicht nur wegen ihrer Geschwindigkeit, sondern auch aufgrund der zunehmend genauen Ergebnisse. Insbesondere die Genauigkeit wird sich in Zukunft noch verbessern. Die von Wissenschaftlern veröffentlichten Modelle und Methoden werden immer umfangreicher und ressourcenhungriger, jedoch wächst damit in vielen Fällen auch die Genauigkeit der CNNs. Deshalb muss bei einer zukünftigen Implementation stets ein Blick auf den aktuellen Stand der Wissenschaft geworfen werden. Der Bereich der Computer-Vision, zu dem auch die CNNs zählen, ist ein aktueller und dynamischer Wissenschaftsbereich, in dem es nahezu täglich neue Veröffentlichungen gibt.

Auch im Bereich der Implementierung von CNNs in verschiedenen Softwareprodukten wird in Zukunft zunehmen. Zum Beispiel beginnt das Unternehmen ESRI, das auch ArcGIS Pro programmiert, solche Implementierungen vorzunehmen, und dies wird in Zukunft noch weiter intensiviert. Auch eCognition besitzt bereits die ersten Werkzeuge im Bereich der CNNs, die jedoch noch einfach gehalten sind und keine große Flexibilität bieten, wie sie zum Beispiel bei einer Implementierung in Python gegeben ist. Die direkte Implementierung in eCognition wurde zunächst auch im BEV geplant, jedoch bereits am Beginn des Schreibprozesses dieser Masterarbeit verworfen, da die genaue Funktionsweise unklar und keine Flexibilität bezüglich vieler Parameter gegeben war.

Die direkte Implementierung in Python war zwar ausgesprochen zeitaufwendig, sie bietet jedoch eine umfangreiche und flexible Umsetzung. Dadurch kann zusätzlich auf andere Bibliotheken in Python zurückgegriffen werden, die bei der Umsetzung helfen. Viele Teile des Programmcodes mussten in mühsamer Arbeit erörtert werden und es musste ihre Funktionsweise herausgefunden werden. Die lange Dauer eines solchen CNN-Trainings führt zudem dazu, dass die Austestung äußerst zeitintensiv ist.

Durch die Verwendung von stärkerer Hardware oder auch Berechnungen in der Cloud (z. B. Azure von Microsoft oder AWS von Amazon) werden solche umfangreichen Berechnungen deutlich verkürzt. Zudem wird *TensorFlow* mit großer Geschwindigkeit weiterentwickelt. Während dem Schreibprozess sind mehrere Versionen veröffentlicht

worden, wovon eine Version große Neuerungen gebracht hat. Daran lässt sich erkennen, dass *TensorFlow* eine große Bedeutung für Google spielt, wodurch dieses kontinuierlich verbessert wird.

Das Fazit dieser Masterarbeit lautet, dass ein CNN durchaus das Potenzial besitzt, den Bereich der Landcover-Klassifikation zu erneuern. Es handelt sich dabei um eine zuverlässige Methode, die teilweise bereits genauere Ergebnisse als andere Methoden liefert. Ob und wann sie sich in großem Maße durchsetzen wird, lässt sich noch nicht vorhersagen, jedoch verwenden Unternehmen wie Google und Microsoft täglich solche Methoden. Zurzeit ist der Einsatz von CNNs noch äußerst zeit- und arbeitsintensiv, und wer damit umgehen möchte, muss über profundes Wissen in diesem Bereich und gute Programmierkenntnisse verfügen. Mit der direkten Implementierung in bestimmte Programme wird dies in Zukunft jedoch wesentlich einfacher sein. In welchen Situationen und welchem Umfang solche CNNs künftig eingesetzt werden, ist noch ungewiss.

Literatur

ABADI, M., AGARWAL, A., BARHAM, P., BREVDO, E., CHEN, Z., CITRO, C., CORRADO, G. S., DAVIS, A., DEAN, J., DEVIN, M., GHEMAWAT, S., GOODFELLOW, I., HARP, A., IRVING, G., ISARD, M., JIA, Y., JOZEFOWICZ, R., KAISER, L., KUDLUR, M., ZHENG, X. (2016): TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. – In: arXiv [cs.DC]; auch online unter: http://arxiv.org/abs/1603.04467

ABADI, M., BARHAM, P., CHEN, J., CHEN, Z., DAVIS, A., DEAN, J., DEVIN, M., GHEMAWAT, S., IRVING, G., ISARD, M. (2016): Tensorflow: A system for large-scale machine learning. – In: 12th Symposium on Operating Systems Design and Implementation, 265–283; auch online unter: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi

ALOM, M. Z., TAHA, T. M., YAKOPCIC, C., WESTBERG, S., SIDIKE, P., NASRIN, M. S., VAN ESESN, B. C., AWWAL, A. A. S., & ASARI, V. K. (2018): The History Began from AlexNet: A Comprehensive Survey on Deep Learning Approaches. – In: arXiv [cs.CV]; auch online unter: http://arxiv.org/abs/1803.01164

ATIENZA, R. (2020): Advanced Deep Learning with TensorFlow 2 and Keras: Apply DL, GANs, VAEs, deep RL, unsupervised learning, object detection and segmentation, and more, 2nd Edition. – Birmingham.

AURELIO, Y. S., DE ALMEIDA, G. M., DE CASTRO, C. L., & BRAGA, A. P. (2019): Learning from Imbalanced Data Sets with Weighted Cross-Entropy Function. – In: Neural Processing Letters 50(2), 1937–1949; DOI: https://doi.org/10.1007/s11063-018-09977-1

BEV (Bundesamt für Eich- und Vermessungswesen) (2019a): Orthophotos Farbe; online 01.08.2019, http://www.bev.gv.at/portal/page?_pageid=713,1573980&_dad=portal&_schema=PORTAL (02.04.2020)

BEV (Bundesamt für Eich- und Vermessungswesen) (2019b): DOM - Digitales Oberflächenmodell; online 01.08.2019, https://www.bev.gv.at/portal/page?_pageid=713,2875569&_dad=portal&_schema=PORTAL (03.04.2020)

BEV (Bundesamt für Eich- und Vermessungswesen) (2012): DLM Digitales Landschaftsmodell; auch online unter: http://www.bev.gv.at/pls/portal/docs/PAGE/BEV_PORTAL_CONTENT_ALLGEMEIN/0550_SUPPORT/0500_DOWNLOADS/PRODUKTFOLDER/DLM_FOLDER_2012.PDF (06.04.2020)

BEV (Bundesamt für Eich- und Vermessungswesen) (o. J.): Blattschnitte; online o. J., http://www.bev.gv.at/pls/portal/docs/PAGE/BEV_PORTAL_CONTENT_ALLGEMEIN/0200_PRODUKTE/UNENTGELTLICHE_PRODUKTE_DES_BEV/Blattschnitte_V1.0.pdf (10.04.2020)

BEV (Bundesamt für Eich- und Vermessungswesen) (2017): Leistungsbericht 2017. – Wien.

BROWNLEE, J. (2017): Gentle Introduction to the Adam Optimization Algorithm for Deep Learning; online 14.11.2019, https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/ (24.04.2020)

BUSLAEV, A., IGLOVIKOV, V. I., KHVEDCHENYA, E., PARINOV, A., DRUZHININ, M., & KALININ, A. A. (2020): Albumentations: Fast and Flexible Image Augmentations. – In: An International Interdisciplinary Journal 11(2), 125; DOI: https://doi.org/10.3390/info11020125

CAPONETTO, G. (2019): Random Search vs. Grid Search for hyperparameter optimization; online 14.11.2019, https://towardsdatascience.com/random-search-vs-grid-search-for-hyperparameter-optimization-345e1422899d (07.04.2020)

CHELLAPILLA, K., PURI, S., & SIMARD, P. (2006): High performance convolutional neural networks for document processing; auch online unter: https://hal.inria.fr/inria-00112631/ (02.05.2020)

CIRESAN, D. C., MEIER, U., MASCI, J., GAMBARDELLA, L. M., & SCHMIDHUBER, J. (2011): Flexible, high performance convolutional neural networks for image classification. – In: Twenty-Second International Joint Conference on Artificial Intelligence; auch online unter: https://www.aaai.org/ocs/index.php/IJCAI/IJCAI11/paper/viewPaper/3098

CLEMATIDE, S. (2011): Evaluationsmasse; online 2011, https://files.ifi.uzh.ch/cl/siclemat/lehre/hs11/ecl1/script/html/scripthse19.html (18.04.2020)

CORINE Land Cover. (o. J.): Umweltbundesamt; online o. J., https://www.umweltbundesamt.at/rp_corine/ (27.01.2020)

DUMOULIN, V., & VISIN, F. (2016): A guide to convolution arithmetic for deep learning. – In: arXiv [stat.ML]; auch online unter: http://arxiv.org/abs/1603.07285

FERANEC, J., SOUKUP, T., HAZEU, G., & JAFFRAIN, G. (Hrsg.) (2016): European Landscape Dynamics. – In: CRC Press; DOI: https://doi.org/10.1201/9781315372860

Steadforce (2018): First steps with TensorFlow; online 16.01.2018, https://steadforce.com/de/first-steps-with-tensorflow-part-2/ (10.04.2020)

FUKUSHIMA, K. (1980): Neocognitron: a self organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. – In: Biological Cybernetics 36(4), 193–202; DOI: https://doi.org/10.1007/bf00344251

Geospatial Harris (2020): Image Types; online 2020, https://www.harrisgeospatial.com/docs/Image_Types.html (04.04.2020)

GÉRON, A. (2019): Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. – Bejing u.a.

GUO, R., LIU, J., LI, N., LIU, S., CHEN, F., CHENG, B., DUAN, J., LI, X., & MA, C. (2018):

Pixel-Wise Classification Method for High Resolution Remote Sensing Imagery Using Deep Neural Networks. – In: ISPRS International Journal of Geo-Information 7(3), 110; DOI: https://doi.org/10.3390/ijgi7030110

HANIN, B. (2018): Which Neural Net Architectures Give Rise to Exploding and Vanishing Gradients? – In: S. BENGIO, H. WALLACH, H. LAROCHELLE, K. GRAUMAN, N. CESA-BIANCHI, & R. GARNETT (Hrsg.): Advances in Neural Information Processing Systems 31. – 582-591.

HAYES, M. M., MILLER, S. N., & MURPHY, M. A. (2014): High-resolution landcover classification using Random Forest. – In: Remote sensing letters 5(2), 112–121; DOI: https://doi.org/10.1080/2150704X.2014.882526

HE, K., ZHANG, X., REN, S., & SUN, J. (2015): Deep Residual Learning for Image Recognition. – In: arXiv [cs.CV]; auch online unter: http://arxiv.org/abs/1512.03385

HINTON, G. E., SRIVASTAVA, N., KRIZHEVSKY, A., SUTSKEVER, I., & SALAKHUTDINOV, R. R. (2012): Improving neural networks by preventing co-adaptation of feature detectors. – In: arXiv [cs.NE]; auch online unter: http://arxiv.org/abs/1207.0580

HOFFER, E., HUBARA, I., & SOUDRY, D. (2017): Train longer, generalize better: closing the generalization gap in large batch training of neural networks. – In: arXiv [stat.ML]; auch online unter: http://arxiv.org/abs/1705.08741

HUBEL, D. H., & WIESEL, T. N. (1968): Receptive fields and functional architecture of monkey striate cortex. – In: The Journal of Physiology 195(1), 215–243; DOI: https://doi.org/10.1113/jphysiol.1968.sp008455

INDOLIA, S., GOSWAMI, A. K., MISHRA, S. P., & ASOPA, P. (2018): Conceptual Understanding of Convolutional Neural Network - A Deep Learning Approach. – In: Procedia computer science 132, 679–688; DOI: https://doi.org/10.1016/j.procs.2018.05.069

IOFFE, S., & SZEGEDY, C. (2015): Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. – In: arXiv [cs.LG]; auch online unter: http://arxiv.org/abs/1502.03167

JÉGOU, S., DROZDZAL, M., VAZQUEZ, D., ROMERO, A., & BENGIO, Y. (2017): The one hundred layers tiramisu: Fully convolutional densenets for semantic segmentation. – In: Proceedings of the IEEE conference on computer vision and pattern recognition workshops, 11–19.

JORDAN, J. (2018): An overview of semantic image segmentation; online 21.05.2018, https://www.jeremyjordan.me/semantic-segmentation/ (17.03.2020)

KAEHLER, A., & BRADSKI, G. (2016): Learning OpenCV 3: Computer Vision in C++ with the OpenCV Library. – Bejing u.a.

KAYALIBAY, B., JENSEN, G., & VAN DER SMAGT, P. (2017): CNN-based Segmentation of Medical Imaging Data. – In: arXiv [cs.CV]; auch online unter:

http://arxiv.org/abs/1701.03056

Keras (2019): Callbacks; online o.J., https://keras.io/callbacks/ (15.04.2020)

KINGMA, D. P., & BA, J. (2014): Adam: A Method for Stochastic Optimization. – In: arXiv [cs.LG]; auch online unter: http://arxiv.org/abs/1412.6980

KOEHRSEN, W. (2018): A Conceptual Explanation of Bayesian Hyperparameter Optimization for Machine Learning; online 24.06.2018, https://towardsdatascience.com/a-conceptual-explanation-of-bayesian-model-based-hyperparameter-optimization-for-machine-learning-b8172278050f (04.02.2020)

KRIZHEVSKY, A., SUTSKEVER, I., & HINTON, G. (2017): ImageNet classification with deep convolutional neural networks. – In: Communications of the ACM; DOI: https://doi.org/10.1145/3065386

LECUN, Y., BOTTOU, L., BENGIO, Y., & HAFFNER, P. (1998): Gradient-based learning applied to document recognition. – In: Proceedings of the IEEE 86(11), 2278–2324; DOI: https://doi.org/10.1109/5.726791

LIN, T.-Y., GOYAL, P., GIRSHICK, R., HE, K., & DOLLÁR, P. (2017): Focal Loss for Dense Object Detection. – In: arXiv [cs.CV]; auch online unter: http://arxiv.org/abs/1708.02002

LIU, W. S. (2019): Convolutional Neural Network Based Landcover Analysis of Satellite Images. – In: Journal of Physics: Conference Series 1345(2); DOI: https://doi.org/10.1088/1742-6596/1345/2/022003

ML Glossary documentation (o. J.): Loss Functions; online o. J., https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html (17.03.2020)

LU, D., & WENG, Q. (2007): A survey of image classification methods and techniques for improving classification performance. – In: International journal of remote sensing 28(5), 823–870; DOI: https://doi.org/10.1080/01431160600746456

LUO, W., LI, Y., URTASUN, R., & ZEMEL, R. (2016): Understanding the effective receptive field in deep convolutional neural networks. – In: Advances in neural information processing systems, 4898–4906.

MAHDIANPARI, M., SALEHI, B., REZAEE, M., MOHAMMADIMANESH, F., & ZHANG, Y. (2018): Very Deep Convolutional Neural Networks for Complex Land Cover Mapping Using Multispectral Remote Sensing Imagery. – In: Remote Sensing 10(7), 1119; DOI: https://doi.org/10.3390/rs10071119

MASTERS, D., & LUSCHI, C. (2018): Revisiting Small Batch Training for Deep Neural Networks. – In arXiv [cs.LG]; auch online unter: http://arxiv.org/abs/1804.07612

MITSCHKE, N., & HEIZMANN, M. (2019): Über die Detektierbarkeit von Objekten in Bildern mittels quantisierter neuronaler Netze. – In: tm - Technisches Messen 86(11), 651–660; DOI: https://doi.org/10.1515/teme-2019-0068

NG, W., MINASNY, B., MONTAZEROLGHAEM, M., PADARIAN, J., & MCBRATNEY, A. B. (2019): Convolutional neural network for simultaneous prediction of several soil properties using visible/near-infrared, mid-infrared, and their combined spectra. – In: Geoderma 352, 251–267; DOI: https://doi.org/10.1016/j.geoderma.2019.06.016

NIERADZIK, L. (2018): Losses for Image Segmentation; online 27.09.2018, https://lars76.github.io/neural-networks/object-detection/losses-for-segmentation/ (18.03.2020)

NWANKPA, C., IJOMAH, W., GACHAGAN, A., & MARSHALL, S. (2018): Activation Functions: Comparison of trends in Practice and Research for Deep Learning. – In: arXiv [cs.LG]; auch online unter: http://arxiv.org/abs/1811.03378

PECHYONKIN, M. (2018): Key Deep Learning Architectures: LeNet-5; online 01.10.2018, https://medium.com/@pechyonkin/key-deep-learning-architectures-lenet-5-6fc3c59e6f4 (09.03.2020)

PLANCHE, B., & ANDRES, E. (2019): Hands-On Computer Vision with TensorFlow 2: Leverage deep learning to create powerful image processing apps with TensorFlow 2.0 and Keras. – Bejing u.a.

PU, Y., APEL, D. B., SZMIGIEL, A., & CHEN, J. (2019): Image Recognition of Coal and Coal Gangue Using a Convolutional Neural Network and Transfer Learning. – In: Energies 12(9), 1735; DOI: https://doi.org/10.3390/en12091735

REITZ, K., & SCHLUSSER, T. (2016): The Hitchhiker's Guide to Python: Best Practices for Development. – Sebastopol.

RONNEBERGER, O., FISCHER, P., & BROX, T. (2015): U-Net: Convolutional Networks for Biomedical Image Segmentation. – In: arXiv [cs.CV]; auch online unter: http://arxiv.org/abs/1505.04597

ROUHI, R., JAFARI, M., KASAEI, S., & KESHAVARZIAN, P. (2015): Benign and malignant breast tumors classification based on region growing and CNN segmentation. – In: Expert systems with applications 42(3), 990–1002; DOI: https://doi.org/10.1016/j.eswa.2014.09.020

RUSSAKOVSKY, O., DENG, J., HUANG, Z., BERG, A. C., & FEI-FEI, L. (2013): Detecting avocados to zucchinis: what have we done, and where are we going? – In: Proceedings of the IEEE International Conference on Computer Vision, 2064–2071; auch online unter: http://ai.stanford.edu/~olga/papers/iccv13-ILSVRCanalysis.pdf

SAHOO, S. (2018): Residual blocks - Building blocks of ResNet; online 27.10.2018, https://towardsdatascience.com/residual-blocks-building-blocks-of-resnet-fd90ca15d6ec (10.04.2020)

SAMALA, R. K., CHAN, H.-P., HADJIISKI, L. M., CHA, K., & HELVIE, M. A. (2016): Deep-learning convolution neural network for computer-aided detection of microcalcifications in digital breast tomosynthesis. – In: Medical Imaging 2016: Computer-Aided Diagnosis 9785; DOI: https://doi.org/10.1117/12.2217092

SHARMA, S. (2017): Activation Functions in Neural Networks; online 06.09.2017, https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6 (20.04.2020)

SHENDRYK, Y., RIST, Y., TICEHURST, C., & THORBURN, P. (2019): Deep learning for multi-modal classification of cloud, shadow and land cover scenes in PlanetScope and Sentinel-2 imagery. – In: ISPRS journal of photogrammetry and remote sensing: official publication of the International Society for Photogrammetry and Remote Sensing 157, 124–136; DOI: https://doi.org/10.1016/j.isprsjprs.2019.08.018

SHIJIE, J., PING, W., PEIYI, J., & SIPING, H. (2017): Research on data augmentation for image classification based on convolution neural networks. – In: 2017 Chinese Automation Congress (CAC), 4165–4170; DOI: https://doi.org/10.1109/CAC.2017.8243510

SINGH, R. (2019): Integrating Deep Learning with GIS. Medium; online 23.02.2019, https://medium.com/geoai/integrating-deep-learning-with-gis-70e7c5aa9dfe (20.04.2020)

Spektrum (2001): Orthophoto; online 2001, https://www.spektrum.de/lexikon/geographie/orthophoto/5740 (02.04.2020)

SZEGEDY, C., WEI LIU, YANGQING JIA, SERMANET, P., REED, S., ANGUELOV, D., ERHAN, D., VANHOUCKE, V., & RABINOVICH, A. (2015): Going deeper with convolutions. – In: 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 1–9; DOI: https://doi.org/10.1109/CVPR.2015.7298594

TensorFlow (2020a): tf.data: Build TensorFlow input pipelines; online 01.04.2020, https://www.tensorflow.org/guide/data (09.04.2020)

TensorFlow (2020b): TFRecord and tf.Example; online 01.04.2020, https://www.tensorflow.org/tutorials/load_data/tfrecord (09.04.2020)

TensorFlow (2020c): tf.io.TFRecordOptions; online 01.04.2020, https://www.tensorflow.org/api_docs/python/tf/io/TFRecordOptions (10.04.2020)

TensorFlow (o. J.): TensorBoard; online o. J., https://www.tensorflow.org/tensorboard (15.04.2020)

THORMÄHLEN, T. (2018): Technische Informatik – Speicher; auch online unter: https://www.mathematik.uni-marburg.de/~thormae/lectures/ti1/ti_8_1_ger_web.html

TRUONG, P. (2019): Loss functions: Why, what, where or when?; online 29.03.2019, https://medium.com/@phuctrt/loss-functions-why-what-where-or-when-189815343d3f (07.04.2020)

VIGUERAS-GUILLÉN, J. P., SARI, B., GOES, S. F., LEMIJ, H. G., VAN ROOIJ, J., VERMEER, K. A., & VAN VLIET, L. J. (2019): Fully convolutional architecture vs sliding-window CNN for corneal endothelium cell segmentation. – In: BMC Biomedical Engineering 1(1), 4; DOI: https://doi.org/10.1186/s42490-019-0003-2

YAKUBOVSKIY, P. (o. J.): Segmentation Models; online o. J., https://segmentation-models.readthedocs.io/en/latest/api.html (01.03.2020)

YANG, C., ROTTENSTEINER, F., & HEIPKE, C. (2018): CLASSIFICATION OF LAND COVER AND LAND USE BASED ON CONVOLUTIONAL NEURAL NETWORKS. – In: ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences IV-3, 251–258; DOI: https://doi.org/10.5194/isprs-annals-IV-3-251-2018

Anhang I: Python-Skript zum Zählen der Pixel

Als Input dienen die Pfade der Ordner, in denen die Bildmasken gespeichert sind. Dies kann sowohl vor als auch nach der Image Augmentation gemacht werden. Wenn die Image Augmentation bereits durchgeführt wurde, muss dieser Ordner ebenso eingelesen werden.

```
1.  import glob
2.  import sys
3.  import numpy as np
4.  import rasterio
5.  import pandas as pd
6.
7.  #Erstellen einer Liste mit den Pfaden aller Bildmaskenn aus den verschiedenen Ordne
    rn. Wenn bereits durch eine Image Augmentation Bilder erzeugt wurden, dann kann der
     Ordner ebenfall eingelesen werden
8.  dachstein_label_liste = glob.glob(r"E:\Daten\UNet\samples_dachstein\labels\*.tif")

9.  mariazell_label_liste = glob.glob(r"E:\Daten\UNet\samples_mariazell\labels\*.tif")

10. wien_label_liste = glob.glob(r"E:\Daten\UNet\samples_wien\labels\*.tif")
11. aug_label_liste =glob.glob(r"E:\Daten\UNet\samples_augmentation\labels\*.tif")
12. liste_gesamt = dachstein_label_liste+mariazell_label_liste+wien_label_liste+aug_lab
    el_liste
13.
14. #Erstellen eines leeren Arrays, der später alle Bildmasken in einem einzelnen Array
     speichert
15. array = np.zeros([len(liste_gesamt), 256, 256], dtype=np.uint8)
16.
17. #Einlesen der Bilder und Speicherung im Array
18. iterator = 0
19. for i in liste_gesamt:
20.     label_src = rasterio.open(i)
21.     label = label_src.read(1)
22.     array[iterator; : ; :] = label
23.     sys.stdout.write('\r>> Opening image %d' % (iterator))
24.     sys.stdout.flush()
25.     iterator += 1
26.
27. #Anzahl der einzigartigen Pixel aus dem Array auslesen
28. unique, count = np.unique(array, return_counts=True)
29.
30. #Speicherung der Zahlen in einem Pandas Dataframe (für eine bessere Darstellung) un
    d anschließende Darstellung als Tortendiagramm
31. dictionary_pixels = [['Gebaeude', count[0]],
32.                      ['Versiegelte Flaechen', count[1]],
33.                      ['Unversiegelte Flaechen', count[2]],
34.                      ['Gewaesser', count[3]],
35.                      ['Nadelbaeume', count[4]],
36.                      ['Laubbaeume', count[5]],
37.                      ['Sonstiges', count[6]]]
38. df = pd.DataFrame(dictionary_pixels, columns = ['Klasse', 'Anzahl'])
39. print(df)
```

Anhang II: Python-Skript für die Image Augmentation

Als Input dienen erneut die Pfade zu den Bildern und Masken die mithilfe von ArcGIS Pro exportiert wurden. Der Output sind die veränderten zusätzlichen Bilder im gewünschten Ordner.

```
import glob
import sys
import rasterio
import numpy as np
import albumentations

#Einlesen aller Pfade in denen sich die Bilder und die Bildmasken befinden
image_path_mariazell = glob.glob(r"E:\Daten\UNet\samples_mariazell\images\*.tif")
label_path_mariazell = glob.glob(r"E:\Daten\UNet\samples_mariazell\labels\*.tif")
image_path_wien = glob.glob(r"E:\Daten\UNet\samples_wien\images\*.tif")
label_path_wien = glob.glob(r"E:\Daten\UNet\samples_wien\labels\*.tif")
image_path_dachstein = glob.glob(r"E:\Daten\UNet\samples_dachstein\images\*.tif")
label_path_dachstein = glob.glob(r"E:\Daten\UNet\samples_dachstein\labels\*.tif")
image_paths_list = image_path_mariazell + image_path_wien + image_path_dachstein
label_paths_list = label_path_mariazell + label_path_wien + label_path_dachstein

#Erstellen einer Funktion die nach der Image Augmentation die Bilder und die dazuge
   hörigen Bildmasken speichert
def create_image(image_path, image, label_path, label):
    with rasterio.open(
        image_path,
        'w',
        driver='GTiff',
        height = 256,
        width = 256,
        count = 5,
        dtype = 'float32'
    ) as dst:
        dst.write(image, [1,2,3,4,5])

    with rasterio.open(
        label_path,
        'w',
        driver='GTiff',
        height = 256,
        width = 256,
        count = 1,
        dtype = 'uint8'
    ) as dst:
        dst.write(label, 1)
    return

#Erstellen einer Funktion mit den passenden Parametern. Der Parameter "p" gibt die
   Wahrscheinlichtkeit an, mit welcher die Veraenderung durchgeführt werden soll.

def augmenters():
    aug = albumentations.Compose([
        albumentations.VerticalFlip(p=0.7),
        albumentations.OneOf([ #Eine der nachfolgenden Veraenderungen wird gewählt

            albumentations.RandomRotate90(p=0.5),
            albumentations.Rotate(limit=(180),p=0.5)
        ], p=1.0),
        albumentations.HorizontalFlip(p=0.7),
        albumentations.Transpose(p=0.6),
        albumentations.ShiftScaleRotate(p=0.5),
```

```
53.         albumentations.OneOf([ #Eine der nachfolgenden Veraenderungen wird gewählt

54.             albumentations.RandomBrightnessContrast(p=0.7),
55.             albumentations.RandomBrightness(p=0.7),
56.             albumentations.RandomGamma(p=0.7)
57.         ], p=0.5),
58.         albumentations.OneOf([ #Eine der nachfolgenden Veraenderungen wird gewählt

59.             albumentations.Blur(blur_limit=3, p=0.5),
60.             albumentations.GaussianBlur(blur_limit=3, p=0.5)
61.         ], p=0.2),
62.         albumentations.OneOf([ #Eine der nachfolgenden Veraenderungen wird gewählt

63.             albumentations.ElasticTransform(p=0.2, alpha=150, sigma=150*0.05, alpha
    _affine=150 * 0.03),
64.             albumentations.RandomSizedCrop(p=0.2, min_max_height=(200, 200), height
    =256, width=256)
65.         ], p=0.3)
66.     ], additional_targets={
67.         'image': 'image',
68.         'mask': 'mask',
69.         'ndsm': 'mask' #Das nDSM wird hier als Maske behandelt, damit gewisse Verae
    nderungen nicht auf diese angewandt werden.
70.     })
71.     return(aug)
72.
73. #Startet die Image Augmentation
74. count_all = 0
75. for image_path, label_path in zip(image_paths_list, label_paths_list):
76.
77.     sys.stdout.write('\r>> Opening image %s' % (image_path))
78.     sys.stdout.flush()
79.
80.     #Einlesen der Maske
81.     label_src = rasterio.open(label_path)
82.     label = label_src.read(1))
83.     label = label.astype(np.uint8)
84.
85.     #Einlesen des Bildes (nur RGBI) und des nDSM
86.     image_src = rasterio.open(image_path)
87.     image = image_src.read([1,2,3,4])
88.     image = np.moveaxis(image, 0, 2)
89.     ndsm = image_src.read([5])
90.     ndsm = np.moveaxis(ndsm, 0, 2)
91.
92.     #Auslesen der Anzahl der einzigartige Pixel in der Maske und anschließende Bere
    chnung des Anteils pro Klasse an der Gesamtanzahl der Pixel pro Maske
93.     unique, count = np.unique(label, return_counts=True)
94.     anteile_array = np.zeros([7,])
95.     for categ, iteration in zip(unique, range(0, len(unique))):
96.         anteile_array[categ] = count[iteration] / np.sum(count) * 100
97.
98.     #Durchfuehrung der Image Augmentation. Für jede Klasse wurde ein bestimmter Ant
    eil festgelegt, der im Bild vorkommen muss.
99.     if anteile_array[0] >= 10.0:
100.        for iteration in range(0, 32): #32 zusaetzliche fuer Bilder mit Gebaeuden
101.            augmenter = augmenters()
102.            data= {'image' : image, 'mask' : label, 'ndsm' : ndsm}
103.            augmented = augmenter(**data)
104.            aug_img_path = "E:\\Daten\\UNet\\samples_augmentation\\images\\" + str(
    count_all) + ".tif"
105.            aug_label_path = "E:\\Daten\\UNet\\samples_augmentation\\labels\\" + st
    r(count_all) + ".tif"
106.            image_new = np.moveaxis(augmented['image'], 2, 0)
107.            ndsm_new = np.moveaxis(augmented['ndsm'], 2, 0)
```

```
            image_comb = np.append(image_new, ndsm_new, axis=0) #Zusammenfuegen des Bildes und des nDSM
            create_image(aug_img_path, image_comb, aug_label_path, augmented['mask'])
            count_all += 1
        continue
    elif anteile_array[2] >= 10.0:
        for iteration in range(0, 32): #32 zusaetzliche fuer Bilder mit Unversiegelten Flaechen
            augmenter = augmenters()
            data= {'image' : image, 'mask' : label, 'ndsm' : ndsm}
            augmented = augmenter(**data)
            aug_img_path = "E:\\Daten\\UNet\\samples_augmentation\\images\\" + str(count_all) + ".tif"
            aug_label_path = "E:\\Daten\\UNet\\samples_augmentation\\labels\\" + str(count_all) + ".tif"
            image_new = np.moveaxis(augmented['image'], 2, 0)
            ndsm_new = np.moveaxis(augmented['ndsm'], 2, 0)
            image_comb = np.append(image_new, ndsm_new, axis=0) #Zusammenfuegen des Bildes und des nDSM
            create_image(aug_img_path, image_comb, aug_label_path, augmented['mask'])
            count_all += 1
        continue
    elif anteile_array[1] >= 20.0:
        for iteration in range(0, 16): #16 zusaetzliche fuer Bilder mit Versiegelten Flaechen
            augmenter = augmenters()
            data= {'image' : image, 'mask' : label, 'ndsm' : ndsm}
            augmented = augmenter(**data)
            aug_img_path = "E:\\Daten\\UNet\\samples_augmentation\\images\\" + str(count_all) + ".tif"
            aug_label_path = "E:\\Daten\\UNet\\samples_augmentation\\labels\\" + str(count_all) + ".tif"
            image_new = np.moveaxis(augmented['image'], 2, 0)
            ndsm_new = np.moveaxis(augmented['ndsm'], 2, 0)
            image_comb = np.append(image_new, ndsm_new, axis=0) #Zusammenfuegen des Bildes und des nDSM
            create_image(aug_img_path, image_comb, aug_label_path, augmented['mask'])
            count_all += 1
        continue
    elif anteile_array[3] >= 15.0:
        for iteration in range(0, 4): #4 zusaetzliche fuer Bilder mit Gewaessern
            augmenter = augmenters()
            data= {'image' : image, 'mask' : label, 'ndsm' : ndsm}
            augmented = augmenter(**data)
            aug_img_path = "E:\\Daten\\UNet\\samples_augmentation\\images\\" + str(count_all) + ".tif"
            aug_label_path = "E:\\Daten\\UNet\\samples_augmentation\\labels\\" + str(count_all) + ".tif"
            image_new = np.moveaxis(augmented['image'], 2, 0)
            ndsm_new = np.moveaxis(augmented['ndsm'], 2, 0)
            image_comb = np.append(image_new, ndsm_new, axis=0) #Zusammenfuegen des Bildes und des nDSM
            create_image(aug_img_path, image_comb, aug_label_path, augmented['mask'])
            count_all += 1
        continue
    elif anteile_array[5] >= 30.0:
        for iteration in range(0, 4): #4 zusaetzliche fuer Bilder mit Laubbaeumen
            augmenter = augmenters()
            data= {'image' : image, 'mask' : label, 'ndsm' : ndsm}
            augmented = augmenter(**data)
            aug_img_path = "E:\\Daten\\UNet\\samples_augmentation\\images\\" + str(count_all) + ".tif"
```

```
157.             aug_label_path = "E:\\Daten\\UNet\\samples_augmentation\\labels\\" + st
     r(count_all) + ".tif"
158.             image_new = np.moveaxis(augmented['image'], 2, 0)
159.             ndsm_new = np.moveaxis(augmented['ndsm'], 2, 0)
160.             image_comb = np.append(image_new, ndsm_new, axis=0) #Zusammenfuegen des
      Bildes und des nDSM
161.             create_image(aug_img_path, image_comb, aug_label_path, augmented['mask'
     ])
162.             count_all += 1
163.         continue
164.     else:
165.         continue #Die übrigen Klassen werden nicht behandelt
```

Anhang III: Python-Skript zur Erstellung der TFRecords

Als Input dienen alle zur Verfügung stehenden Daten. Diese werden in ein *TFRecord* umgewandelt und dann als solche gespeichert.

```
import glob
import os
import rasterio
import numpy as np
import random
import sys
import tensorflow as tf
import math
import cv2

#Festlegen der Kompressionsmethode bei den TFRecords
options = tf.io.TFRecordOptions(compression_type="GZIP")

#Ausgabeordner für die TFRecords
tfrecord_folder = "E:/Daten/UNet/tfrecord_imbalance/"
image_format = "tif"

#Anzahl in wie viele kleine Teile der gesamte Datensatz geteilt werden soll
num_shards = 500

#Einlesen der Pfade zu den Bildern
list_images_dachstein = glob.glob("E:/Daten/UNet/samples_dachstein/images/*.tif")
list_images_wien = glob.glob("E:/Daten/UNet/samples_wien/images/*.tif")
list_images_mariazell = glob.glob("E:/Daten/UNet/samples_mariazell/images/*.tif")
list_images_augment = glob.glob("E:/Daten/UNet/samples_augmentation/images/*.tif")

list_images_gesamt = list_images_dachstein + list_images_wien + list_images_mariaze
ll + list_images_augment

list_image = []
for i in list_images_gesamt:
    list_image.append(i)

#Zufälliges durchmischen der Liste mit den Bildern und Trennung des Trainings-
 und Validierungsdatensatzes
random.shuffle(list_image)
num_images = len(list_image)
train_samples = list_image[:math.ceil(num_images*0.8)]
trainval_samples = list_image[math.ceil(num_images*0.8):]

if os.path.exists("E:/Daten/UNet/index/train.txt"):
    os.remove("E:/Daten/UNet/index/train.txt")
    os.remove("E:/Daten/UNet/index/trainval.txt")

#Erstellen von Text-Files mit den Pfaden zu den passenden Bildern
txt_train = open("E:/Daten/UNet/index/train.txt", "a")
for example in train_samples:
    txt_train.write(example + "\n")
txt_trainval = open("E:/Daten/UNet/index/trainval.txt", "a")
for example in trainval_samples:
    txt_trainval.write(example + "\n")
txt_train.close()
txt_trainval.close()

#Funktion zum Starten der Konvertierung
def dataset_split():
    dataset_splits = tf.io.gfile.glob(os.path.join('E:/Daten/UNet/index/', '*.txt')
)
```

```
55.     for dataset_split in dataset_splits:
56.         convert_dataset(dataset_split)
57.
58. #Funktionen für Beispiele festlegen, welche Daten in die TFRecords fließen sollen u
    nd wie diese serialisiert werden sollen
59. def _bytes_feature(value):
60.     if isinstance(value, type(tf.constant(0))):
61.         value = value.numpy()
62.     return tf.train.Feature(bytes_list=tf.train.BytesList(value=[value]))
63.
64. def _int64_feature(value):
65.     return tf.train.Feature(int64_list=tf.train.Int64List(value=[value]))
66.
67. def serialize_example(image_data, filename, format_type, height, width, channels, m
    ask_data):
68.     return tf.train.Example(features=tf.train.Features(feature = {
69.         'image/encoded': _bytes_feature(image_data),
70.         'image/filename' : _bytes_feature(filename),
71.         'image/format' : _bytes_feature(format_type),
72.         'height' : _int64_feature(height),
73.         'width' : _int64_feature(width),
74.         'depth' : _int64_feature(channels),
75.         'segmentation/encoded' : _bytes_feature(mask_data),
76.         'segmentation/format': _bytes_feature(format_type),
77.     }))
78.
79. #Funktion für die Konvertierung
80. def convert_dataset(dataset_split):
81.     dataset = os.path.basename(dataset_split)[:-4]
82.     sys.stdout.write('Processing ' + dataset)
83.     filenames = [x.strip('\n') for x in open(dataset_split, 'r')]
84.     num_images = len(filenames)
85.     num_per_shard = int(math.ceil(num_images / num_shards)) #Berechnen wie viele Bi
    lder pro Shard enthalten sind
86.
87.     for shard in range(num_shards):
88.         tfrecord_filename = os.path.join(tfrecord_folder, "%s-%05d-of-
    %05d.tfrecord" % (dataset, shard + 1, num_shards))
89.         with tf.io.TFRecordWriter(tfrecord_filename, options) as tfrecord_writer: #
    Startet den Schreibprozess
90.             start_index = shard * num_per_shard
91.             end_index = min((shard + 1) * num_per_shard, num_images)
92.             for i in range(start_index, end_index):
93.                 sys.stdout.write('\r>> Converting image %d/%d shard %d' % (i + 1, l
    en(filenames), shard))
94.                 sys.stdout.flush()
95.                 image_filename = filenames[i]
96.                 src = rasterio.open(image_filename)
97.                 image =  src.read([1,2,3,4]) #Einlesen des RGBI-Fotos
98.                 image = np.moveaxis(image, 0, 2)
99.                 ndsm = src.read([5]) #Einlesen des nDSM
100.                ndsm = cv2.normalize(ndsm, 0, 1, cv2.NORM_MINMAX) #Normalisierung d
    es nDSMs zwischen die Werte 0 und 1
101.                ndsm = np.moveaxis(ndsm, 0, 2)
102.                image_data = np.concatenate((image, ndsm), axis = 2) #Zusammenfügen
     des RGBI-Fotos und des nDSM
103.                image_data = tf.io.serialize_tensor(image_data) #Serialisierung des
     Bildes
104.                height, width, channels = src.shape[0], src.shape[1], 5
105.                mask_filename = filenames[i].replace("images", "labels")
106.                src = rasterio.open(mask_filename)
107.                mask = src.read([1]) #Einlesen der passenden Maske
108.                mask = np.moveaxis(mask, 0, 2)
109.                mask = np.squeeze(mask, axis = 2)
110.                mask = mask.astype(np.uint8)
111.                mask_data = tf.io.serialize_tensor(mask) #Serialisieren der Maske
```

```
                seg_height, seg_width = src.shape[0], src.shape[1]
                format_type = tf.io.serialize_tensor("tif")
                filename_byte = tf.io.serialize_tensor(filenames[i])
                if height != seg_height or width != seg_width:
                    raise RuntimeError('Bildgroesse und Maskengroesse sind ungleich
    !')
                example = serialize_example(image_data, filename_byte, format_type,
     height, width, channels, mask_data) #Schreiben der serialisierten Daten
                tfrecord_writer.write(example.SerializeToString())
        sys.stdout.write('\n')
        sys.stdout.flush()

dataset_split() #AUfrufen der Funktion, welche den gesamten Prozess startet
```

Anhang IV: Python-Skript zur Hyperparametersuche

Als Input dienen dieselben *TFRecords* wie sie nachher beim Training verwendet werden. Der Output ist eine Übersicht über die besten gefundenen Hyperparameter.

```
1.  import segmentation_models as sm
2.  import tensorflow as tf
3.  import numpy as np
4.  from kerastuner.tuners import bayesian
5.  import kerastuner
6.
7.  #Einlesen der TFRecords
8.  train_tfrecord = tf.data.Dataset.list_files(str('E:/Daten/UNet/tfrecord/train-
    *.tfrecord'), shuffle=True)
9.  trainval_tfrecord = tf.data.Dataset.list_files(str('E:/Daten/UNet/tfrecord/trainval
    -*.tfrecord'), shuffle=True)
10. tf_train_dataset = tf.data.TFRecordDataset(train_tfrecord, num_parallel_reads=8, co
    mpression_type='GZIP')
11. tf_val_dataset = tf.data.TFRecordDataset(trainval_tfrecord, num_parallel_reads=8, c
    ompression_type='GZIP')
12.
13. def read_tfrecord(serialized_example):
14.     feature_description = {
15.         'image/encoded' : tf.io.FixedLenFeature((), tf.string),
16.         'image/filename' : tf.io.FixedLenFeature((), tf.string),
17.         'image/format' : tf.io.FixedLenFeature((), tf.string),
18.         'height' : tf.io.FixedLenFeature((), tf.int64),
19.         'width' : tf.io.FixedLenFeature((), tf.int64),
20.         'depth' : tf.io.FixedLenFeature((), tf.int64),
21.         'image/segmentation/class/encoded' : tf.io.FixedLenFeature((), tf.string),

22.         'image/segmentation/class/format' : tf.io.FixedLenFeature((), tf.string),
23.     }
24.
25.     example = tf.io.parse_single_example(serialized_example, feature_description)
26.
27.     image = tf.io.parse_tensor(example['image/encoded'], out_type = tf.float32)
28.     image_shape = [example['height'], example['width'], 5]
29.     image = tf.reshape(image, image_shape)
30.     image = image[:, :, :4] #Kanaele angeben
31.     mask = tf.io.parse_tensor(example['image/segmentation/class/encoded'], out_type
     = tf.uint8)
32.     mask_shape = [example['height'], example['width']]
33.     mask = tf.reshape(mask, mask_shape)
34.     mask = tf.one_hot(mask, depth=7)
35.     return(image, mask)
36.
37. parsed_train_dataset = tf_train_dataset.map(read_tfrecord, num_parallel_calls = tf.
    data.experimental.AUTOTUNE)
38. parsed_val_dataset = tf_val_dataset.map(read_tfrecord, num_parallel_calls = tf.data
    .experimental.AUTOTUNE)
39. train_ds = parsed_train_dataset.shuffle(buffer_size=4000, reshuffle_each_iteration=
    True).batch(16, drop_remainder=False).prefetch(buffer_size=tf.data.experimental.AUT
    OTUNE)
40. val_ds = parsed_val_dataset.shuffle(buffer_size=4000, reshuffle_each_iteration=True
    ).batch(16, drop_remainder=False).prefetch(buffer_size=tf.data.experimental.AUTOTUN
    E)
41.
42. #Erstellen eines Hyperparameter Spaces
43. def build_model(hp):
44.     sm.set_framework('tf.keras')
45.     model = sm.Unet('resnet152', classes=7, activation='softmax', encoder_weights=N
    one, input_shape=(None, None, 4), decoder_use_batchnorm=True)
```

```
46.     weights_class = np.array([1.,  1., 2.,  1.,  0.5, 1.0,  0.5], dtype=np.float32)
47.     loss = sm.losses.CategoricalCELoss(class_weights=weights_class)
48.     metrics = [sm.metrics.IOUScore(threshold = 0.5), sm.metrics.FScore(threshold = 0.5)]
49.     optimizer = tf.keras.optimizers.Adam(hp.Float('learning_rate', 1e-5, 1e-1, sampling='log'))
50.     model.compile(optimizer=optimizer, loss=loss, metrics=metrics)
51.     return(model)
52.
53. #Festlegen welcher Wert minimiert werden soll
54. tuner = bayesian.BayesianOptimization(build_model, objective=kerastuner.Objective("val_loss", direction="min"), max_trials=20, num_initial_points= 3)
55.
56. #Zusammenfassung ausgeben
57. tuner.search_space_summary()
58.
59. #Suche beginnen
60. tuner.search(train_ds, epochs=5, validation_data=val_ds, verbose=2)
```

Anhang V: Python-Skript zum Training eines Convolutional Neural Networks

Als Input dienen die zuvor erstellten *TFRecords*. Der Output dieses Vorgangs ist ein .h5-File, welches die trainierten Gewichtungen und das Modell beinhaltet.

```
1.  import segmentation_models as sm
2.  import tensorflow as tf
3.  import numpy as np
4.
5.  #Erstellen eines Modells mit ResNet, welches sieben Klassen klassifizieren kann. Di
    e Anzahl der Kanaele, welche das Bild besitzt, muss ebenso angegeben werden.
6.  sm.set_framework('tf.keras')
7.  model = sm.Unet('resnet101', classes=7, activation='softmax', encoder_weights=None,
     input_shape=(None, None, 4), decoder_use_batchnorm=True)
8.
9.  #Erstellen einer tf.data.Dataset Instanz mit welcher die TFREcords eingelesen werde
    n koennen
10. train_tfrecord = tf.data.Dataset.list_files(str('E:/Daten/UNet/tfrecord_imbalance/t
    rain-*.tfrecord'), shuffle=True)
11. trainval_tfrecord = tf.data.Dataset.list_files(str('E:/Daten/UNet/tfrecord_imbalanc
    e/trainval-*.tfrecord'), shuffle=True)
12. tf_train_dataset = tf.data.TFRecordDataset(train_tfrecord, num_parallel_reads=8, co
    mpression_type='GZIP')
13. tf_val_dataset = tf.data.TFRecordDataset(trainval_tfrecord, num_parallel_reads=8, c
    ompression_type='GZIP')
14.
15. #Erstellung einer Funktion damit TensorFlow weiß, wie die TFREcords formatiert sind
     und wie diese in den urspruenglichen Zustand umgeformt werden
16. def read_tfrecord(serialized_example):
17.     feature_description = {
18.         'image/encoded' : tf.io.FixedLenFeature((), tf.string),
19.         'image/filename' : tf.io.FixedLenFeature((), tf.string),
20.         'image/format' : tf.io.FixedLenFeature((), tf.string),
21.         'height' : tf.io.FixedLenFeature((), tf.int64),
22.         'width' : tf.io.FixedLenFeature((), tf.int64),
23.         'depth' : tf.io.FixedLenFeature((), tf.int64),
24.         'segmentation/encoded' : tf.io.FixedLenFeature((), tf.string),
25.         'segmentation/format' : tf.io.FixedLenFeature((), tf.string),
26.     }
27.
28.     example = tf.io.parse_single_example(serialized_example, feature_description)
29.
30.     image = tf.io.parse_tensor(example['image/encoded'], out_type = tf.float32)
31.     image_shape = [example['height'], example['width'], 5]
32.     image = tf.reshape(image, image_shape)
33.     image = image[:, :, :4] #Bei RGB muss die Zahl durch eine 3 ersetzt werden und
    wenn alle verfügbaren Kanele benutzt werden sollen, muss nur ein Doppelpunkt eingef
    uegt werden
34.     mask = tf.io.parse_tensor(example['segmentation/encoded'], out_type = tf.uint8)

35.     mask_shape = [example['height'], example['width']]
36.     mask = tf.reshape(mask, mask_shape)
37.     mask = tf.one_hot(mask, depth=7)
38.     return(image, mask)
39.
40. #Erstellung eines Datasets, welches einen Puffer einliest und den durchmischt. Dara
    us werden Batches aus 16 Bildern erstellt. Durch das Prefetchen werden die Bilder v
    on der CPU eingelesen und stehen so schneller zur Verfuegung.
41. parsed_train_dataset = tf_train_dataset.map(read_tfrecord, num_parallel_calls = tf.
    data.experimental.AUTOTUNE)
42. parsed_val_dataset = tf_val_dataset.map(read_tfrecord, num_parallel_calls = tf.data
    .experimental.AUTOTUNE)
```

```
43. train_ds = parsed_train_dataset.shuffle(buffer_size=5000, reshuffle_each_iteration=
    True).batch(16, drop_remainder=False).prefetch(buffer_size=tf.data.experimental.AUT
    OTUNE)
44. val_ds = parsed_val_dataset.shuffle(buffer_size=5000, reshuffle_each_iteration=True
    ).batch(16, drop_remainder=False).prefetch(buffer_size=tf.data.experimental.AUTOTUN
    E)
45.
46. #Erstellung einer Gewichtung pro Klasse
47. weights_class = np.array([1.,  1., 2.,  1.,  0.5, 1.0,  0.5], dtype=np.float32)
48.
49. #Definieren von Callbacks
50. model_callback = tf.keras.callbacks.ModelCheckpoint(filepath= 'E:/Daten/UNet/callba
    ck/' + 'model_rgbi_softmax_rgbi.h5', monitor='val_loss', save_best_only=True)
51. tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir='E:\\Daten\\UNet\\ten
    sorboard\\model_rgbi_softmax_rgbi\\')
52. early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss', min_delta=0,
    patience=10)
53.
54. #Definieren des Losses, der Genauigkeitmasse und des Optimizers
55. loss = sm.losses.CategoricalCELoss(class_weights=weights_class)
56. metrics = [sm.metrics.IOUScore(threshold = 0.5), sm.metrics.FScore(threshold = 0.5)
    ]
57. optimizer = tf.keras.optimizers.Adam(0.001)
58.
59. #Modell kompilieren
60. model.compile(optimizer=optimizer, loss=loss, metrics=metrics)
61.
62. #Training starten
63. history = model.fit(train_ds, epochs=300, validation_data=val_ds, callbacks=[model_
    callback, tensorboard_callback, early_stopping])
```

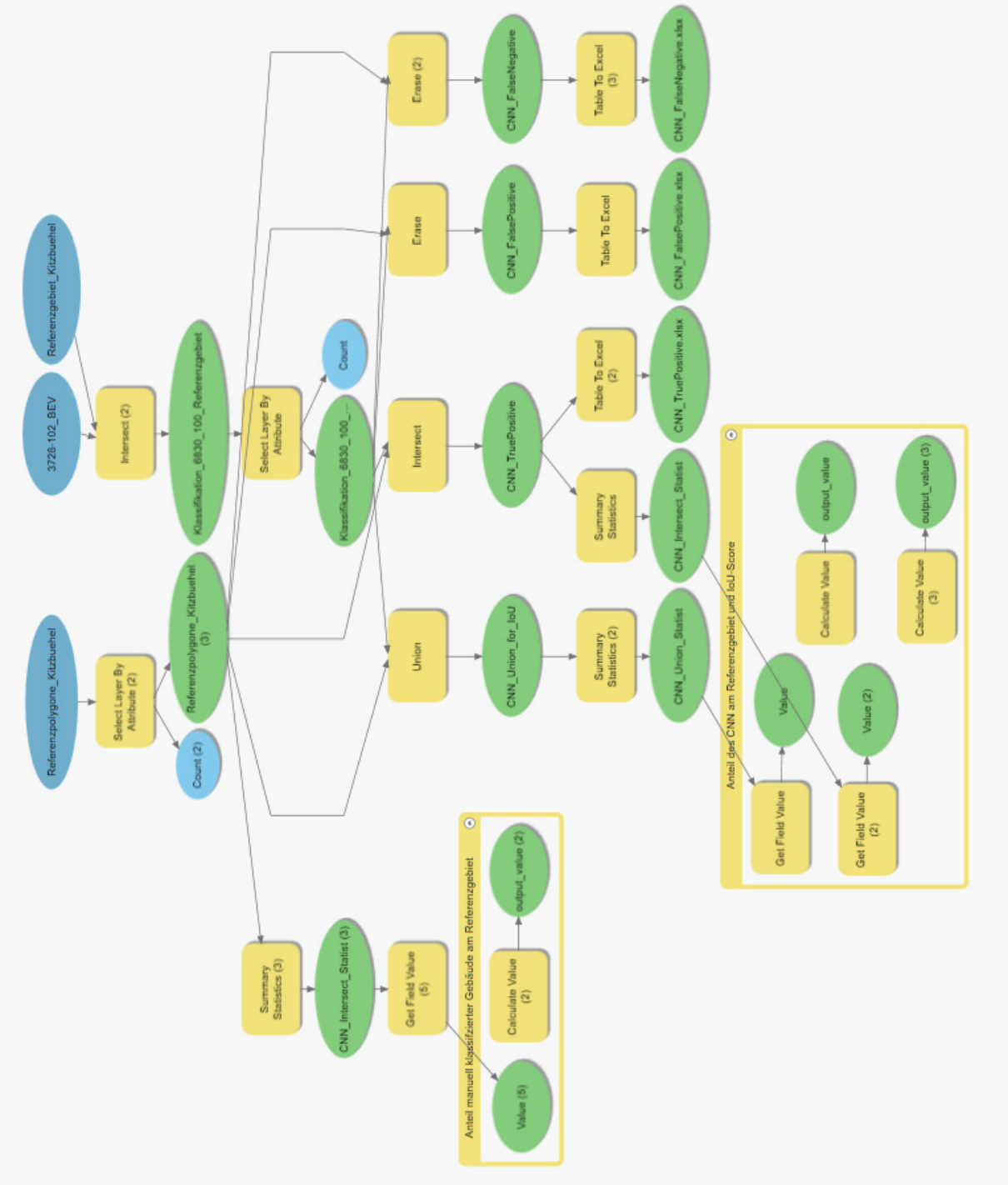
Anhang VI: Python-Skript zur Klassifikation und Segmentierung von Orthofotos

Als Input dient das zu klassifizierende Orthofoto im ÖLK-Blattschnitt. Zusätzlich können das Geländemodell und das Höhenmodell eingelesen werden, wenn dieses benötigt wird. Der Output dieses Vorgangs ist ein Bild, welches die Klassifikation und Segmentierung repräsentiert.

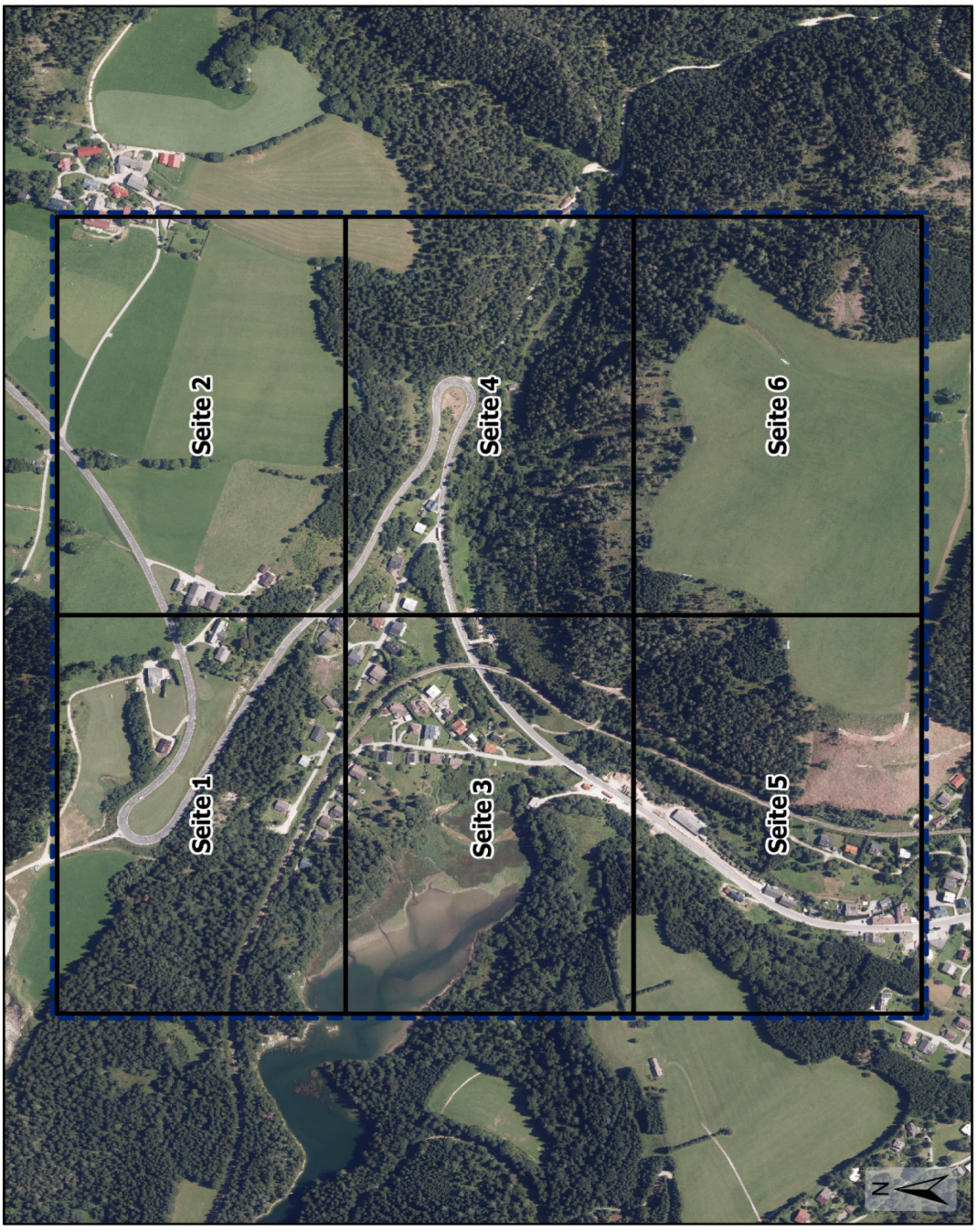
```
import segmentation_models as sm
import tensorflow as tf
import numpy as np
import cv2
import os
import sys
import rasterio
import skimage

#Erstellen eines, wie beim Training, identen Modells
sm.set_framework('tf.keras')
model = sm.Unet('resnet101', classes=7, activation='softmax', encoder_weights=None, input_shape=(None, None, 4), decoder_use_batchnorm=True)

#Laden der Gewichtungen des Modells
model.load_weights(r"E:\Daten\UNet\callback\model_rgbi_softmax_rgbi.h5")

#Öffnen des Orthofotos
image_src = rasterio.open(r"D:\Klassifikation\Mariazell\DOP\630-100_DOP_G.tif")
image = image_src.read([1,2,3,4])
image = skimage.img_as_float32(image)
image = np.moveaxis(image, 0, 2)
image = cv2.copyMakeBorder(image, 0, 100, 0, 100, cv2.BORDER_REFLECT101)

#Wenn ein nDSM verwendet werden soll, müssen die nachfolgenden Zeilen verwendet werden
dsm_src = rasterio.open(r"D:\Klassifikation\Mariazell\DSM\6830-100_DSM_G.tif")
dtm_src = rasterio.open(r"D:\Klassifikation\Mariazell\DTM\6830-100_DTM_G.tif")
dsm = dsm_src.read([1])
dsm = np.moveaxis(dsm, 0, 2)
dsm = skimage.transform.resize(dsm, (25500, 25500, 1))
dtm = dtm_src.read([1])
dtm = np.moveaxis(dtm, 0, 2)
dtm = skimage.transform.resize(dtm, (25500, 25500, 1))
ndsm = dsm - dtm
ndsm = cv2.copyMakeBorder(ndsm, 0, 100, 0, 100, cv2.BORDER_REFLECT101)
ndsm = cv2.normalize(ndsm, 0, 1, cv2.NORM_MINMAX)
ndsm = np.expand_dims(ndsm, 2)
dtm, dsm = 0, 0
image = np.concatenate((image, ndsm), axis=2)
ndsm = 0

#Erstellen eines leeren Arrays (Klassifikations-Array), wo die berechneten Klassifikation gespeichert werden sollen
classification_pred = np.zeros((1, 25600, 25600, 7), dtype=np.float16)

#Erstellen des Sliding-Window der über das Orthofoto fährt und an jeder Position eine Klassifikation berechnet und in den soeben erstellten Array speichert
for row in range(0, 25601-2560, 256):
    for col in range(0, 25601-2560, 256):
        sys.stdout.write('\r>> Converting row %d and col %d' % (row, col))
        sys.stdout.flush()
```

```
56.         prediction = model.predict(np.expand_dims(image[row:row+2560, col:col+2560,
    :], 0))
57.         prediction = prediction.astype(np.float16)
58.         classification_pred[:, row:row+2560, col:col+2560, :] = np.where(prediction
    > classification_pred[:, row:row+2560, col:col+2560, :], prediction, classificatio
    n_pred[:, row:row+2560, col:col+2560, :])
59.         sys.stdout.flush()
60.     sys.stdout.flush()
61.
62. #Beschneiden des Arrays auf die originale Größe des Bildes und anschließende Speich
    erung
63. classification_pred = classification_pred[0, :25500, :25500, :]
64. classification_pred = classification_pred.argmax(axis=2)
65. classification = classification.astype(np.uint8)
66. classification = np.expand_dims(classification, 0)
67.
68. new_dataset = rasterio.open(
69.     r"D:\Klassifikation\Mariazell\Klassifikation\6830-
    100_Klassifikation_rgbi.tif",
70.     'w',
71.     driver = 'GTiff',
72.     height=25500,
73.     width=25500,
74.     count=1,
75.     dtype='uint8',
76. )
77. new_dataset.write(classification, [1])
78. new_dataset.close()
```

Anhang VII: Abbildung der zur Evaluierung verwendeten ArcGIS-Modells



Anhang VIII: Klassifikationsergebnisse für das Referenzgebiet in Mariazell



Überblick über den Referenzbereich in Mariazell (ÖLK: 6830-100) 2020

Die Karte liefert einen Überblick über den klassifizierten Bereich in Mariazell und gibt die Seitennummern für die Ausschnitte in den Klassifikationsergebnissen an. In den nachfolgenden drei Anhängen sind die Klassifikationen durch das CNN mit allen drei Modellvarianten visualisiert.

0 80 160 240 320 m

Maßstab: 1:8.000

Erstellung: Felix Pekárek

Datengrundlage Orthofoto: Bundesamt für Eich- und Vermessungswesen 2020

Klassifikation von Mariazell (ÖLK: 6830-100) mithilfe eines CNN (RGB-Modell) 2020

Landcover-Klassen

- Gebäude
- Versiegelte Flächen
- Unversiegelte Flächen
- Gewässer
- Nadelbäume
- Laubbäume
- Sonstiges

Referenzgebiet Mariazell

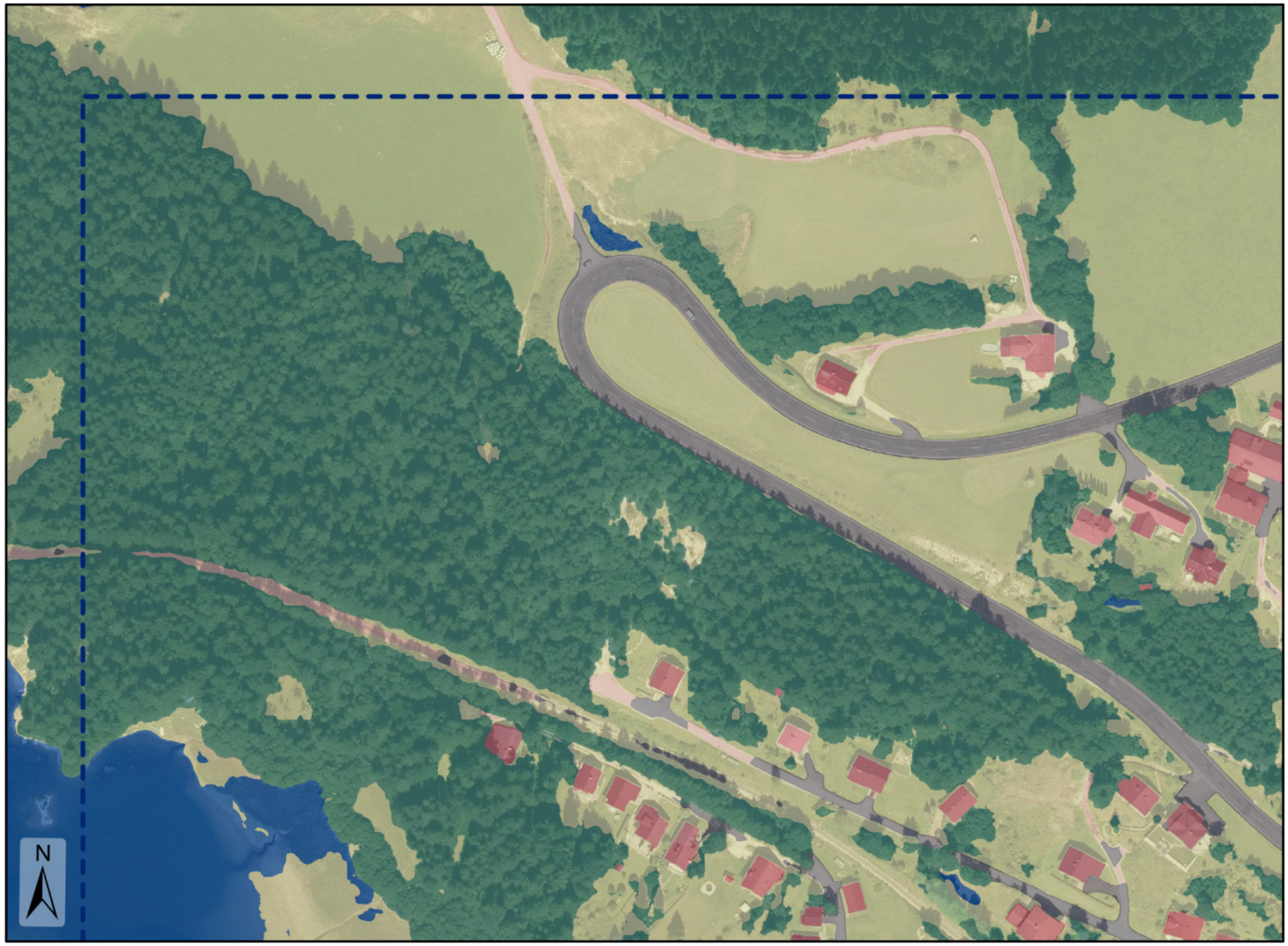
0 30 60 90 120 m

Maßstab: 1:3.000

Seite 1 von 6

Erstellung: Felix Pekárek

Datengrundlage Orthofoto: Bundesamt für Eich- und Vermessungswesen 2020



Klassifikation von Mariazell (ÖLK: 6830-100) mithilfe eines CNN (RGB-Modell) 2020

Landcover-Klassen

- Gebäude
- Versiegelte Flächen
- Unversiegelte Flächen
- Gewässer
- Nadelbäume
- Laubbäume
- Sonstiges

Referenzgebiet Mariazell

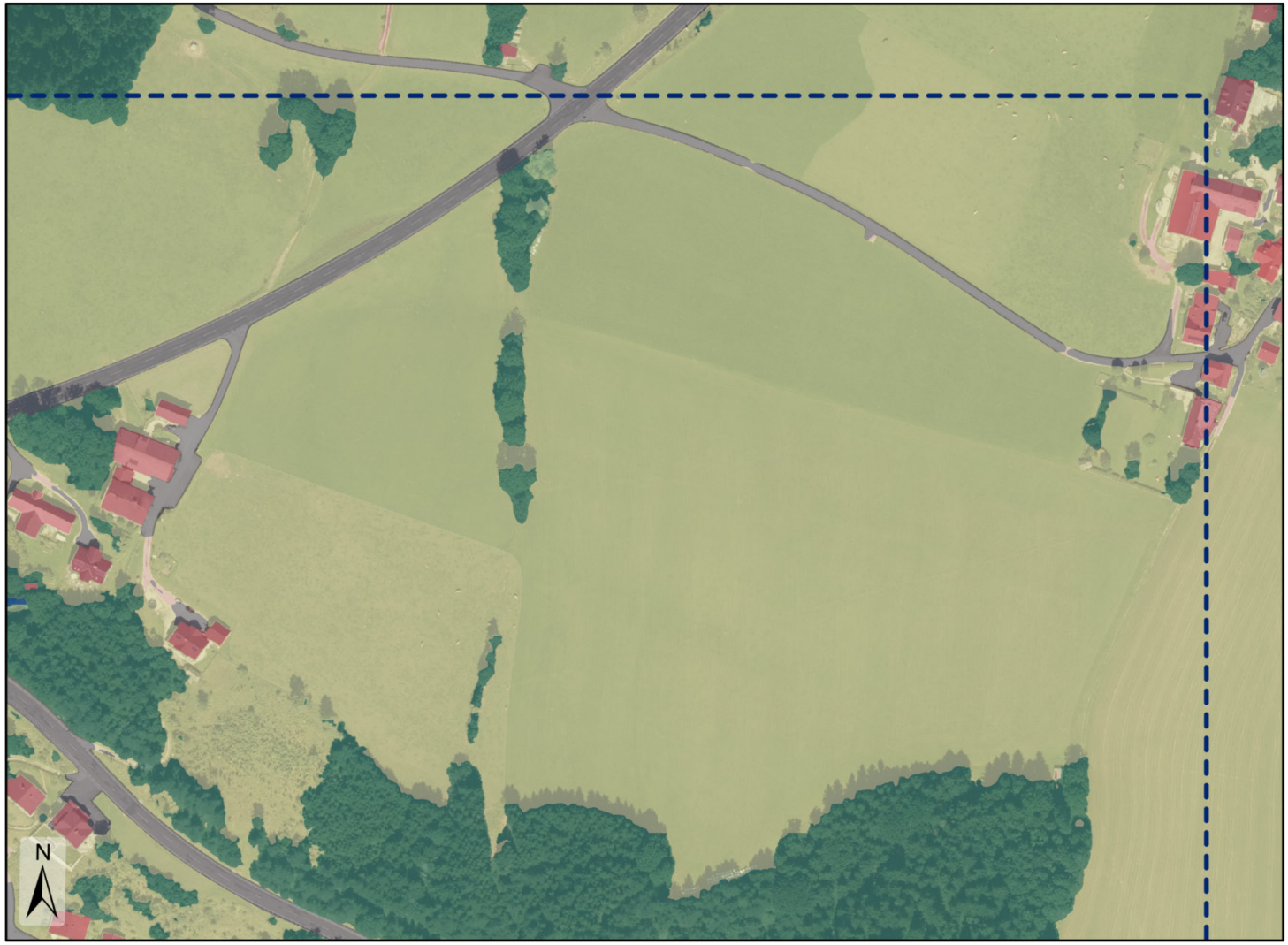
0 30 60 90 120 m

Maßstab: 1:3.000

Seite 2 von 6

Erstellung: Felix Pekárek

Datengrundlage Orthofoto: Bundesamt für Eich- und Vermessungswesen 2020



Klassifikation von Mariazell (ÖLK: 6830-100) mithilfe eines CNN (RGB-Modell) 2020

Landcover-Klassen

- Gebäude
- Versiegelte Flächen
- Unversiegelte Flächen
- Gewässer
- Nadelbäume
- Laubbäume
- Sonstiges

Referenzgebiet Mariazell

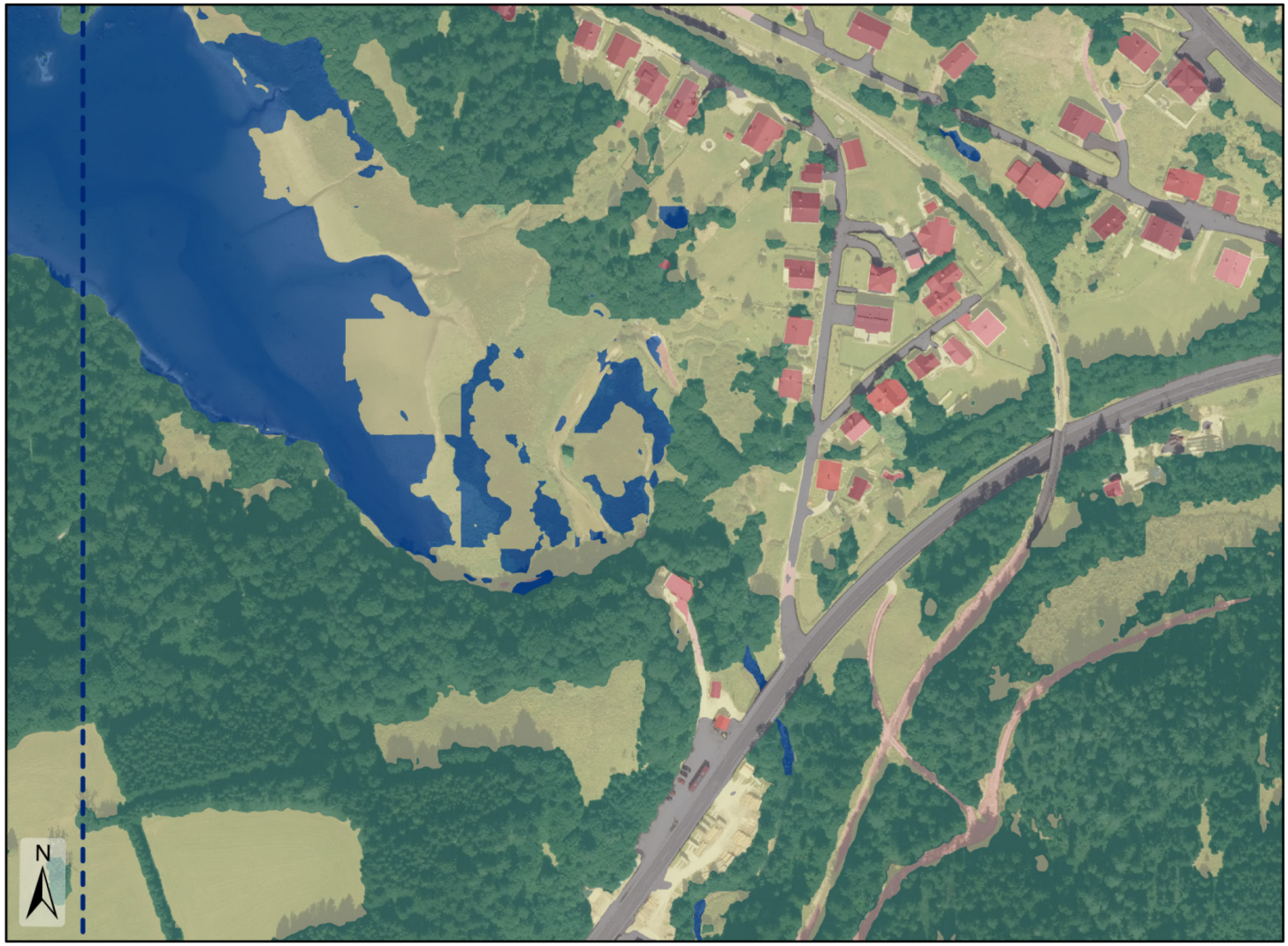
0 30 60 90 120 m

Maßstab: 1:3.000

Seite 3 von 6

Erstellung: Felix Pekárek

Datengrundlage Orthofoto: Bundesamt für Eich- und Vermessungswesen 2020



Klassifikation von Mariazell (ÖLK: 6830-100) mithilfe eines CNN (RGB-Modell) 2020

Landcover-Klassen

- Gebäude
- Versiegelte Flächen
- Unversiegelte Flächen
- Gewässer
- Nadelbäume
- Laubbäume
- Sonstiges

Referenzgebiet Mariazell

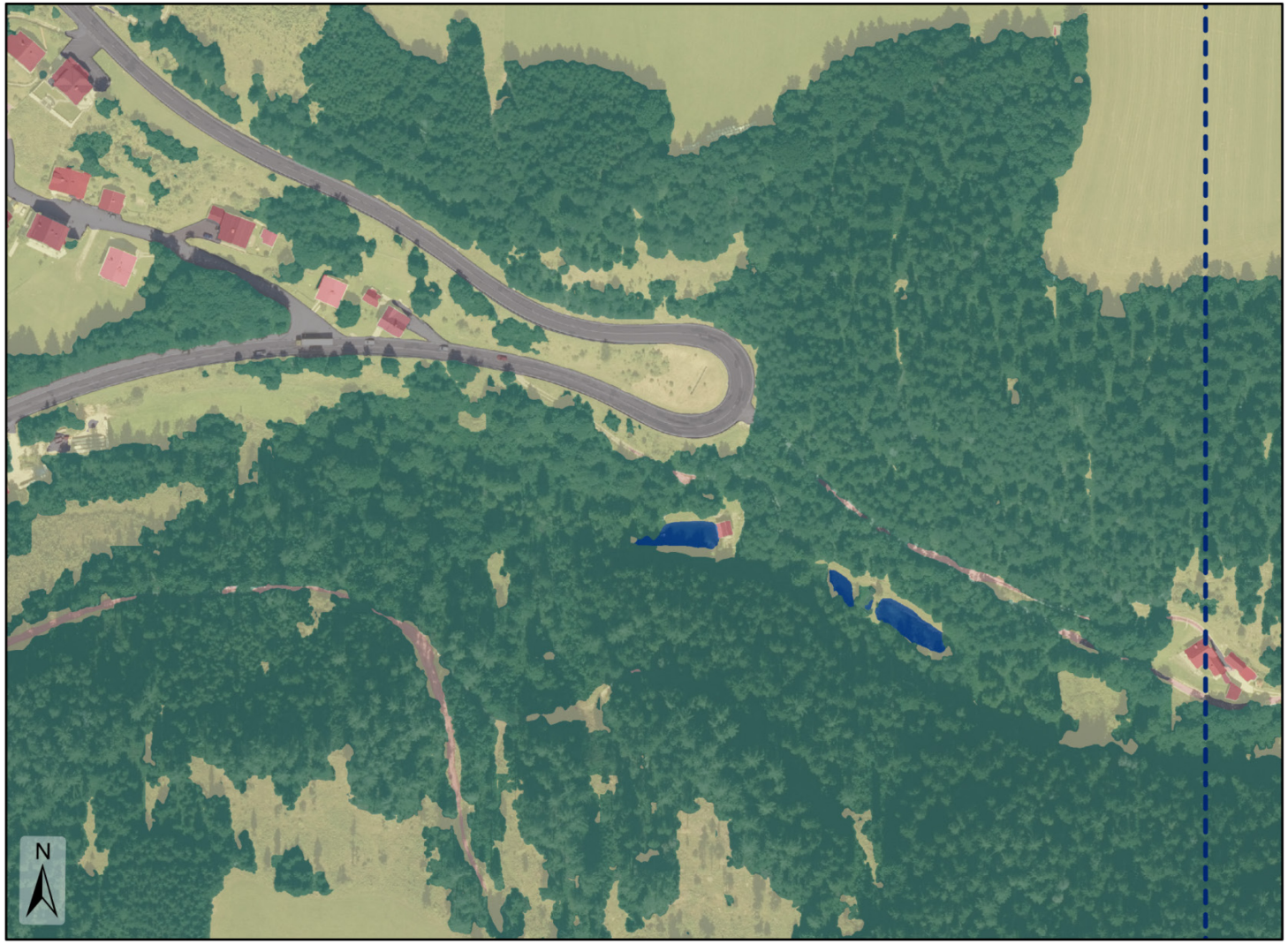
0 30 60 90 120 m

Maßstab: 1:3.000

Seite 4 von 6

Erstellung: Felix Pekárek

Datengrundlage Orthofoto: Bundesamt für Eich- und Vermessungswesen 2020



Klassifikation von Mariazell (ÖLK: 6830-100) mithilfe eines CNN (RGB-Modell) 2020

Landcover-Klassen

- Gebäude
- Versiegelte Flächen
- Unversiegelte Flächen
- Gewässer
- Nadelbäume
- Laubbäume
- Sonstiges

Referenzgebiet Mariazell

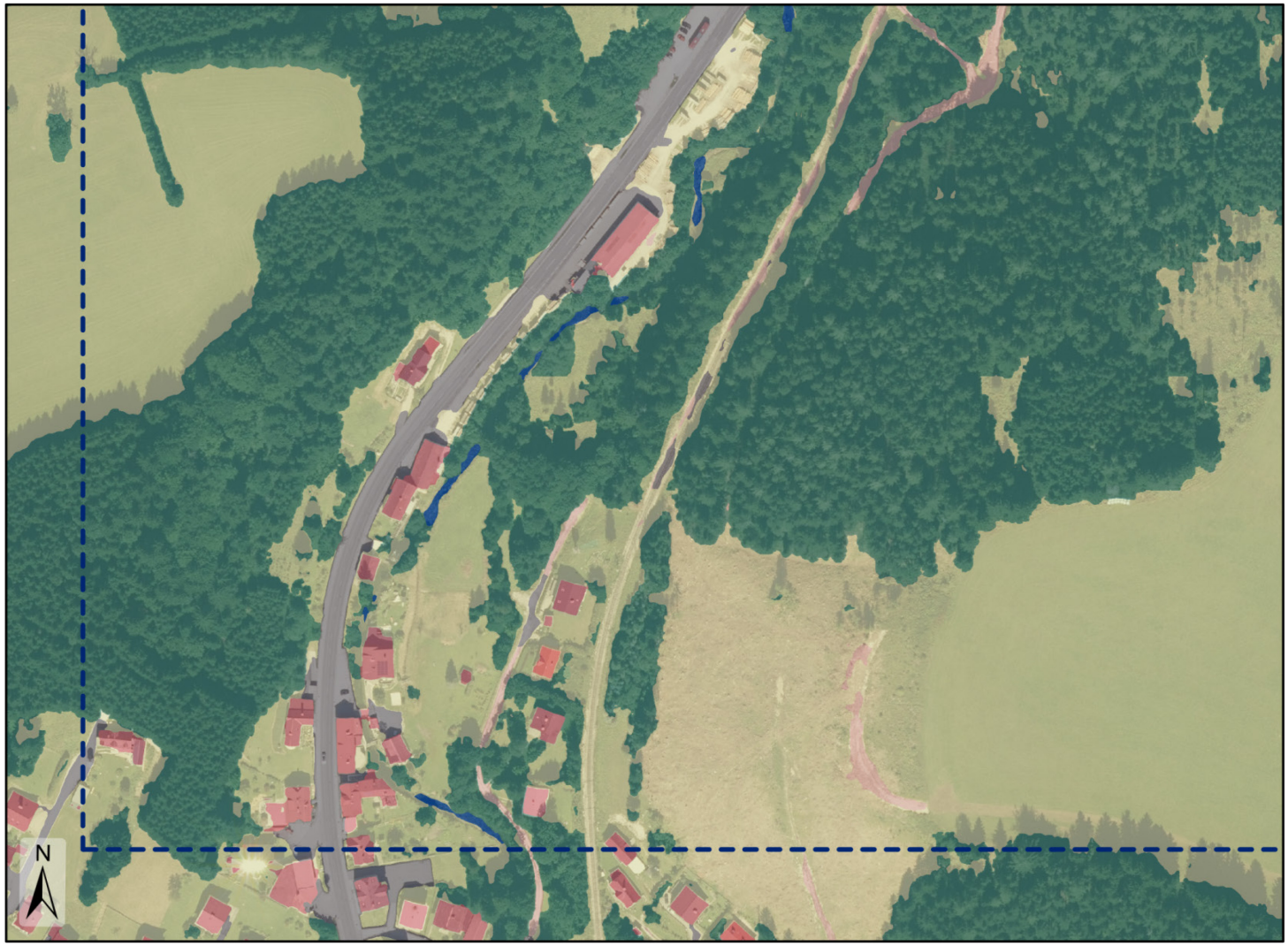
0 30 60 90 120 m

Maßstab: 1:3.000

Seite 5 von 6

Erstellung: Felix Pekárek

Datengrundlage Orthofoto: Bundesamt für Eich- und Vermessungswesen 2020



Klassifikation von Mariazell (ÖLK: 6830-100) mithilfe eines CNN (RGB-Modell) 2020

Landcover-Klassen

- Gebäude
- Versiegelte Flächen
- Unversiegelte Flächen
- Gewässer
- Nadelbäume
- Laubbäume
- Sonstiges

Referenzgebiet Mariazell

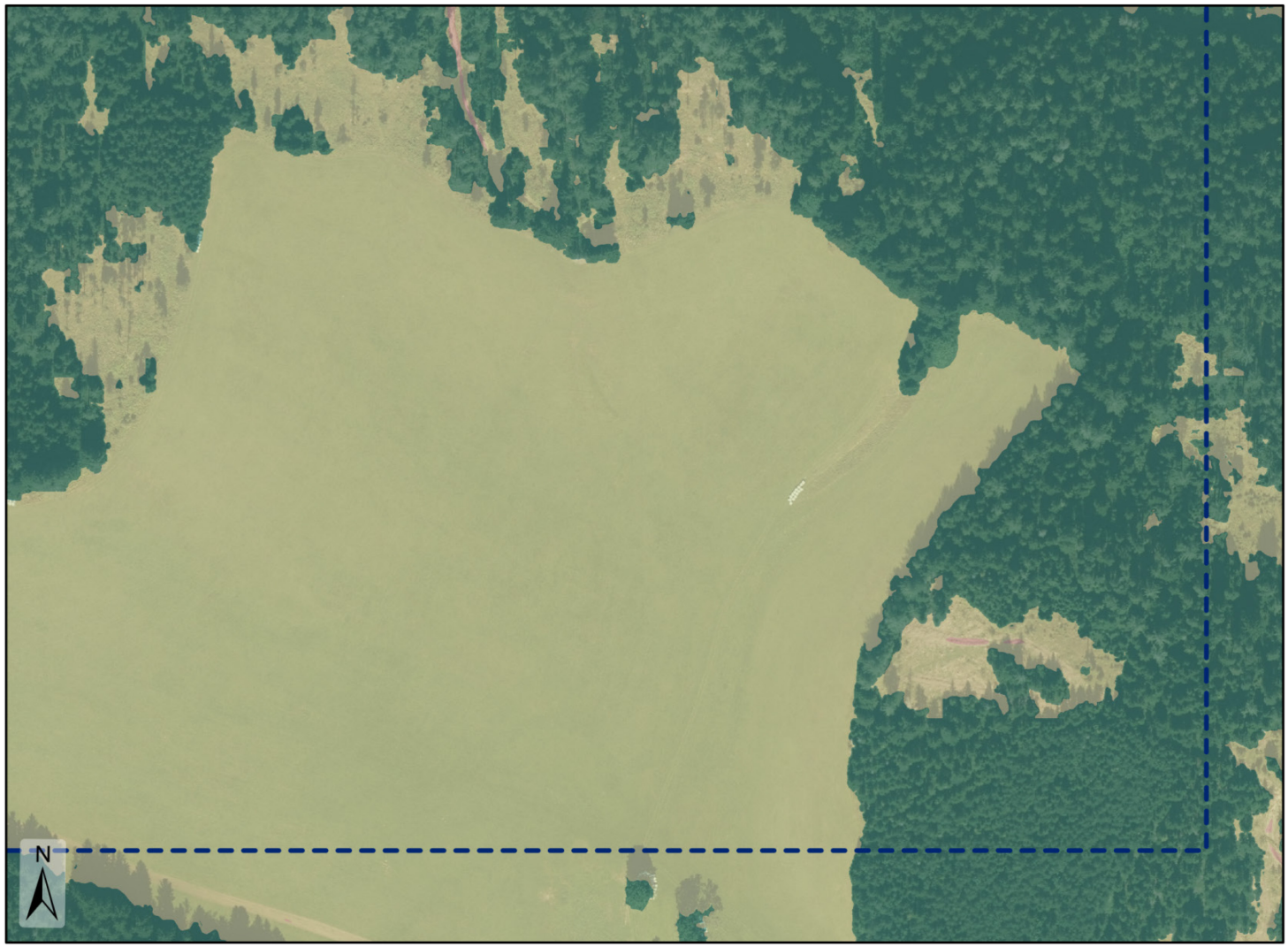
0 30 60 90 120 m

Maßstab: 1:3.000

Seite 6 von 6

Erstellung: Felix Pekárek

Datengrundlage Orthofoto: Bundesamt für Eich- und Vermessungswesen 2020



Klassifikation von Mariazell (ÖLK: 6830-100) mithilfe eines CNN (RGBI-Modell) 2020

Landcover-Klassen

- Gebäude
- Versiegelte Flächen
- Unversiegelte Flächen
- Gewässer
- Nadelbäume
- Laubbäume
- Sonstiges

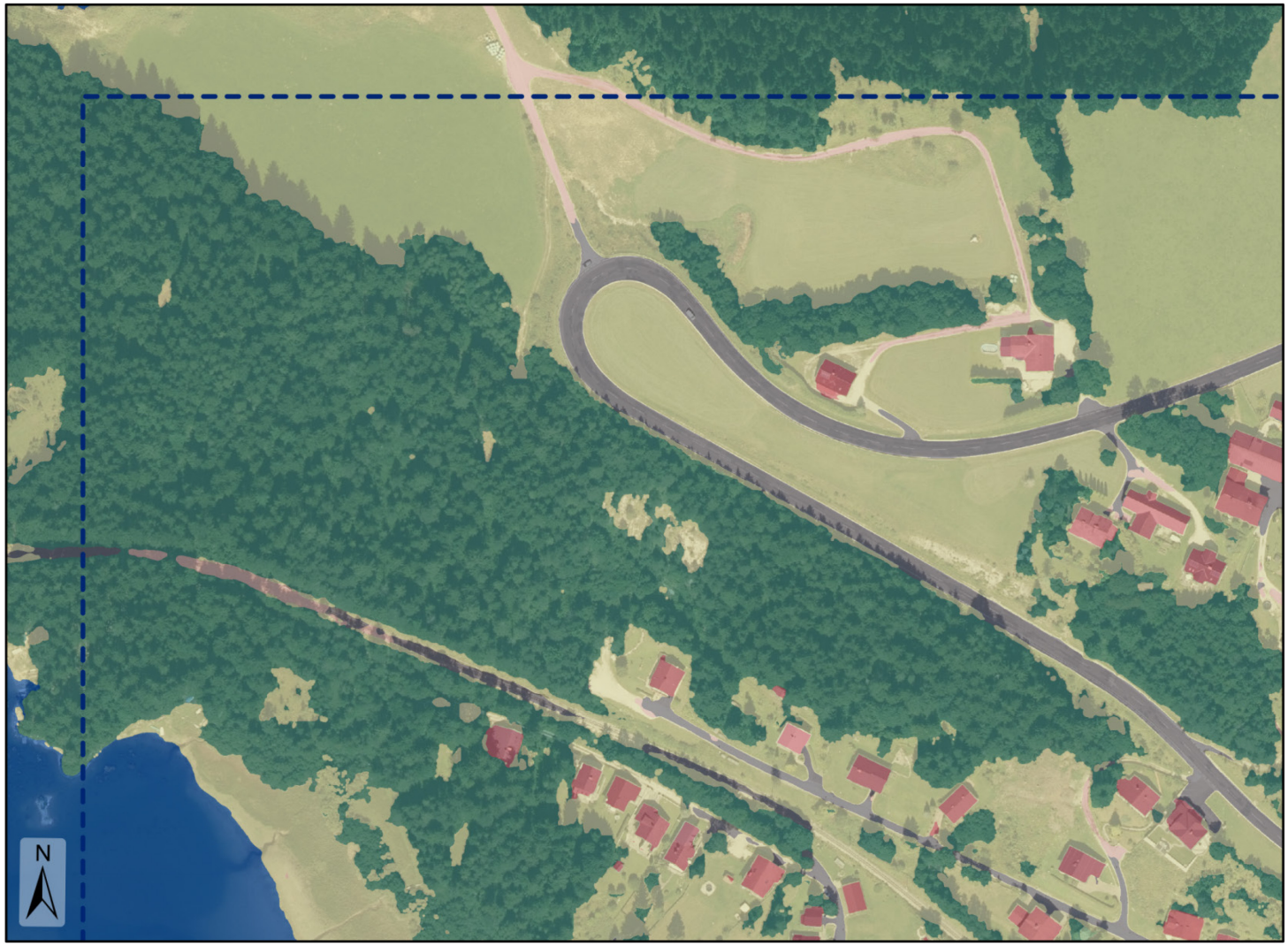
Referenzgebiet Mariazell

0 30 60 90 120 m

Maßstab: 1:3.000
Seite 1 von 6

Erstellung: Felix Pekárek

Datengrundlage Orthofoto: Bundesamt für Eich- und Vermessungswesen 2020



Klassifikation von Mariazell (ÖLK: 6830-100) mithilfe eines CNN (RGBI-Modell) 2020

Landcover-Klassen

- Gebäude
- Versiegelte Flächen
- Unversiegelte Flächen
- Gewässer
- Nadelbäume
- Laubbäume
- Sonstiges

Referenzgebiet Mariazell

0 30 60 90 120 m

Maßstab: 1:3.000

Seite 2 von 6

Erstellung: Felix Pekárek

Datengrundlage Orthofoto: Bundesamt für Eich- und Vermessungswesen 2020

Klassifikation von Mariazell (ÖLK: 6830-100) mithilfe eines CNN (RGBI-Modell) 2020

Landcover-Klassen

- Gebäude
- Versiegelte Flächen
- Unversiegelte Flächen
- Gewässer
- Nadelbäume
- Laubbäume
- Sonstiges

Referenzgebiet Mariazell

0 30 60 90 120 m

Maßstab: 1:3.000

Seite 3 von 6

Erstellung: Felix Pekárek

Datengrundlage Orthofoto: Bundesamt für Eich- und Vermessungswesen 2020

Klassifikation von Mariazell (ÖLK: 6830-100) mithilfe eines CNN (RGBI-Modell) 2020

Landcover-Klassen

- Gebäude
- Versiegelte Flächen
- Unversiegelte Flächen
- Gewässer
- Nadelbäume
- Laubbäume
- Sonstiges

Referenzgebiet Mariazell

0 30 60 90 120 m

Maßstab: 1:3.000

Seite 4 von 6

Erstellung: Felix Pekárek

Datengrundlage Orthofoto: Bundesamt für Eich- und Vermessungswesen 2020

Klassifikation von Mariazell (ÖLK: 6830-100) mithilfe eines CNN (RGBI-Modell) 2020

Landcover-Klassen

- Gebäude
- Versiegelte Flächen
- Unversiegelte Flächen
- Gewässer
- Nadelbäume
- Laubbäume
- Sonstiges

Referenzgebiet Mariazell

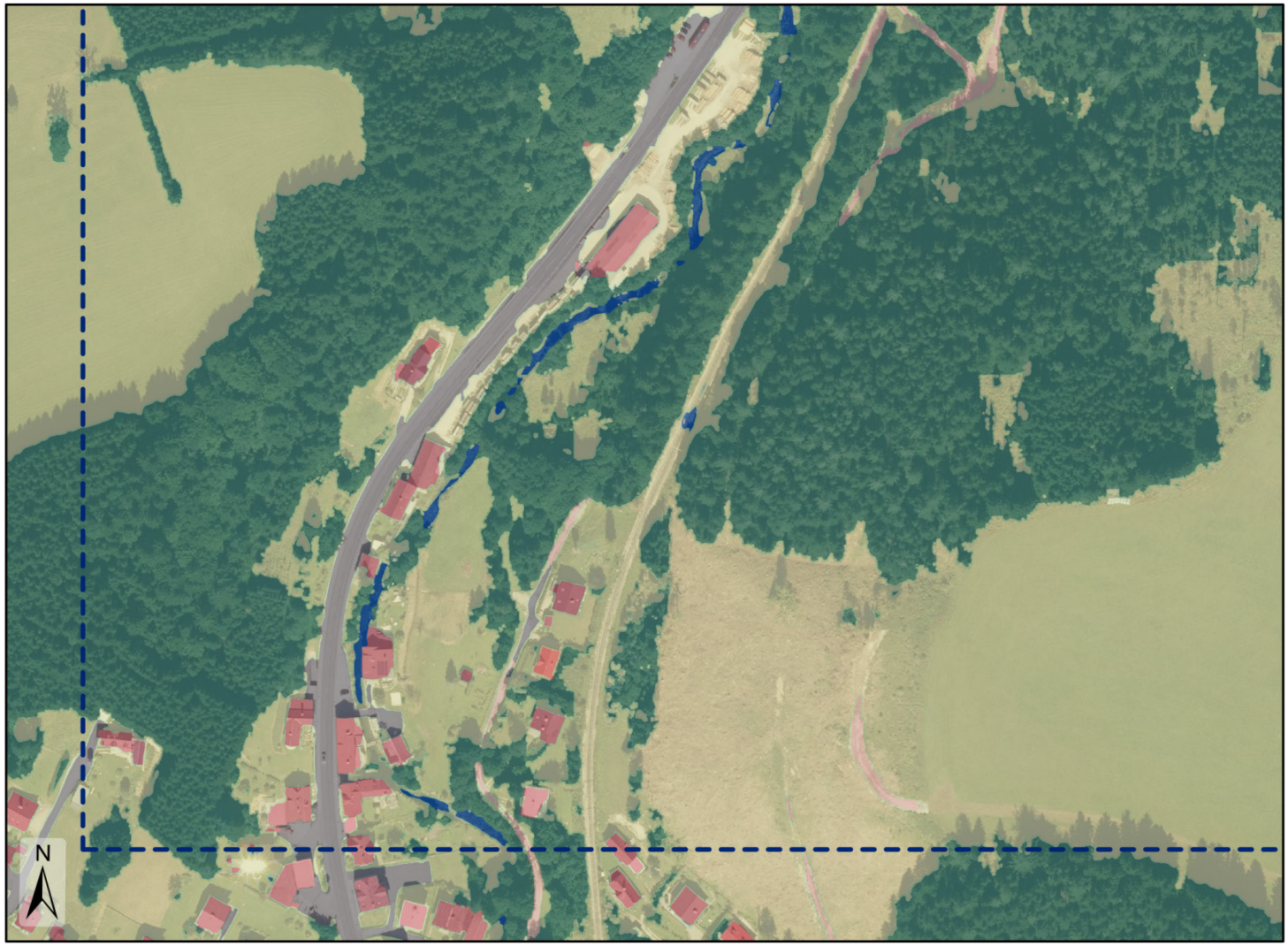
0 30 60 90 120 m

Maßstab: 1:3.000

Seite 5 von 6

Erstellung: Felix Pekárek

Datengrundlage Orthofoto: Bundesamt für Eich- und Vermessungswesen 2020



Klassifikation von Mariazell (ÖLK: 6830-100) mithilfe eines CNN (RGBI-Modell) 2020

Landcover-Klassen

- Gebäude
- Versiegelte Flächen
- Unversiegelte Flächen
- Gewässer
- Nadelbäume
- Laubbäume
- Sonstiges

Referenzgebiet Mariazell

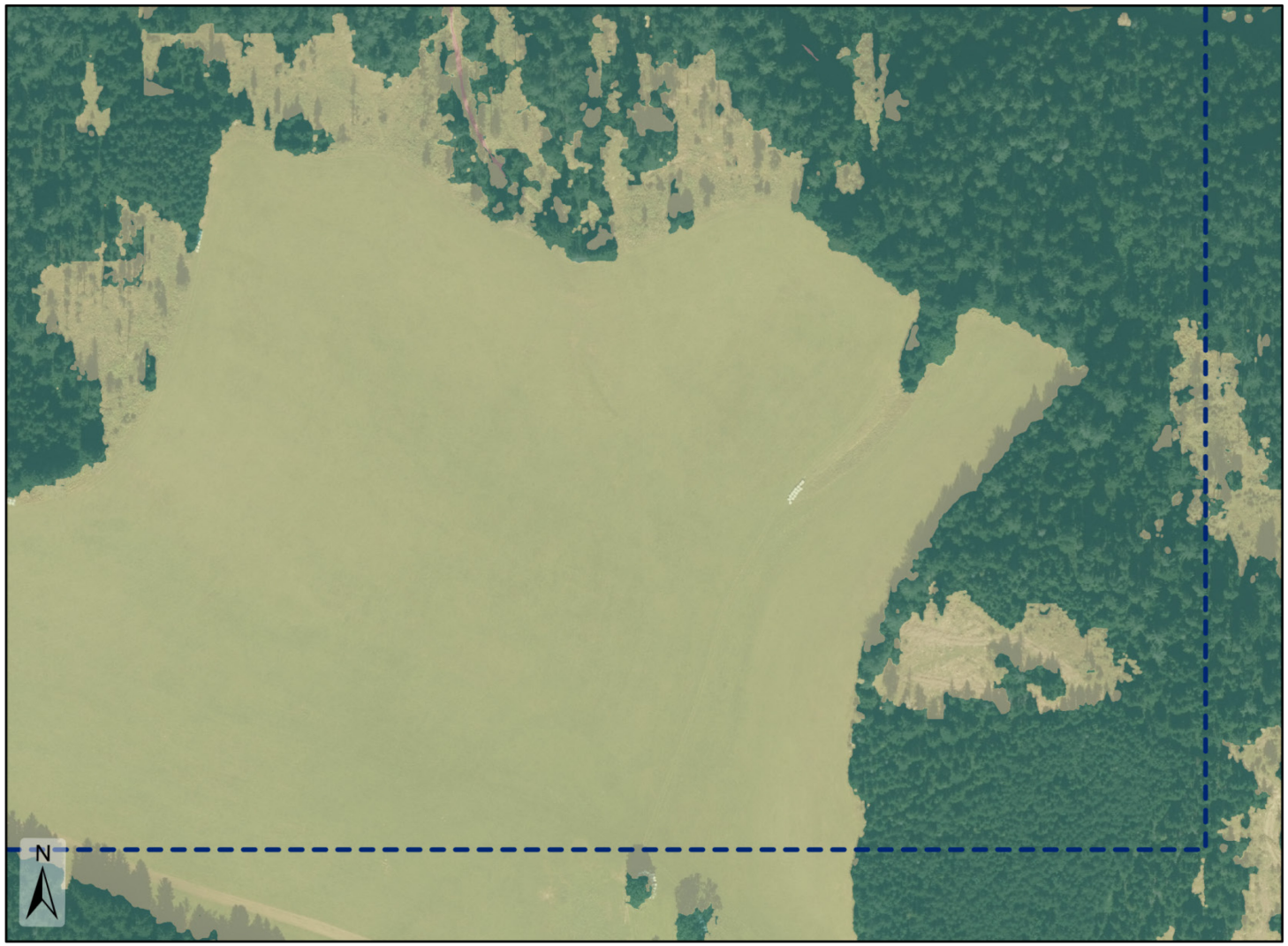
0 30 60 90 120 m

Maßstab: 1:3.000

Seite 6 von 6

Erstellung: Felix Pekárek

Datengrundlage Orthofoto: Bundesamt für Eich- und Vermessungswesen 2020



Klassifikation von Mariazell (ÖLK: 6830-100) mithilfe eines CNN (RGBI- und nDSM-Modell) 2020

Landcover-Klassen

- Gebäude
- Versiegelte Flächen
- Unversiegelte Flächen
- Gewässer
- Nadelbäume
- Laubbäume
- Sonstiges

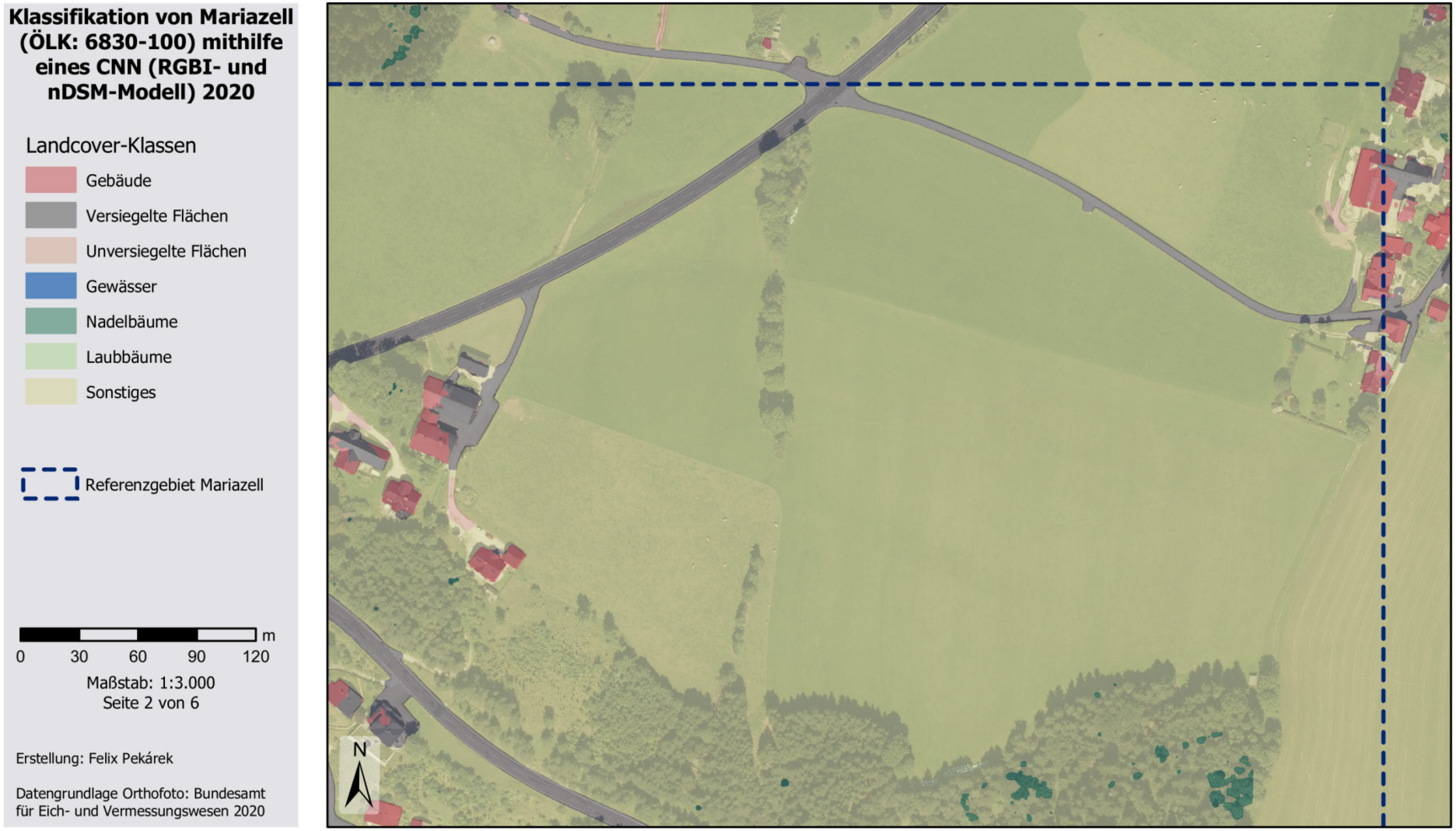
Referenzgebiet Mariazell

0 30 60 90 120 m

Maßstab: 1:3.000
Seite 1 von 6

Erstellung: Felix Pekárek

Datengrundlage Orthofoto: Bundesamt für Eich- und Vermessungswesen 2020


Klassifikation von Mariazell (ÖLK: 6830-100) mithilfe eines CNN (RGBI- und nDSM-Modell) 2020
Landcover-Klassen
Gebäude
Versiegelte Flächen
Unversiegelte Flächen
Gewässer
Nadelbäume
Laubbäume
Sonstiges
Referenzgebiet Mariazell
0
30
60
90
120
m
Maßstab: 1:3.000
Seite 2 von 6
Erstellung: Felix Pekárek
Datengrundlage Orthofoto: Bundesamt für Eich- und Vermessungswesen 2020
N

Klassifikation von Mariazell (ÖLK: 6830-100) mithilfe eines CNN (RGBI- und nDSM-Modell) 2020

Maßstab: 1:3.000
Seite 3 von 6

Erstellung: Felix Pekárek

Datengrundlage Orthofoto: Bundesamt für Eich- und Vermessungswesen 2020

Klassifikation von Mariazell (ÖLK: 6830-100) mithilfe eines CNN (RGBI- und nDSM-Modell) 2020

Landcover-Klassen

- Gebäude
- Versiegelte Flächen
- Unversiegelte Flächen
- Gewässer
- Nadelbäume
- Laubbäume
- Sonstiges

Referenzgebiet Mariazell

0 30 60 90 120 m

Maßstab: 1:3.000

Seite 4 von 6

Erstellung: Felix Pekárek

Datengrundlage Orthofoto: Bundesamt für Eich- und Vermessungswesen 2020

Klassifikation von Mariazell (ÖLK: 6830-100) mithilfe eines CNN (RGBI- und nDSM-Modell) 2020

Landcover-Klassen

- Gebäude
- Versiegelte Flächen
- Unversiegelte Flächen
- Gewässer
- Nadelbäume
- Laubbäume
- Sonstiges

Referenzgebiet Mariazell

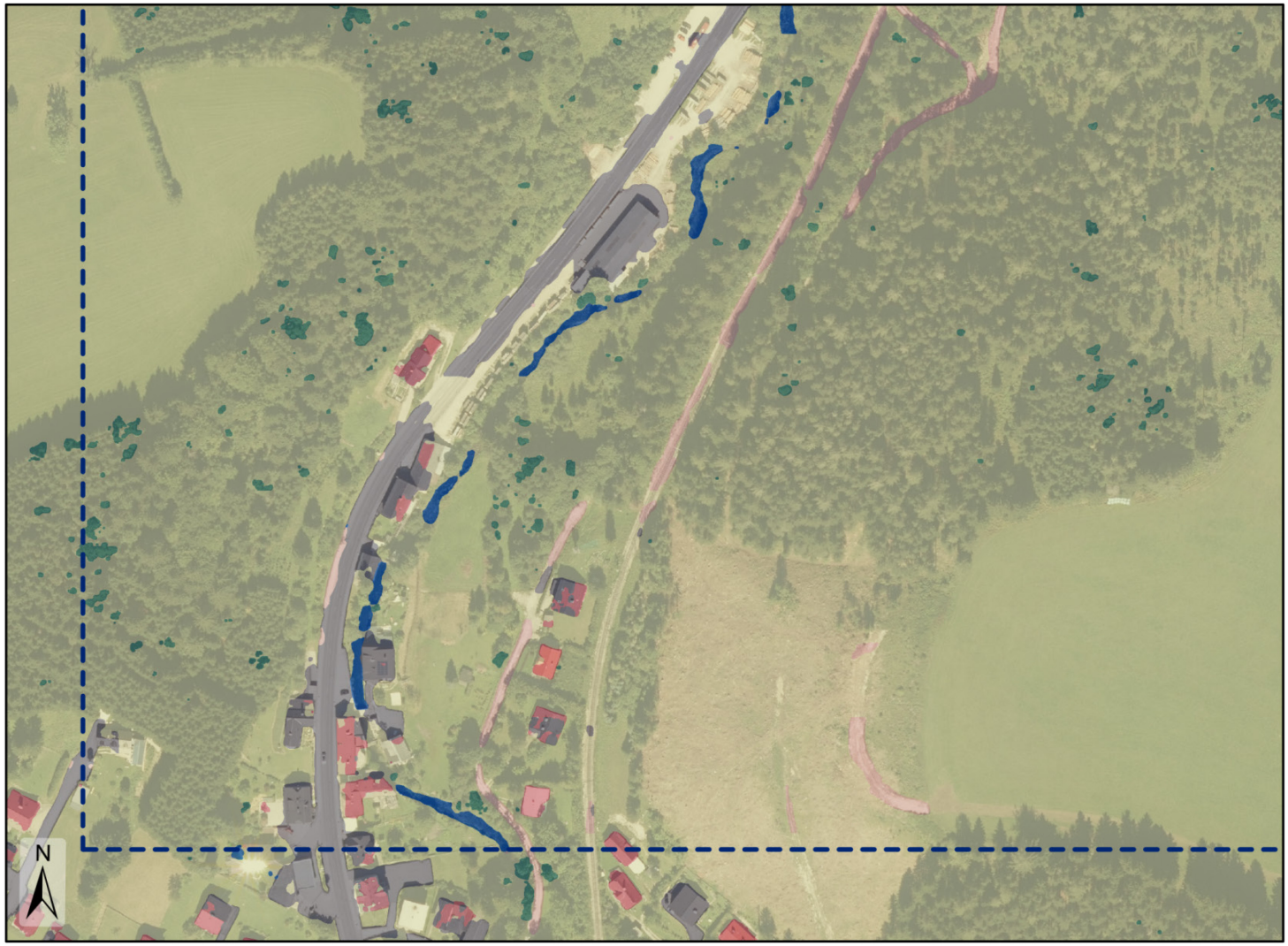
0 30 60 90 120 m

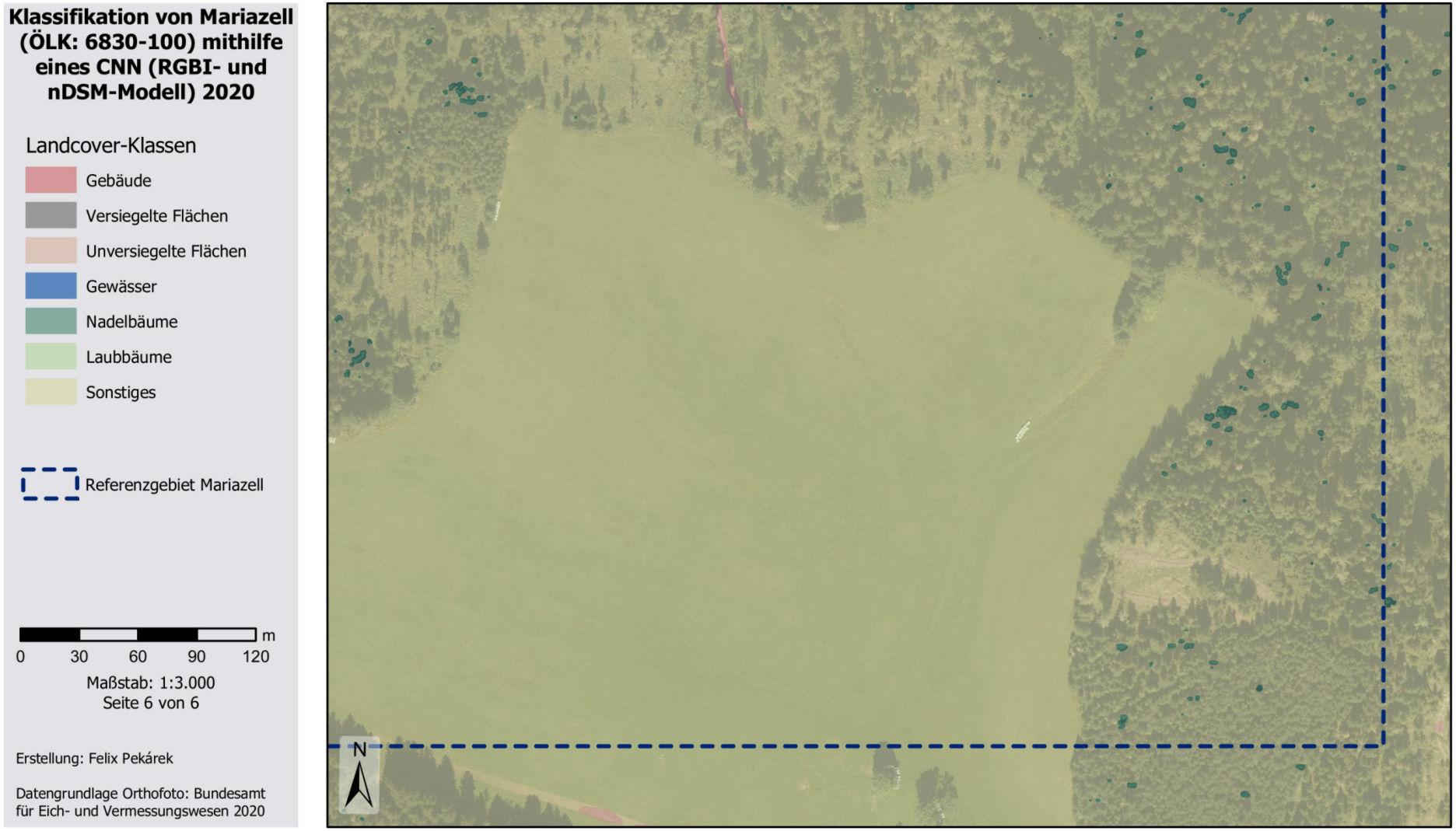
Maßstab: 1:3.000

Seite 5 von 6

Erstellung: Felix Pekárek

Datengrundlage Orthofoto: Bundesamt für Eich- und Vermessungswesen 2020




Klassifikation von Mariazell (ÖLK: 6830-100) mithilfe eines CNN (RGBI- und nDSM-Modell) 2020
Landcover-Klassen
Gebäude
Versiegelte Flächen
Unversiegelte Flächen
Gewässer
Nadelbäume
Laubbäume
Sonstiges
Referenzgebiet Mariazell
0 30 60 90 120 m
Maßstab: 1:3.000
Seite 6 von 6
Erstellung: Felix Pekárek
Datengrundlage Orthofoto: Bundesamt für Eich- und Vermessungswesen 2020
N

Klassifikation von Mariazell (ÖLK: 6830-100) durch das Bundesamt für Eich- und Vermessungswesen 2019

Landcover-Klassen

- Baum oder Wald
- Bodenflächen
- Buschwerk
- Gebäude
- Gewässer
- Niedrigvegetation

Referenzgebiet Mariazell

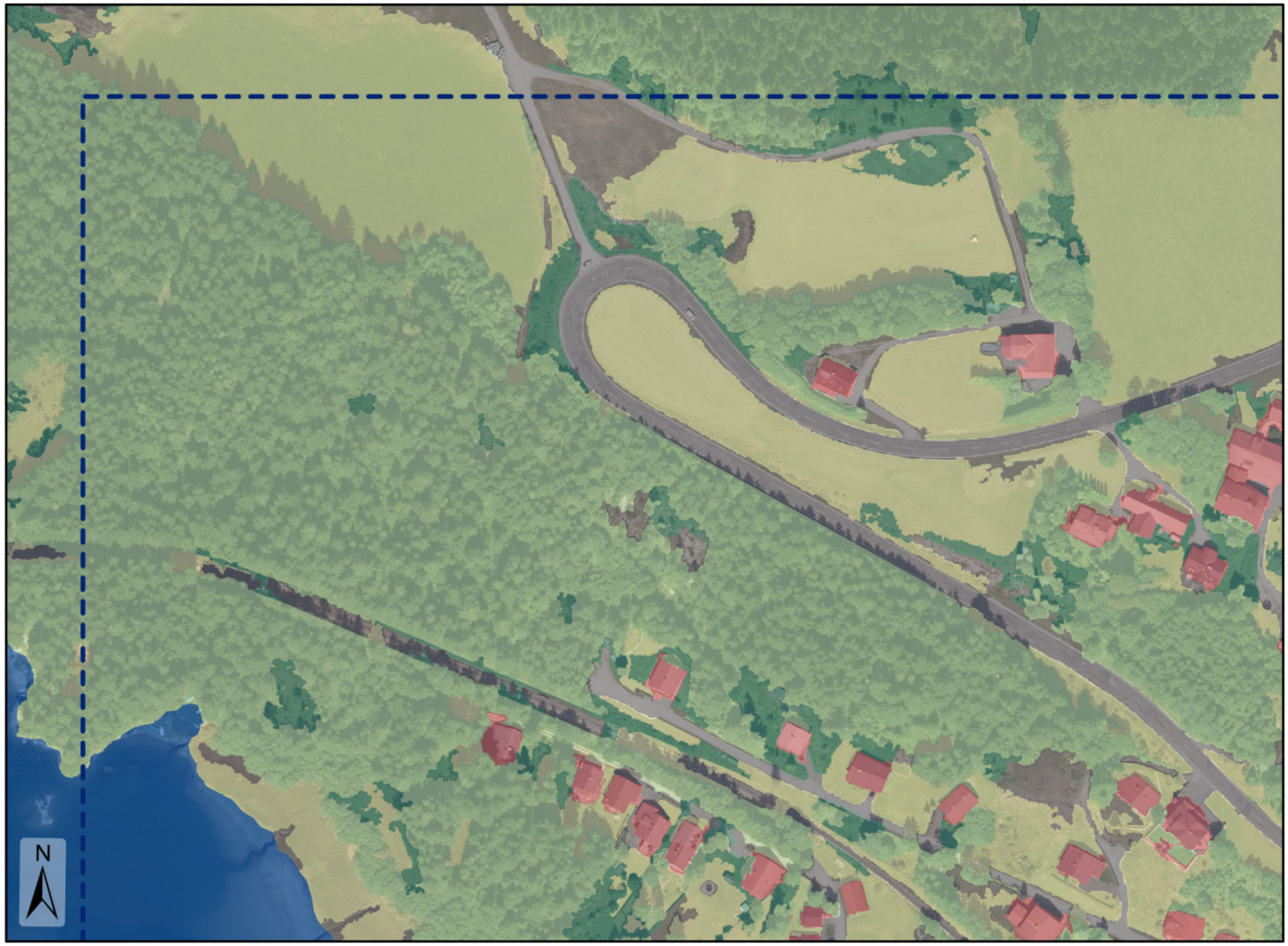
0 30 60 90 120 m

Maßstab: 1:3.000
Seite 1 von 6

Erstellung: Felix Pekárek

Datengrundlage Orthofoto: Bundesamt für Eich- und Vermessungswesen 2020

Datengrundlage Landcover-Klassifikation: Bundesamt für Eich- und Vermessungswesen 2020



Klassifikation von Mariazell (ÖLK: 6830-100) durch das Bundesamt für Eich- und Vermessungswesen 2019

Landcover-Klassen

- Baum oder Wald
- Bodenflächen
- Buschwerk
- Gebäude
- Gewässer
- Niedrigvegetation

Referenzgebiet Mariazell

0 30 60 90 120 m

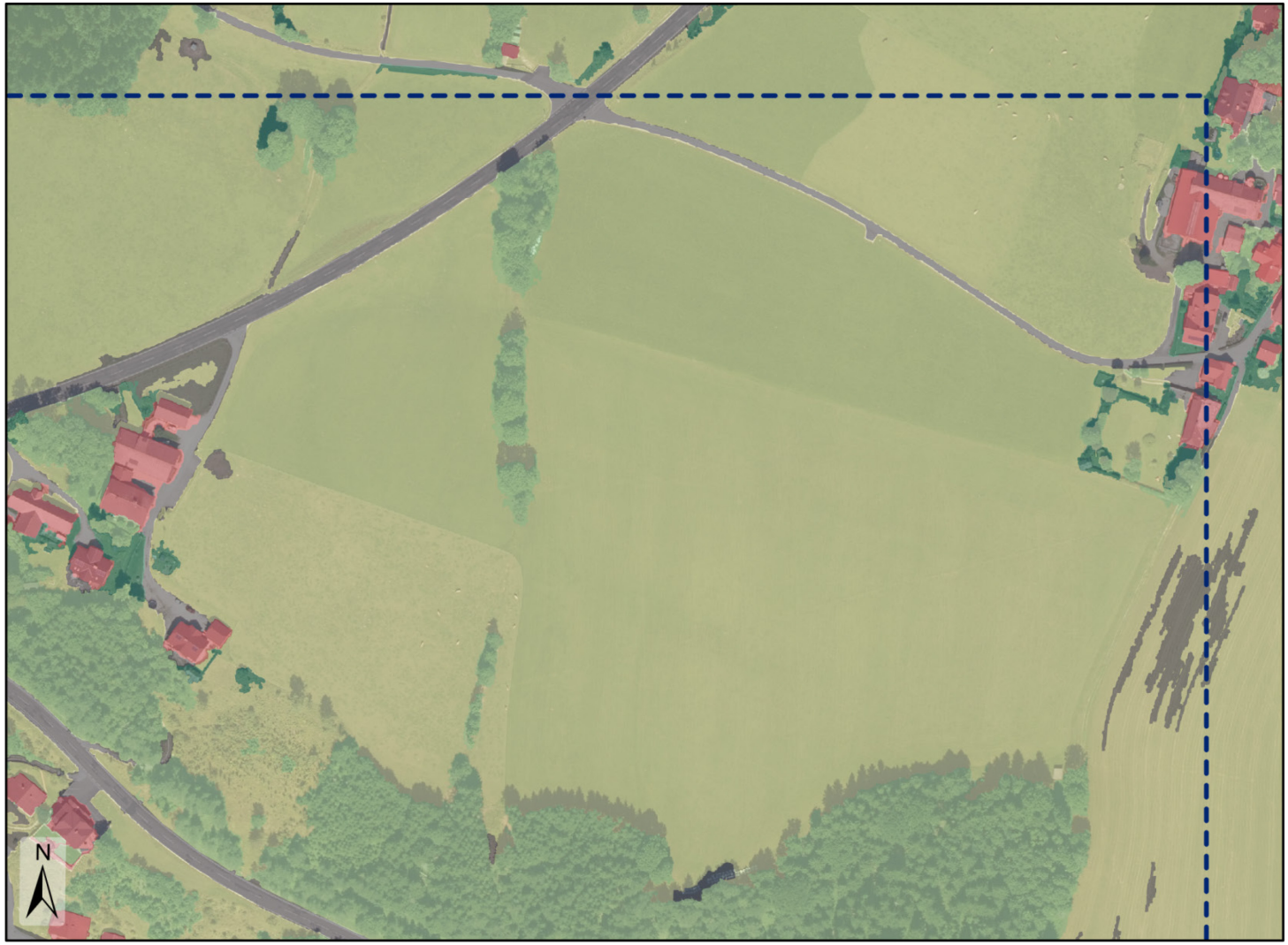
Maßstab: 1:3.000

Seite 2 von 6

Erstellung: Felix Pekárek

Datengrundlage Orthofoto: Bundesamt für Eich- und Vermessungswesen 2020

Datengrundlage Landcover-Klassifikation: Bundesamt für Eich- und Vermessungswesen 2020



Klassifikation von Mariazell (ÖLK: 6830-100) durch das Bundesamt für Eich- und Vermessungswesen 2019

Landcover-Klassen

- Baum oder Wald
- Bodenflächen
- Buschwerk
- Gebäude
- Gewässer
- Niedrigvegetation

Referenzgebiet Mariazell

0 30 60 90 120 m

Maßstab: 1:3.000
Seite 3 von 6

Erstellung: Felix Pekárek

Datengrundlage Orthofoto: Bundesamt für Eich- und Vermessungswesen 2020

Datengrundlage Landcover-Klassifikation: Bundesamt für Eich- und Vermessungswesen 2020

Klassifikation von Mariazell (ÖLK: 6830-100) durch das Bundesamt für Eich- und Vermessungswesen 2019

Landcover-Klassen

- Baum oder Wald
- Bodenflächen
- Buschwerk
- Gebäude
- Gewässer
- Niedrigvegetation

Referenzgebiet Mariazell

0 30 60 90 120 m

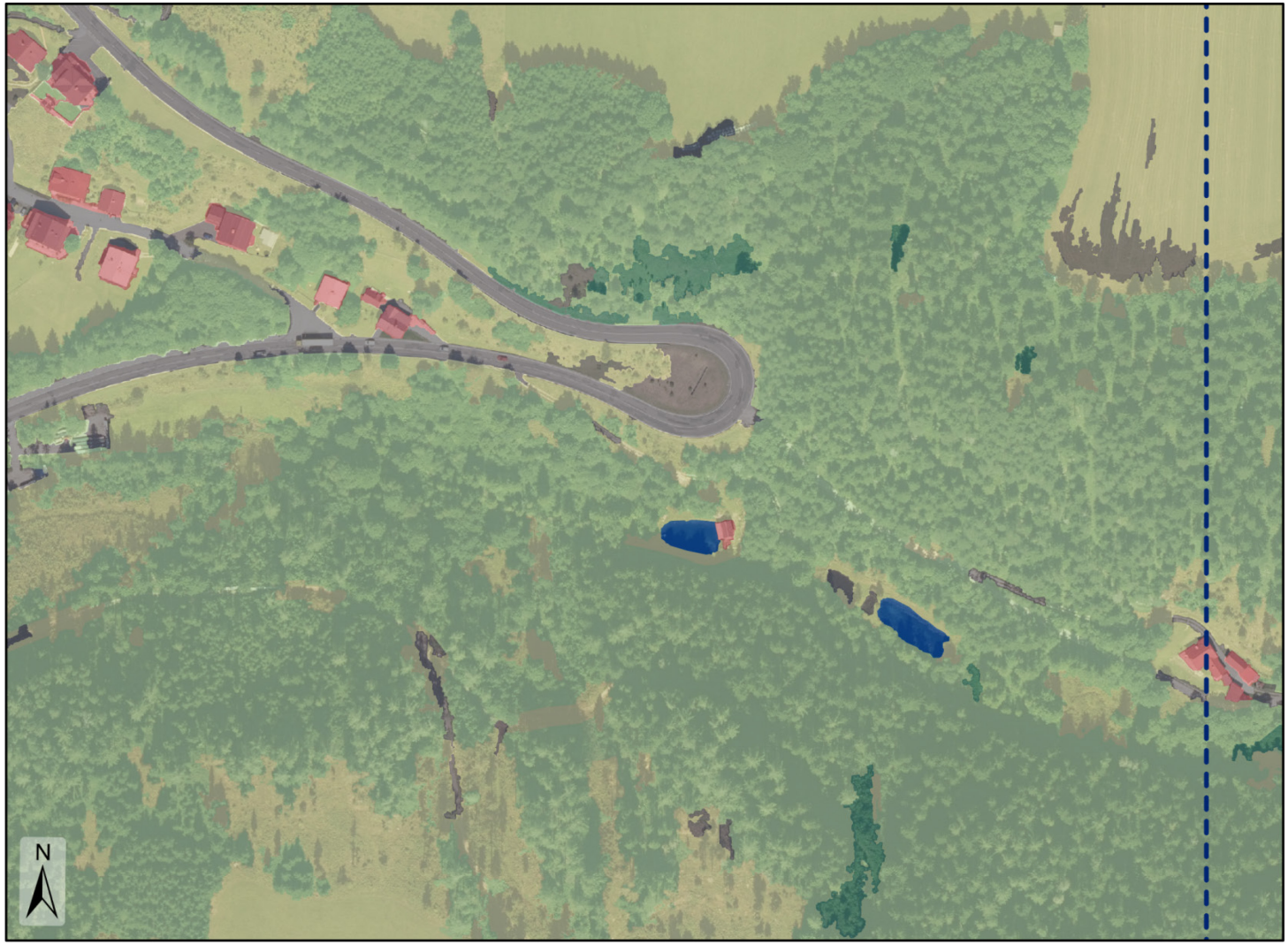
Maßstab: 1:3.000

Seite 4 von 6

Erstellung: Felix Pekárek

Datengrundlage Orthofoto: Bundesamt für Eich- und Vermessungswesen 2020

Datengrundlage Landcover-Klassifikation: Bundesamt für Eich- und Vermessungswesen 2020



Klassifikation von Mariazell (ÖLK: 6830-100) durch das Bundesamt für Eich- und Vermessungswesen 2019

Landcover-Klassen

- Baum oder Wald
- Bodenflächen
- Buschwerk
- Gebäude
- Gewässer
- Niedrigvegetation

Referenzgebiet Mariazell

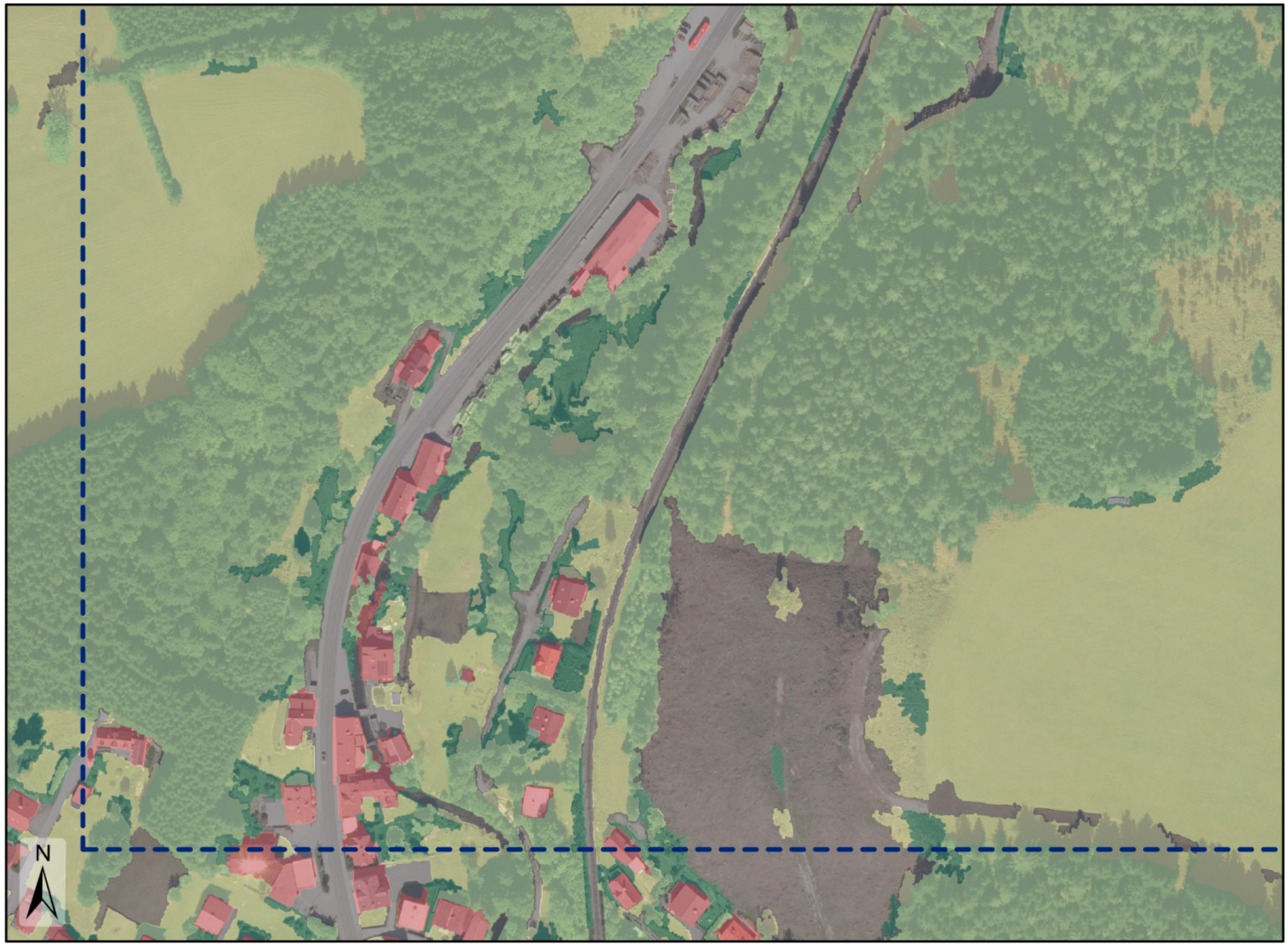
0 30 60 90 120 m

Maßstab: 1:3.000
Seite 5 von 6

Erstellung: Felix Pekárek

Datengrundlage Orthofoto: Bundesamt für Eich- und Vermessungswesen 2020

Datengrundlage Landcover-Klassifikation: Bundesamt für Eich- und Vermessungswesen 2020



Klassifikation von Mariazell (ÖLK: 6830-100) durch das Bundesamt für Eich- und Vermessungswesen 2019

Landcover-Klassen

- Baum oder Wald
- Bodenflächen
- Buschwerk
- Gebäude
- Gewässer
- Niedrigvegetation

Referenzgebiet Mariazell

0 30 60 90 120 m

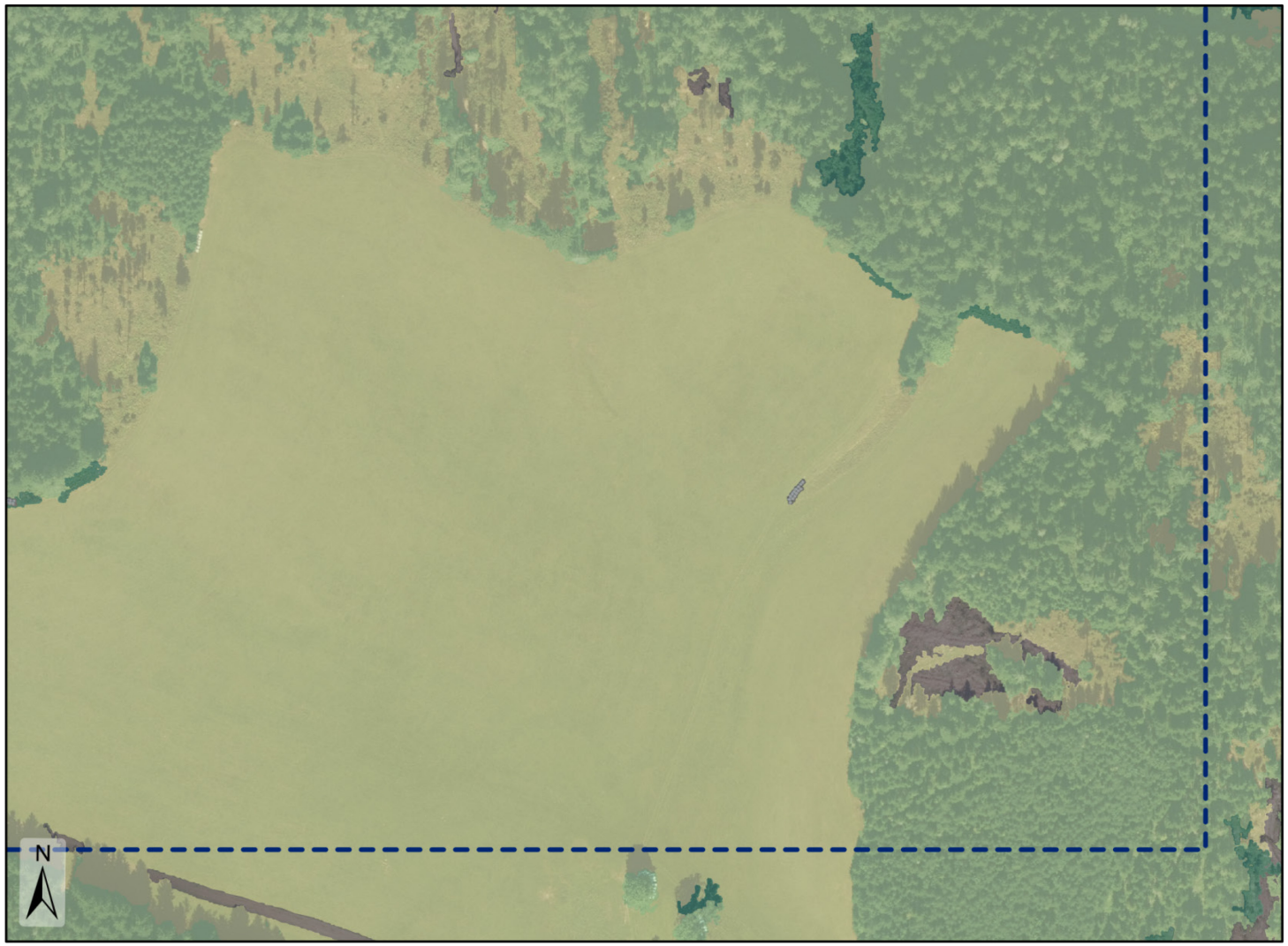
Maßstab: 1:3.000

Seite 6 von 6

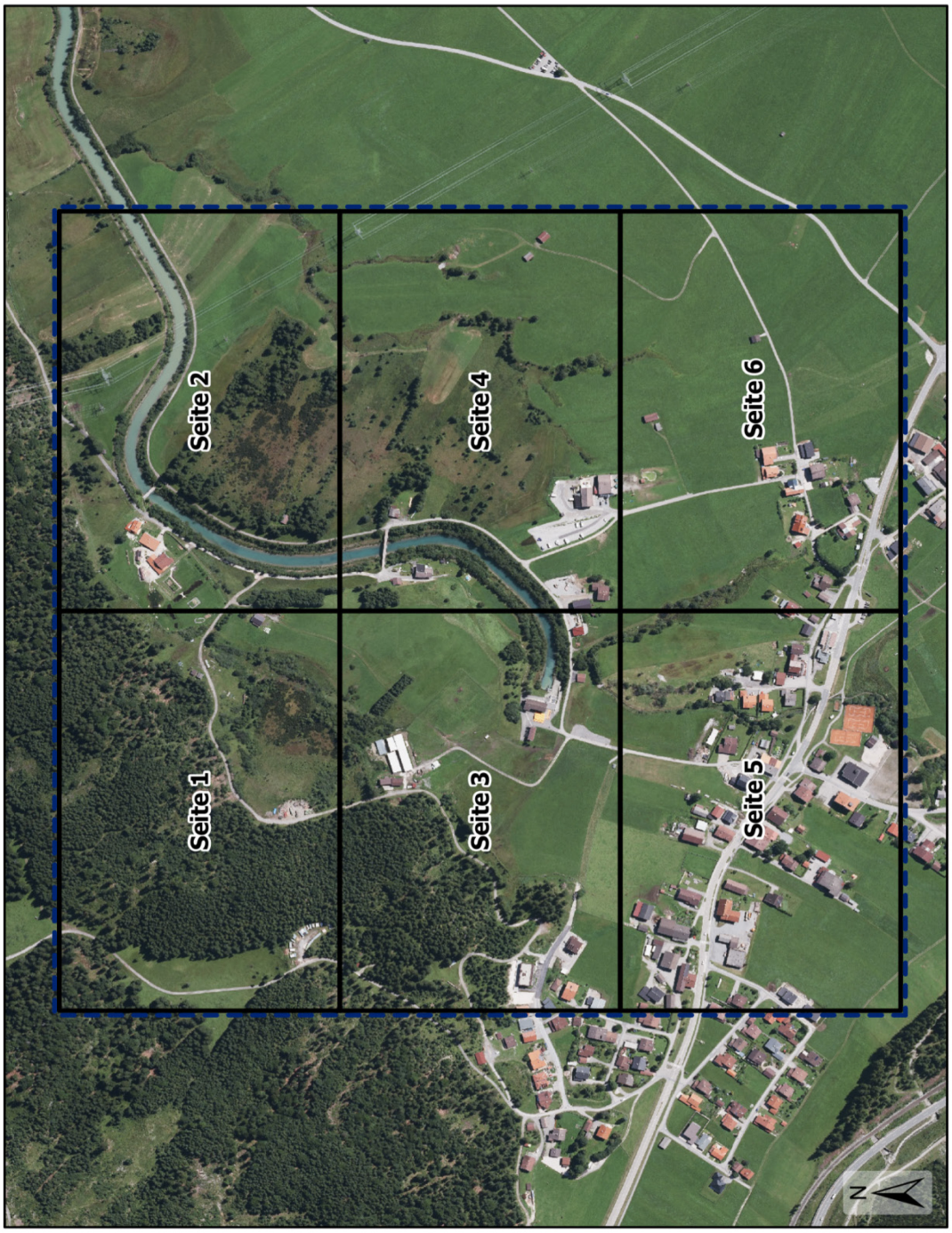
Erstellung: Felix Pekárek

Datengrundlage Orthofoto: Bundesamt für Eich- und Vermessungswesen 2020

Datengrundlage Landcover-Klassifikation: Bundesamt für Eich- und Vermessungswesen 2020



Anhang IX: Klassifikationsergebnisse für das Referenzgebiet in Imst



Überblick über den Referenzbereich in Imst (ÖLK: 1926-100) 2020

Die Karte liefert einen Überblick über den klassifizierten Bereich in Imst und gibt die Seitennummern für die Ausschnitte in den Klassifikationsergebnissen an. In den nachfolgenden Anhängen sind die Klassifikationen durch das RGBI-Modell visualisiert.

0 80 160 240 320 m

Maßstab: 1:8.000

Erstellung: Felix Pekárek

Datengrundlage Orthofoto: Bundesamt für Eich- und Vermessungswesen 2020

Klassifikation von Imst (ÖLK: 1926-100) mithilfe eines CNN (RGBI-Modell) 2020

Landcover-Klassen

- Gebäude
- Versiegelte Flächen
- Unversiegelte Flächen
- Gewässer
- Nadelbäume
- Laubbäume
- Sonstiges

Referenzgebiet Imst

0 30 60 90 120 m

Maßstab: 1:3.000

Seite 1 von 6

Erstellung: Felix Pekárek

Datengrundlage Orthofoto: Bundesamt für Eich- und Vermessungswesen 2020

Klassifikation von Imst (ÖLK: 1926-100) mithilfe eines CNN (RGBI-Modell) 2020

Landcover-Klassen

- Gebäude
- Versiegelte Flächen
- Unversiegelte Flächen
- Gewässer
- Nadelbäume
- Laubbäume
- Sonstiges

Referenzgebiet Imst

0 30 60 90 120 m

Maßstab: 1:3.000

Seite 2 von 6

Erstellung: Felix Pekárek

Datengrundlage Orthofoto: Bundesamt für Eich- und Vermessungswesen 2020

Klassifikation von Imst (ÖLK: 1926-100) mithilfe eines CNN (RGBI-Modell) 2020

Landcover-Klassen

- Gebäude
- Versiegelte Flächen
- Unversiegelte Flächen
- Gewässer
- Nadelbäume
- Laubbäume
- Sonstiges

Referenzgebiet Imst

0 30 60 90 120 m

Maßstab: 1:3.000

Seite 3 von 6

Erstellung: Felix Pekárek

Datengrundlage Orthofoto: Bundesamt für Eich- und Vermessungswesen 2020

Klassifikation von Imst (ÖLK: 1926-100) mithilfe eines CNN (RGBI-Modell) 2020

Landcover-Klassen

- Gebäude
- Versiegelte Flächen
- Unversiegelte Flächen
- Gewässer
- Nadelbäume
- Laubbäume
- Sonstiges

Referenzgebiet Imst

0 30 60 90 120 m

Maßstab: 1:3.000
Seite 4 von 6

Erstellung: Felix Pekárek

Datengrundlage Orthofoto: Bundesamt für Eich- und Vermessungswesen 2020

Klassifikation von Imst (ÖLK: 1926-100) mithilfe eines CNN (RGBI-Modell) 2020

Landcover-Klassen

- Gebäude
- Versiegelte Flächen
- Unversiegelte Flächen
- Gewässer
- Nadelbäume
- Laubbäume
- Sonstiges

Referenzgebiet Imst

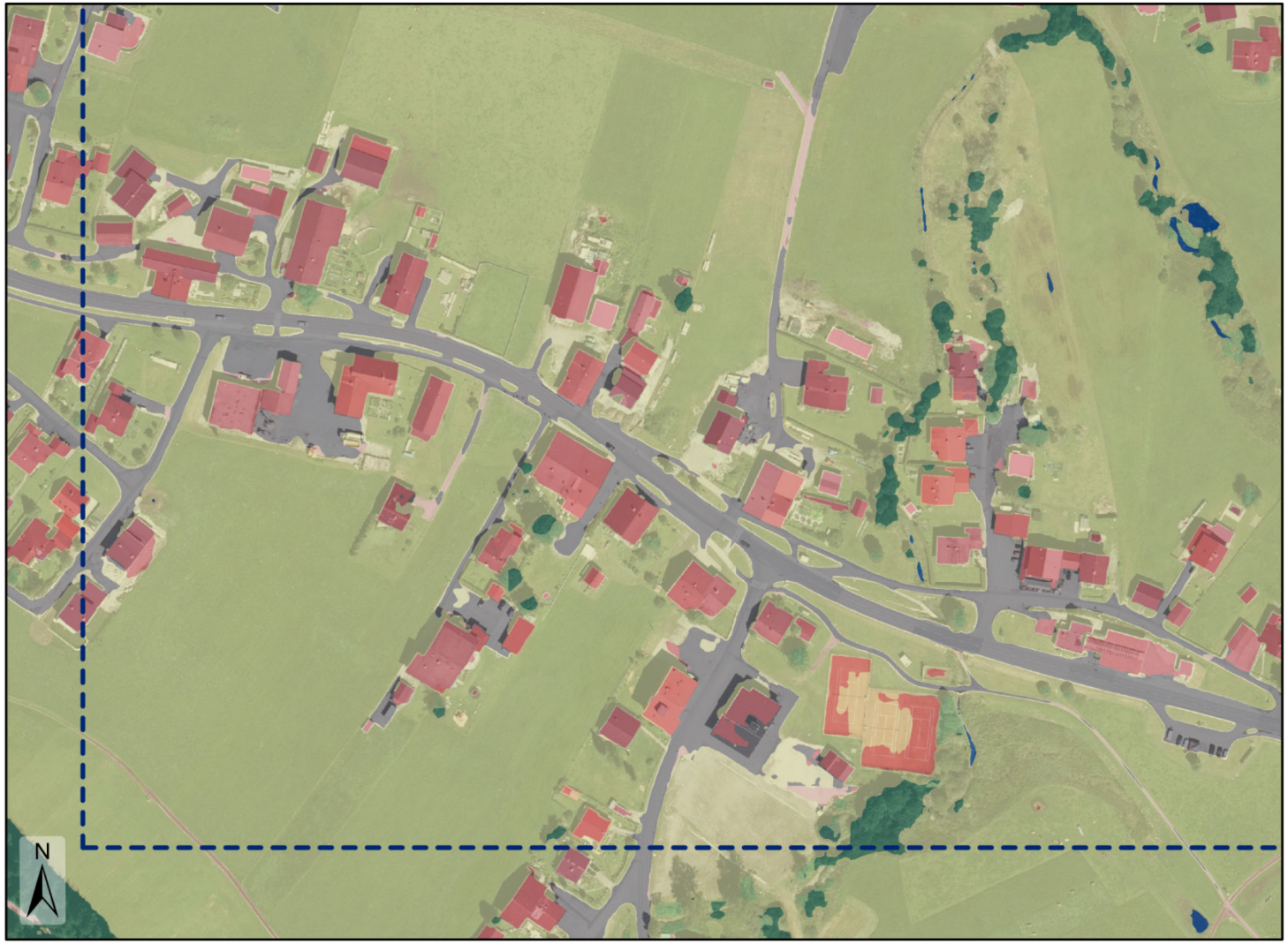
0 30 60 90 120 m

Maßstab: 1:3.000

Seite 5 von 6

Erstellung: Felix Pekárek

Datengrundlage Orthofoto: Bundesamt für Eich- und Vermessungswesen 2020



Klassifikation von Imst (ÖLK: 1926-100) mithilfe eines CNN (RGBI-Modell) 2020

Landcover-Klassen

- Gebäude
- Versiegelte Flächen
- Unversiegelte Flächen
- Gewässer
- Nadelbäume
- Laubbäume
- Sonstiges

Referenzgebiet Imst

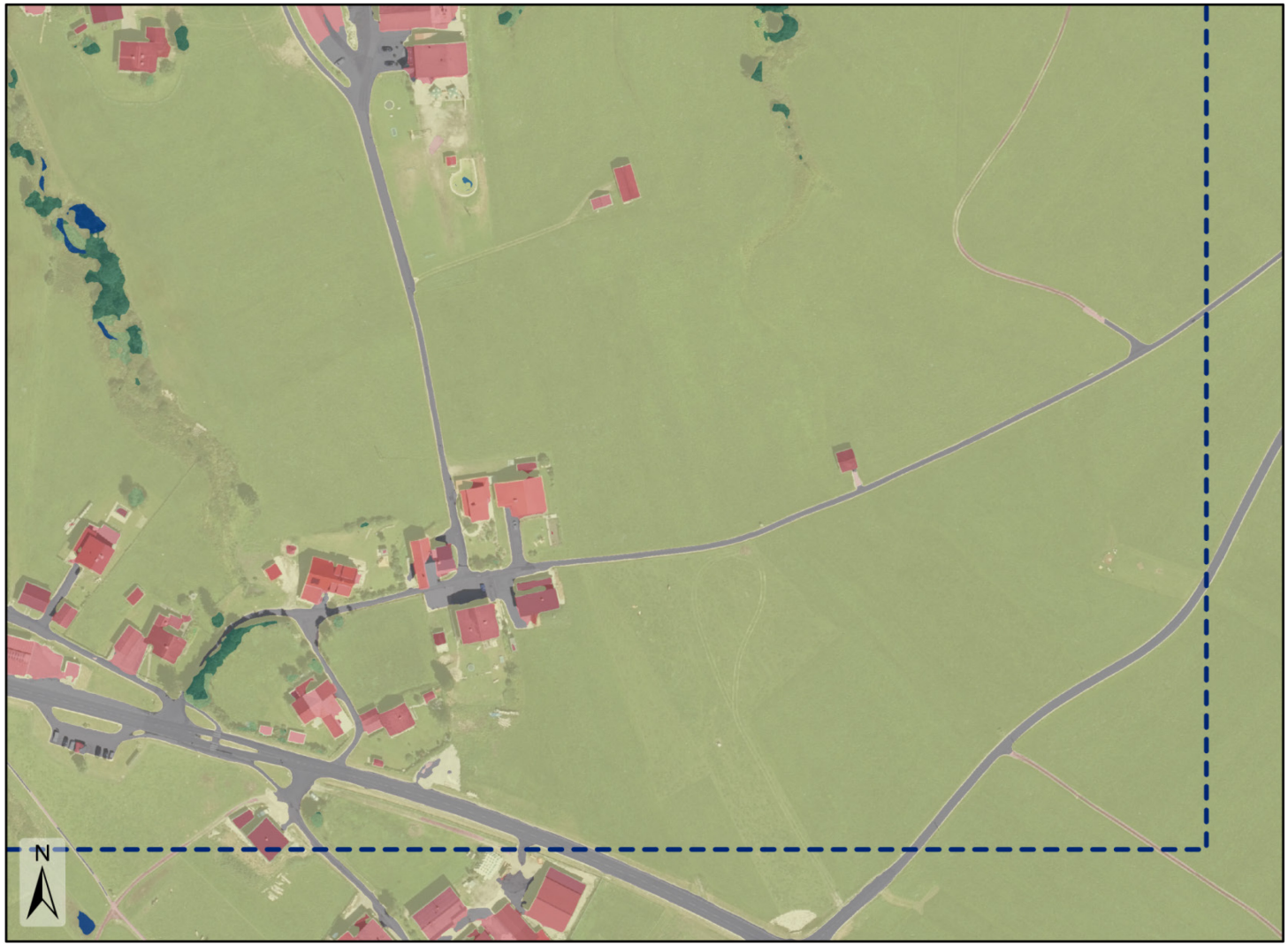
0 30 60 90 120 m

Maßstab: 1:3.000

Seite 6 von 6

Erstellung: Felix Pekárek

Datengrundlage Orthofoto: Bundesamt für Eich- und Vermessungswesen 2020



Klassifikation von Imst (ÖLK: 1926-100) durch das Bundesamt für Eich- und Vermessungswesen 2019

Landcover-Klassen

- Baum oder Wald
- Bodenflächen
- Buschwerk
- Gebäude
- Gewässer
- Niedrigvegetation

Referenzgebiet Imst

0 30 60 90 120 m

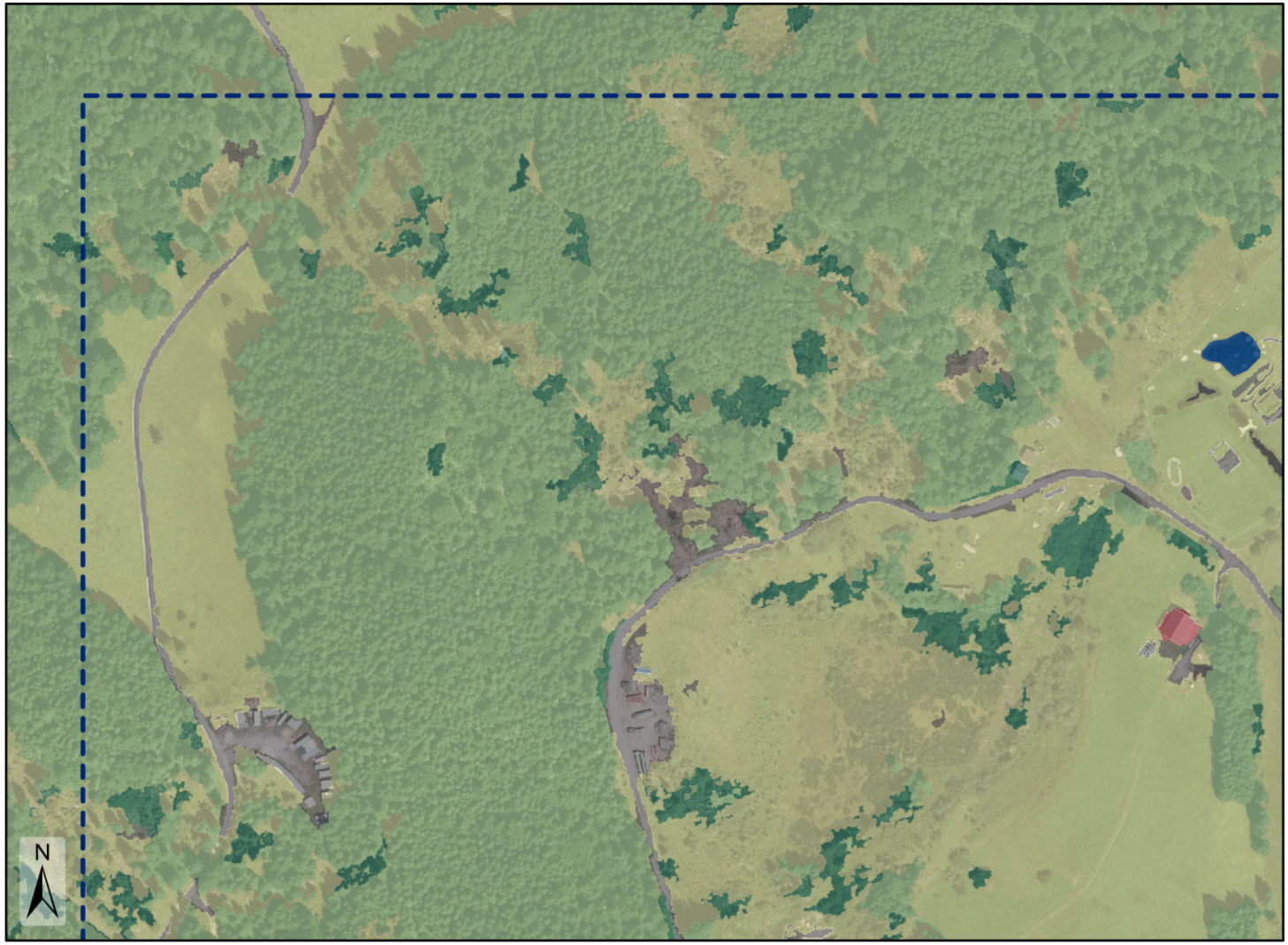
Maßstab: 1:3.000

Seite 1 von 6

Erstellung: Felix Pekárek

Datengrundlage Orthofoto: Bundesamt für Eich- und Vermessungswesen 2020

Datengrundlage Landcover-Klassifikation: Bundesamt für Eich- und Vermessungswesen 2020



Klassifikation von Imst (ÖLK: 1926-100) durch das Bundesamt für Eich- und Vermessungswesen 2019

Landcover-Klassen

- Baum oder Wald
- Bodenflächen
- Buschwerk
- Gebäude
- Gewässer
- Niedrigvegetation

Referenzgebiet Imst

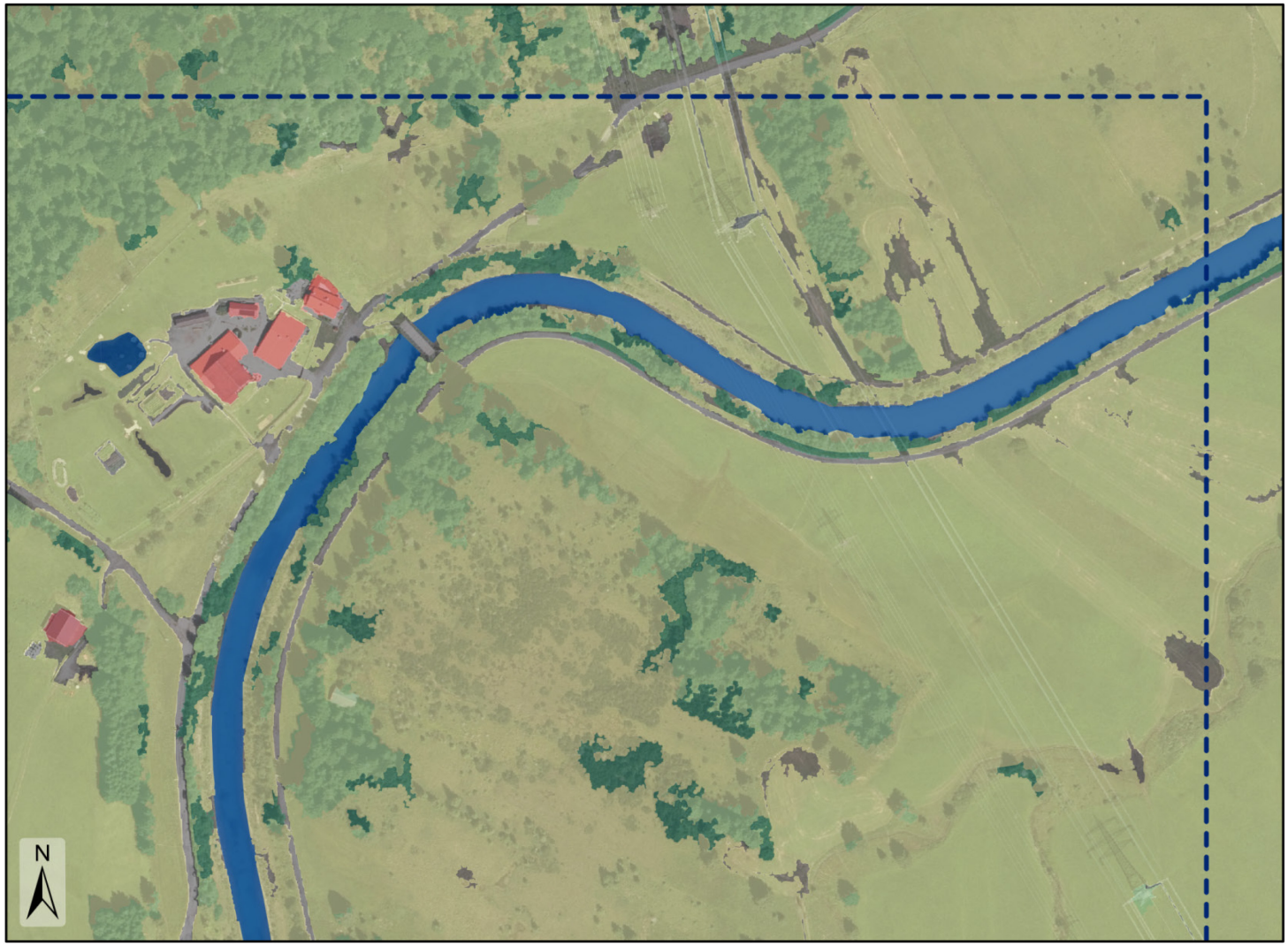
0 30 60 90 120 m

Maßstab: 1:3.000
Seite 2 von 6

Erstellung: Felix Pekárek

Datengrundlage Orthofoto: Bundesamt für Eich- und Vermessungswesen 2020

Datengrundlage Landcover-Klassifikation: Bundesamt für Eich- und Vermessungswesen 2020



Klassifikation von Imst (ÖLK: 1926-100) durch das Bundesamt für Eich- und Vermessungswesen 2019

Landcover-Klassen

- Baum oder Wald
- Bodenflächen
- Buschwerk
- Gebäude
- Gewässer
- Niedrigvegetation

Referenzgebiet Imst

0 30 60 90 120 m

Maßstab: 1:3.000

Seite 3 von 6

Erstellung: Felix Pekárek

Datengrundlage Orthofoto: Bundesamt für Eich- und Vermessungswesen 2020

Datengrundlage Landcover-Klassifikation: Bundesamt für Eich- und Vermessungswesen 2020

Klassifikation von Imst (ÖLK: 1926-100) durch das Bundesamt für Eich- und Vermessungswesen 2019

Landcover-Klassen

- Baum oder Wald
- Bodenflächen
- Buschwerk
- Gebäude
- Gewässer
- Niedrigvegetation

Referenzgebiet Imst

0 30 60 90 120 m

Maßstab: 1:3.000
Seite 4 von 6

Erstellung: Felix Pekárek

Datengrundlage Orthofoto: Bundesamt für Eich- und Vermessungswesen 2020

Datengrundlage Landcover-Klassifikation: Bundesamt für Eich- und Vermessungswesen 2020

Klassifikation von Imst (ÖLK: 1926-100) durch das Bundesamt für Eich- und Vermessungswesen 2019

Landcover-Klassen

- Baum oder Wald
- Bodenflächen
- Buschwerk
- Gebäude
- Gewässer
- Niedrigvegetation

Referenzgebiet Imst

0 30 60 90 120 m

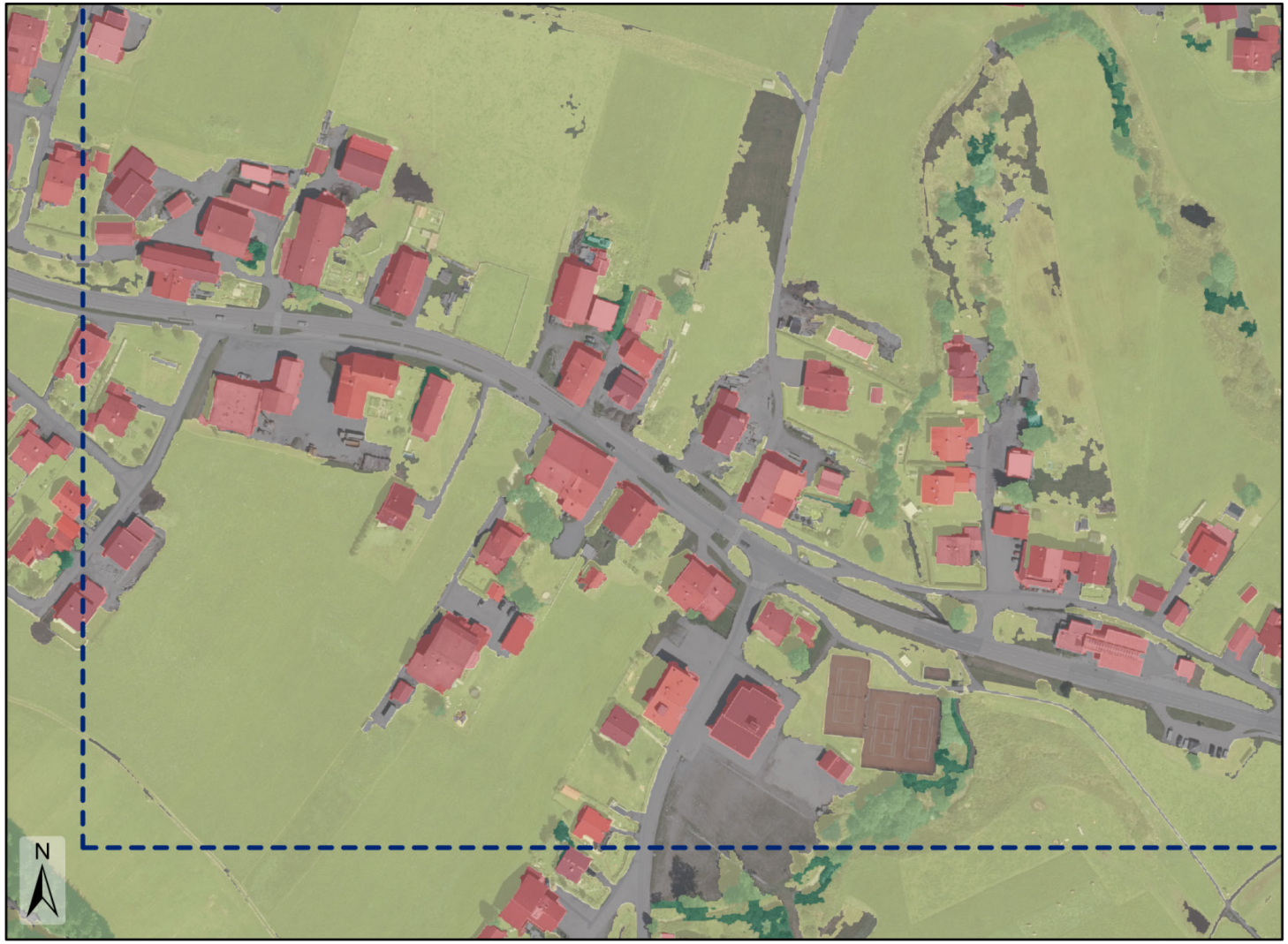
Maßstab: 1:3.000

Seite 5 von 6

Erstellung: Felix Pekárek

Datengrundlage Orthofoto: Bundesamt für Eich- und Vermessungswesen 2020

Datengrundlage Landcover-Klassifikation: Bundesamt für Eich- und Vermessungswesen 2020



Klassifikation von Imst (ÖLK: 1926-100) durch das Bundesamt für Eich- und Vermessungswesen 2019

Landcover-Klassen

- Baum oder Wald
- Bodenflächen
- Buschwerk
- Gebäude
- Gewässer
- Niedrigvegetation

Referenzgebiet Imst

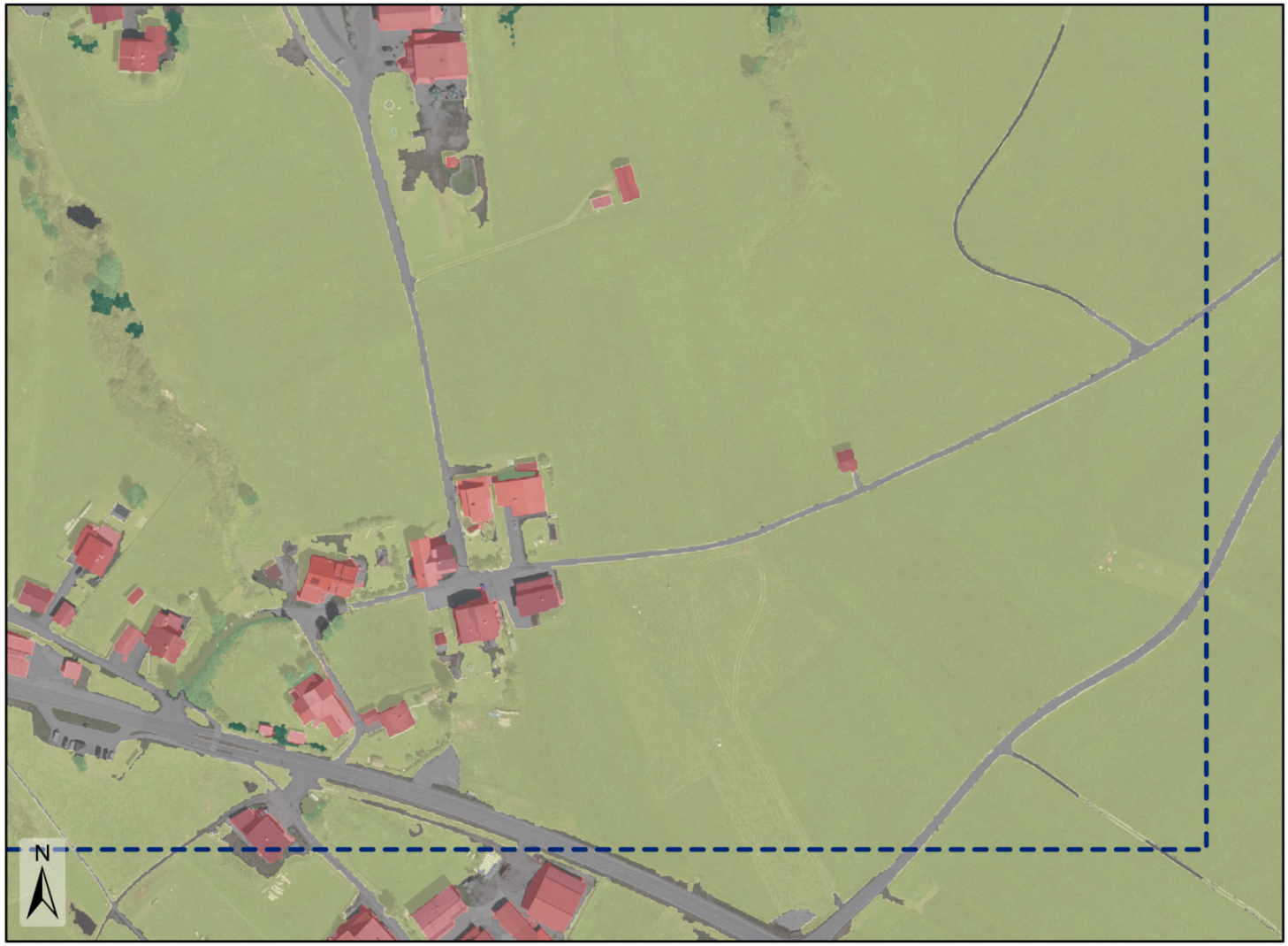
0 30 60 90 120 m

Maßstab: 1:3.000
Seite 6 von 6

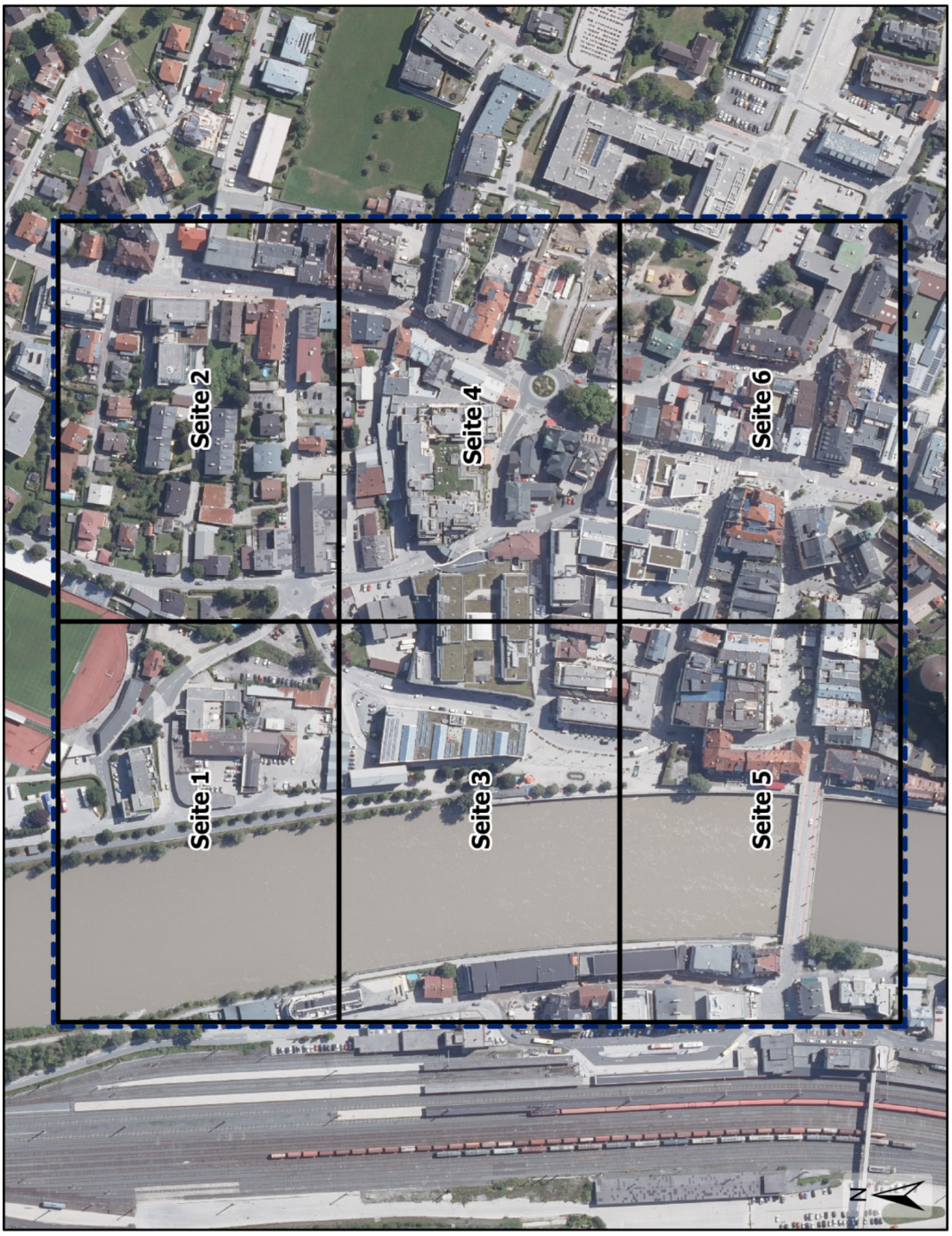
Erstellung: Felix Pekárek

Datengrundlage Orthofoto: Bundesamt für Eich- und Vermessungswesen 2020

Datengrundlage Landcover-Klassifikation: Bundesamt für Eich- und Vermessungswesen 2020



Anhang X: Klassifikationsergebnisse für das Referenzgebiet in Kufstein



Überblick über den Referenzbereich in Kufstein (ÖLK: 3728-102) 2020

Die Karte liefert einen Überblick über den klassifizierten Bereich in Kufstein und gibt die Seitennummern für die Ausschnitte in den Klassifikationsergebnissen an. In den nachfolgenden Anhängen sind die Klassifikationen durch das RGBI-Modell visualisiert.

0 40 80 120 160 m

Maßstab: 1:4.000

Erstellung: Felix Pekárek

Datengrundlage Orthofoto: Bundesamt für Eich- und Vermessungswesen 2020

Klassifikation von Kufstein (ÖLK: 3728-102) mithilfe eines CNN (RGBI-Modell) 2020

Landcover-Klassen

- Gebäude
- Versiegelte Flächen
- Unversiegelte Flächen
- Gewässer
- Nadelbäume
- Laubbäume
- Sonstiges

Referenzgebiet Kufstein

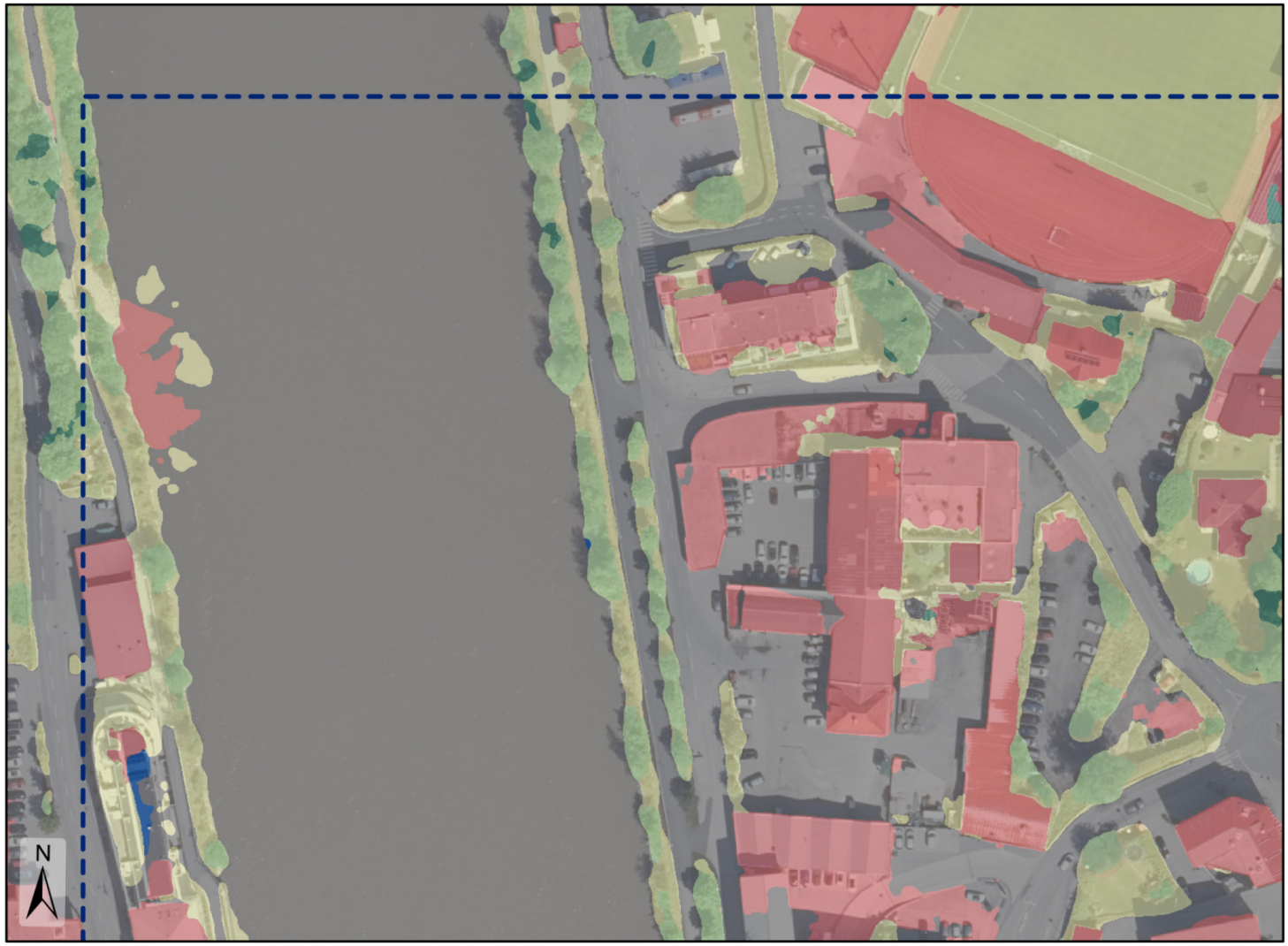
0 10 20 30 40 m

Maßstab: 1:1.500

Seite 1 von 6

Erstellung: Felix Pekárek

Datengrundlage Orthofoto: Bundesamt für Eich- und Vermessungswesen 2020



Klassifikation von Kufstein (ÖLK: 3728-102) mithilfe eines CNN (RGBI-Modell) 2020

Landcover-Klassen

- Gebäude
- Versiegelte Flächen
- Unversiegelte Flächen
- Gewässer
- Nadelbäume
- Laubbäume
- Sonstiges

Referenzgebiet Kufstein

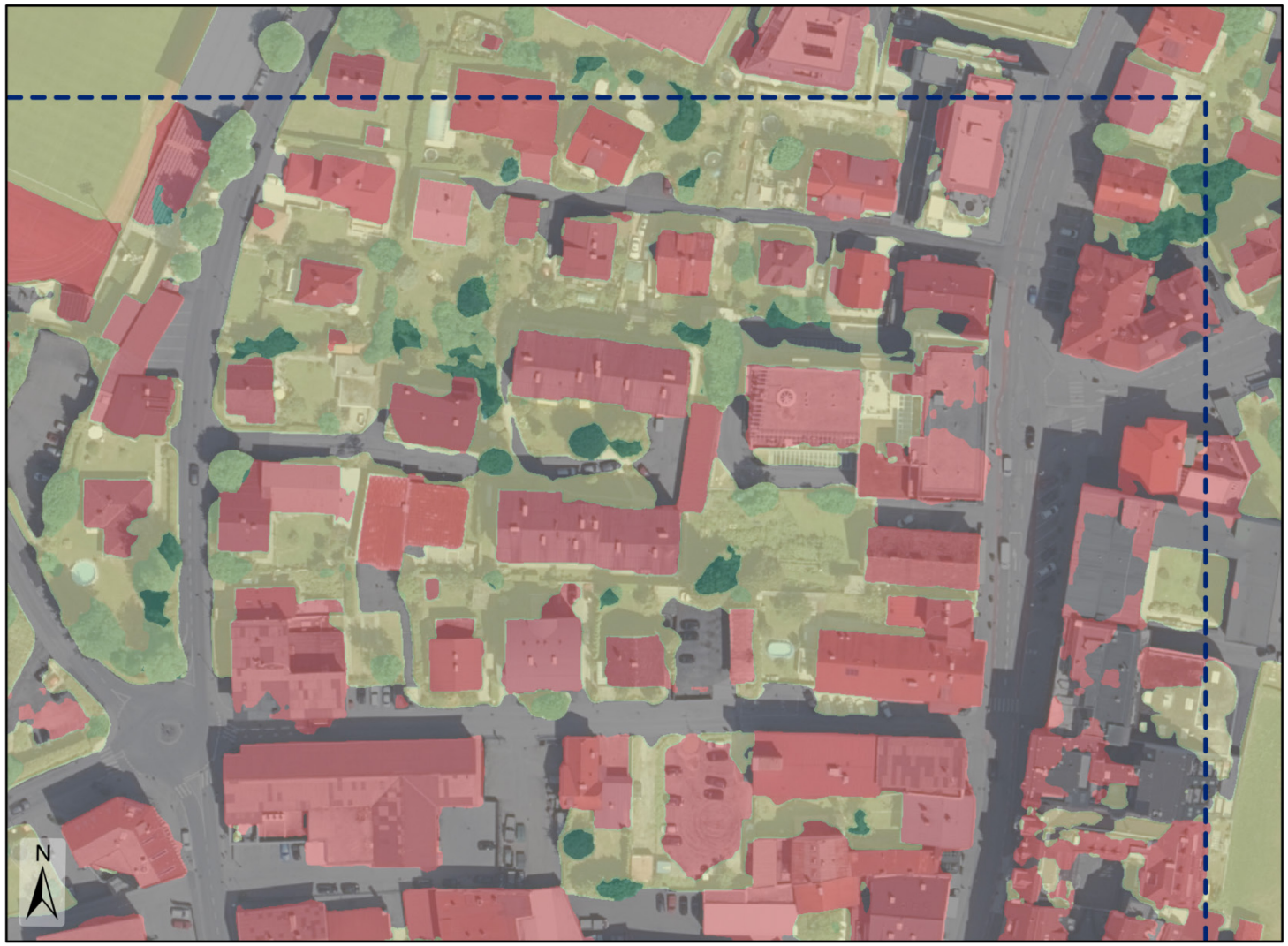
0 10 20 30 40 m

Maßstab: 1:1.500

Seite 2 von 6

Erstellung: Felix Pekárek

Datengrundlage Orthofoto: Bundesamt für Eich- und Vermessungswesen 2020



Klassifikation von Kufstein (ÖLK: 3728-102) mithilfe eines CNN (RGBI-Modell) 2020

Landcover-Klassen

- Gebäude
- Versiegelte Flächen
- Unversiegelte Flächen
- Gewässer
- Nadelbäume
- Laubbäume
- Sonstiges

Referenzgebiet Kufstein

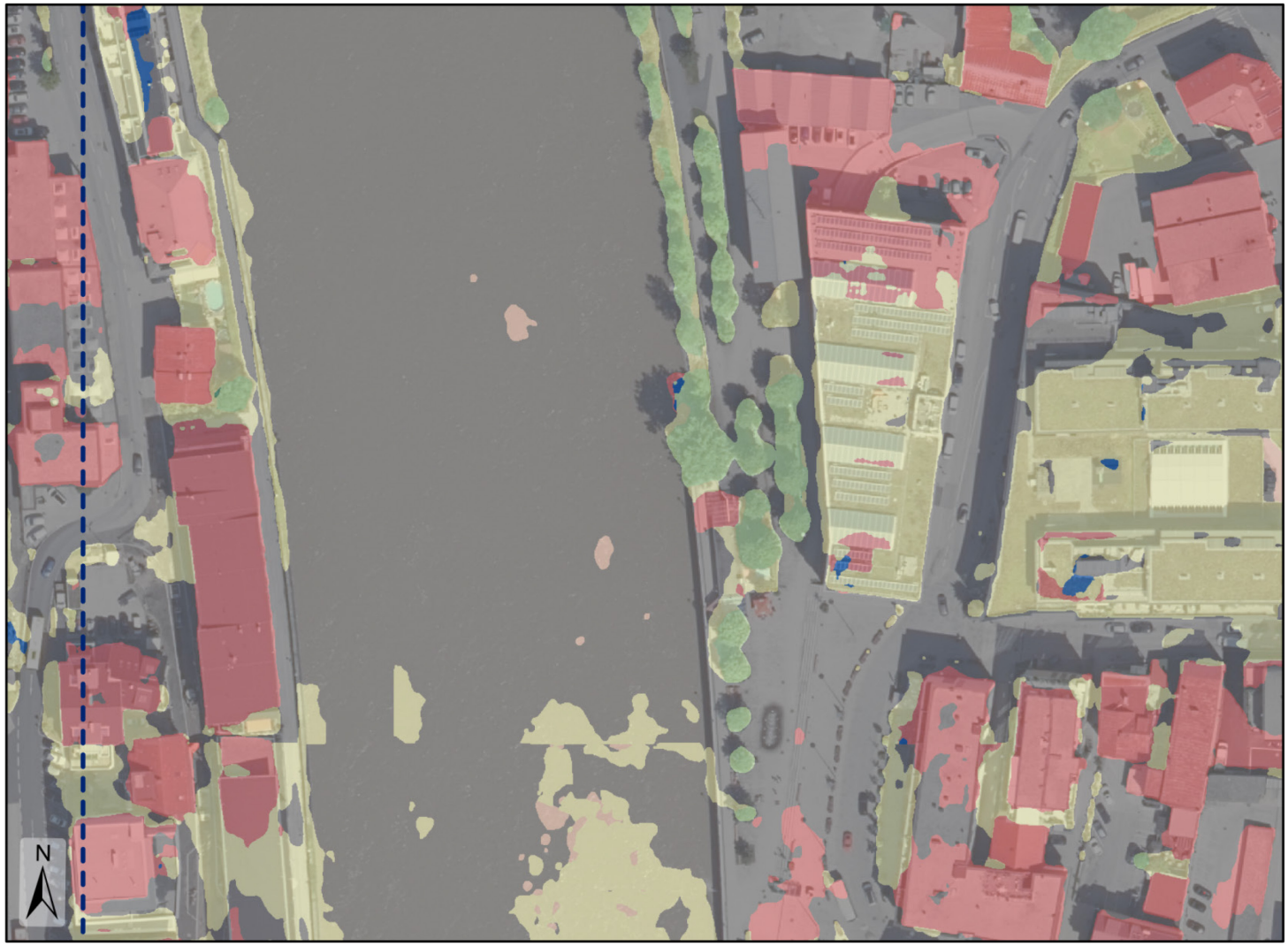
0 10 20 30 40 m

Maßstab: 1:1.500

Seite 3 von 6

Erstellung: Felix Pekárek

Datengrundlage Orthofoto: Bundesamt für Eich- und Vermessungswesen 2020



Klassifikation von Kufstein (ÖLK: 3728-102) mithilfe eines CNN (RGBI-Modell) 2020

Landcover-Klassen

- Gebäude
- Versiegelte Flächen
- Unversiegelte Flächen
- Gewässer
- Nadelbäume
- Laubbäume
- Sonstiges

Referenzgebiet Kufstein

0 10 20 30 40 m

Maßstab: 1:1.500

Seite 4 von 6

Erstellung: Felix Pekárek

Datengrundlage Orthofoto: Bundesamt für Eich- und Vermessungswesen 2020

Klassifikation von Kufstein (ÖLK: 3728-102) mithilfe eines CNN (RGBI-Modell) 2020

Landcover-Klassen

- Gebäude
- Versiegelte Flächen
- Unversiegelte Flächen
- Gewässer
- Nadelbäume
- Laubbäume
- Sonstiges

Referenzgebiet Kufstein

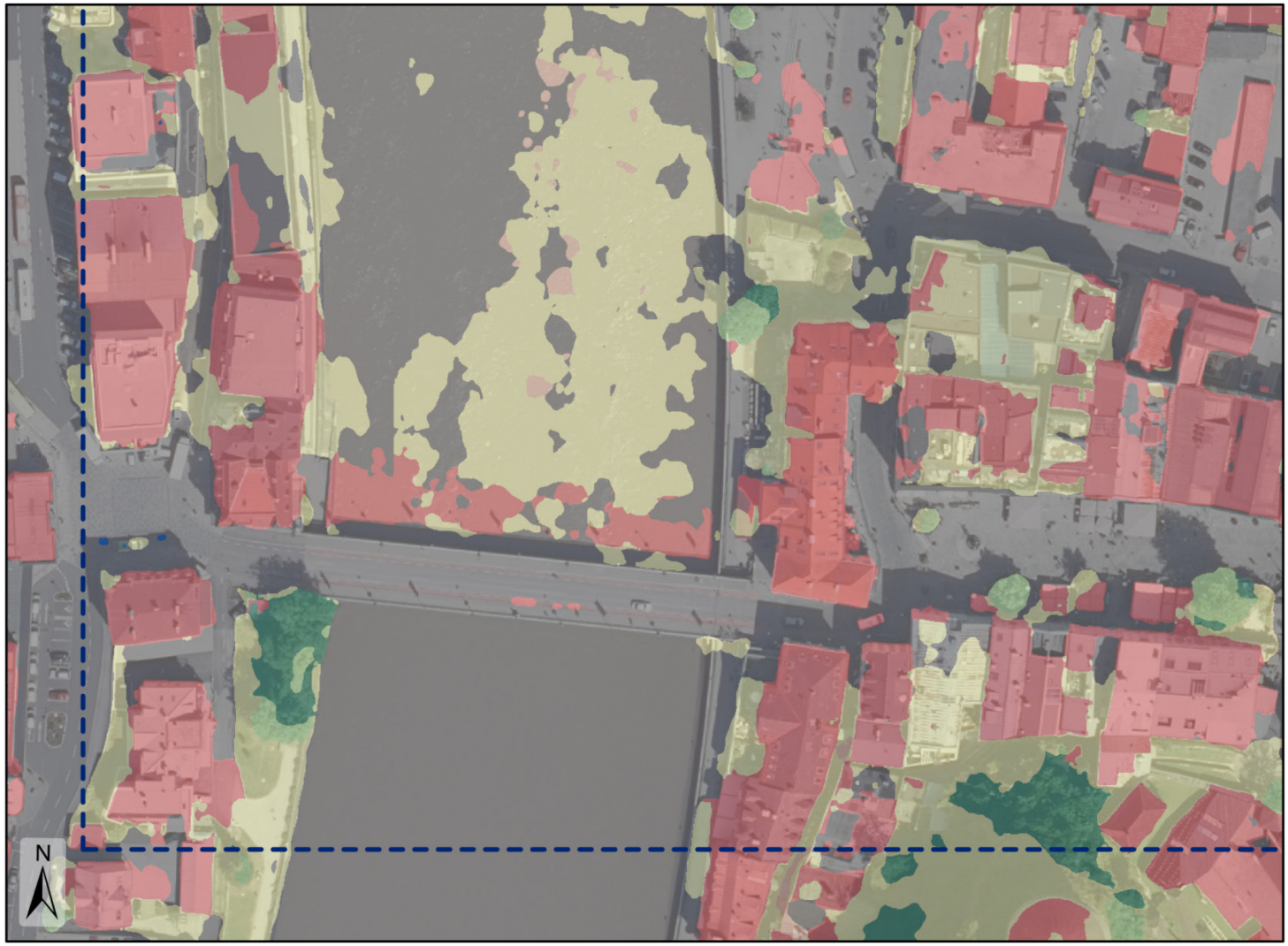
0 10 20 30 40 m

Maßstab: 1:1.500

Seite 5 von 6

Erstellung: Felix Pekárek

Datengrundlage Orthofoto: Bundesamt für Eich- und Vermessungswesen 2020



Klassifikation von Kufstein (ÖLK: 3728-102) mithilfe eines CNN (RGBI-Modell) 2020

Landcover-Klassen

- Gebäude
- Versiegelte Flächen
- Unversiegelte Flächen
- Gewässer
- Nadelbäume
- Laubbäume
- Sonstiges

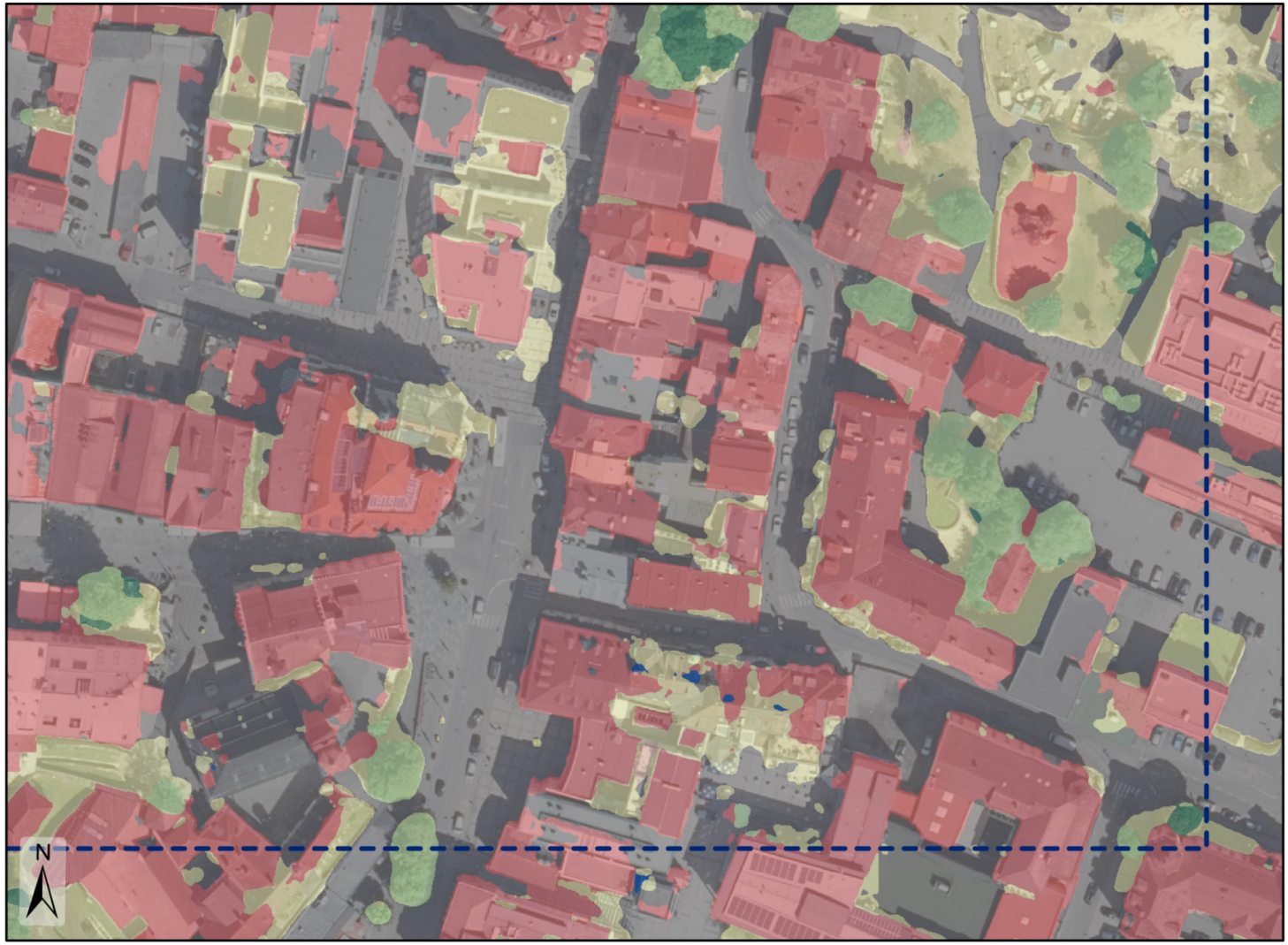
Referenzgebiet Kufstein

m
0 10 20 30 40

Maßstab: 1:1.500
Seite 6 von 6

Erstellung: Felix Pekárek

Datengrundlage Orthofoto: Bundesamt für Eich- und Vermessungswesen 2020



Klassifikation von Kufstein (ÖLK: 3728-102) durch das Bundesamt für Eich- und Vermessungswesen 2020

Landcover-Klassen

- Baum oder Wald
- Bodenflächen
- Buschwerk
- Gebäude
- Gewässer
- Niedrigvegetation

Referenzgebiet Kufstein

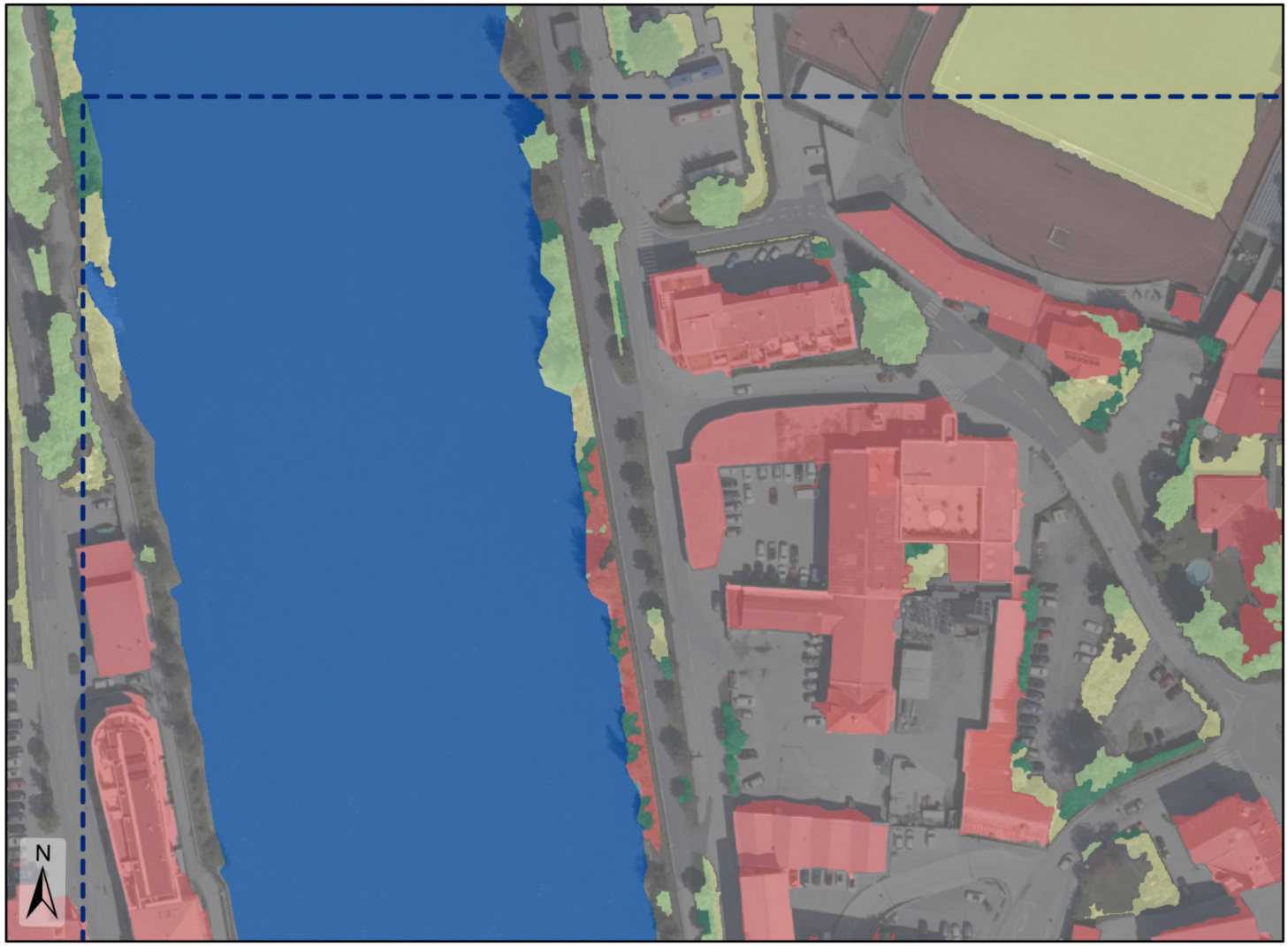
0 10 20 30 40 m

Maßstab: 1:1.500
Seite 1 von 6

Erstellung: Felix Pekárek

Datengrundlage Orthofoto: Bundesamt für Eich- und Vermessungswesen 2020

Datengrundlage Landcover-Klassifikation: Bundesamt für Eich- und Vermessungswesen 2020



Klassifikation von Kufstein (ÖLK: 3728-102) durch das Bundesamt für Eich- und Vermessungswesen 2020

Landcover-Klassen

- Baum oder Wald
- Bodenflächen
- Buschwerk
- Gebäude
- Gewässer
- Niedrigvegetation

Referenzgebiet Kufstein

0 10 20 30 40 m

Maßstab: 1:1.500

Seite 2 von 6

Erstellung: Felix Pekárek

Datengrundlage Orthofoto: Bundesamt für Eich- und Vermessungswesen 2020

Datengrundlage Landcover-Klassifikation: Bundesamt für Eich- und Vermessungswesen 2020

Klassifikation von Kufstein (ÖLK: 3728-102) durch das Bundesamt für Eich- und Vermessungswesen 2020

Landcover-Klassen

- Baum oder Wald
- Bodenflächen
- Buschwerk
- Gebäude
- Gewässer
- Niedrigvegetation

Referenzgebiet Kufstein

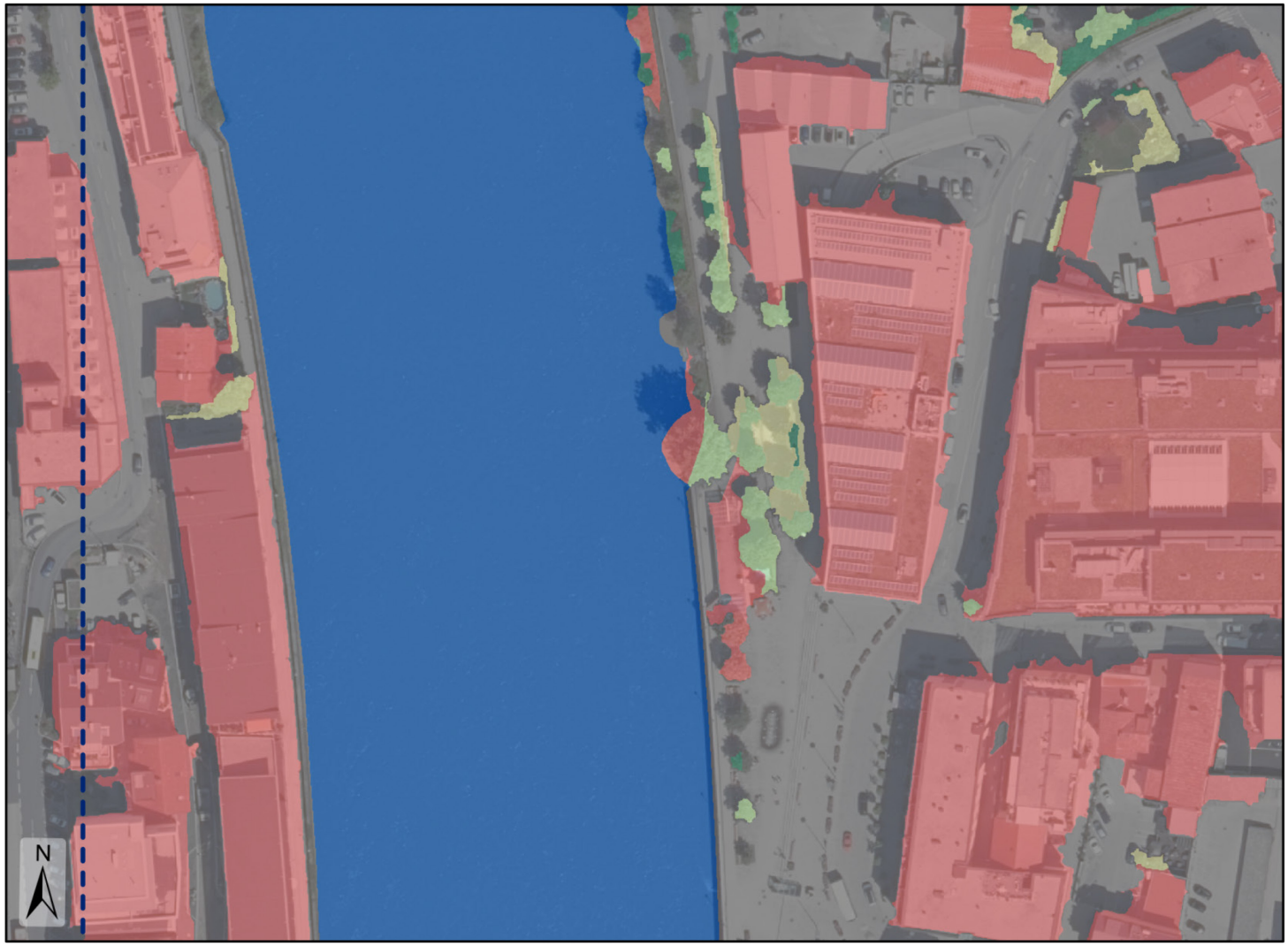
0 10 20 30 40 m

Maßstab: 1:1.500
Seite 3 von 6

Erstellung: Felix Pekárek

Datengrundlage Orthofoto: Bundesamt für Eich- und Vermessungswesen 2020

Datengrundlage Landcover-Klassifikation: Bundesamt für Eich- und Vermessungswesen 2020



Klassifikation von Kufstein (ÖLK: 3728-102) durch das Bundesamt für Eich- und Vermessungswesen 2020

Landcover-Klassen

- Baum oder Wald
- Bodenflächen
- Buschwerk
- Gebäude
- Gewässer
- Niedrigvegetation

Referenzgebiet Kufstein

0 10 20 30 40 m

Maßstab: 1:1.500
Seite 4 von 6

Erstellung: Felix Pekárek

Datengrundlage Orthofoto: Bundesamt für Eich- und Vermessungswesen 2020

Datengrundlage Landcover-Klassifikation: Bundesamt für Eich- und Vermessungswesen 2020

Klassifikation von Kufstein (ÖLK: 3728-102) durch das Bundesamt für Eich- und Vermessungswesen 2020

Landcover-Klassen

- Baum oder Wald
- Bodenflächen
- Buschwerk
- Gebäude
- Gewässer
- Niedrigvegetation

Referenzgebiet Kufstein

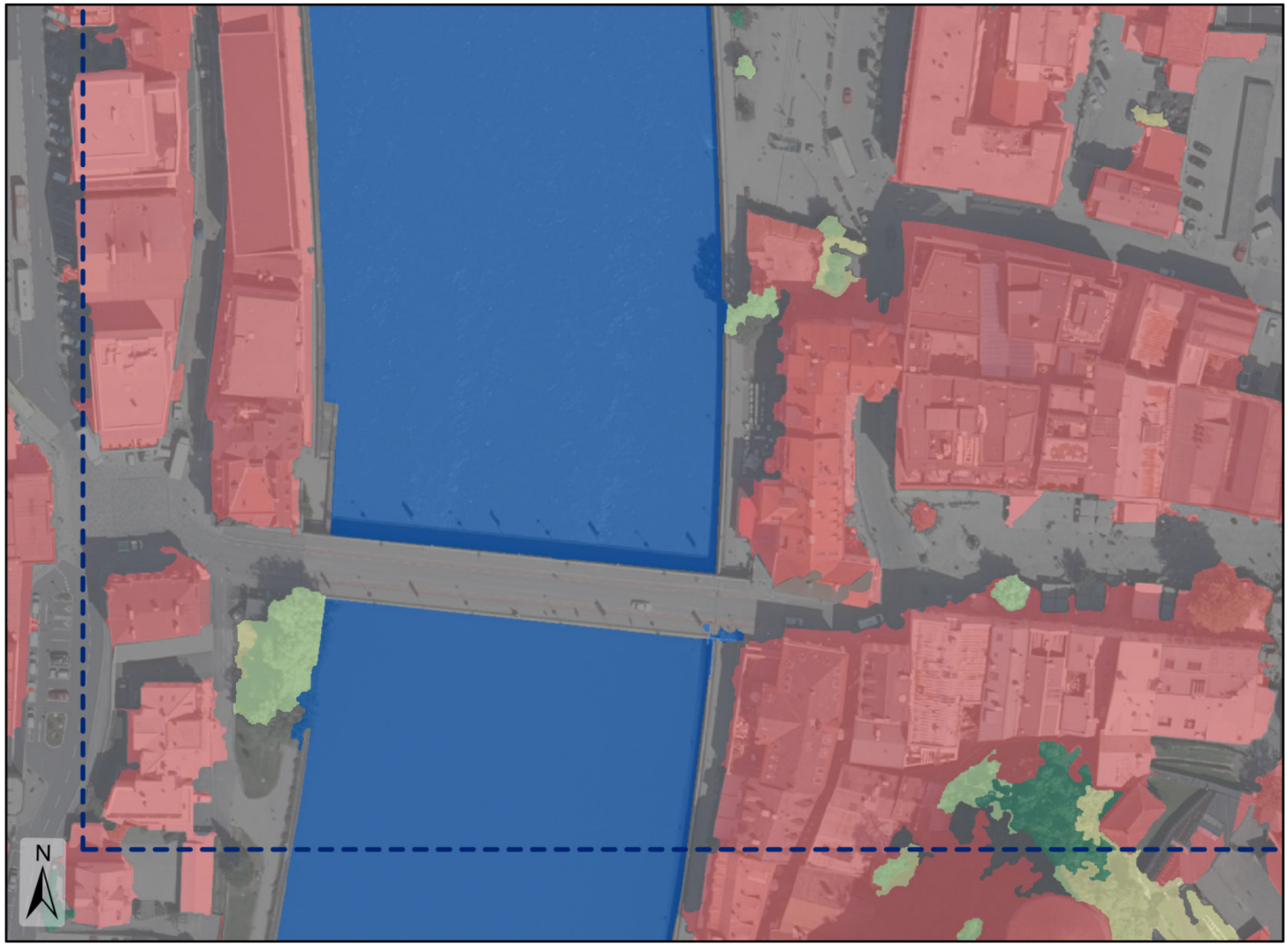
0 10 20 30 40 m

Maßstab: 1:1.500
Seite 5 von 6

Erstellung: Felix Pekárek

Datengrundlage Orthofoto: Bundesamt für Eich- und Vermessungswesen 2020

Datengrundlage Landcover-Klassifikation: Bundesamt für Eich- und Vermessungswesen 2020



Klassifikation von Kufstein (ÖLK: 3728-102) durch das Bundesamt für Eich- und Vermessungswesen 2020

Landcover-Klassen

- Baum oder Wald
- Bodenflächen
- Buschwerk
- Gebäude
- Gewässer
- Niedrigvegetation

Referenzgebiet Kufstein

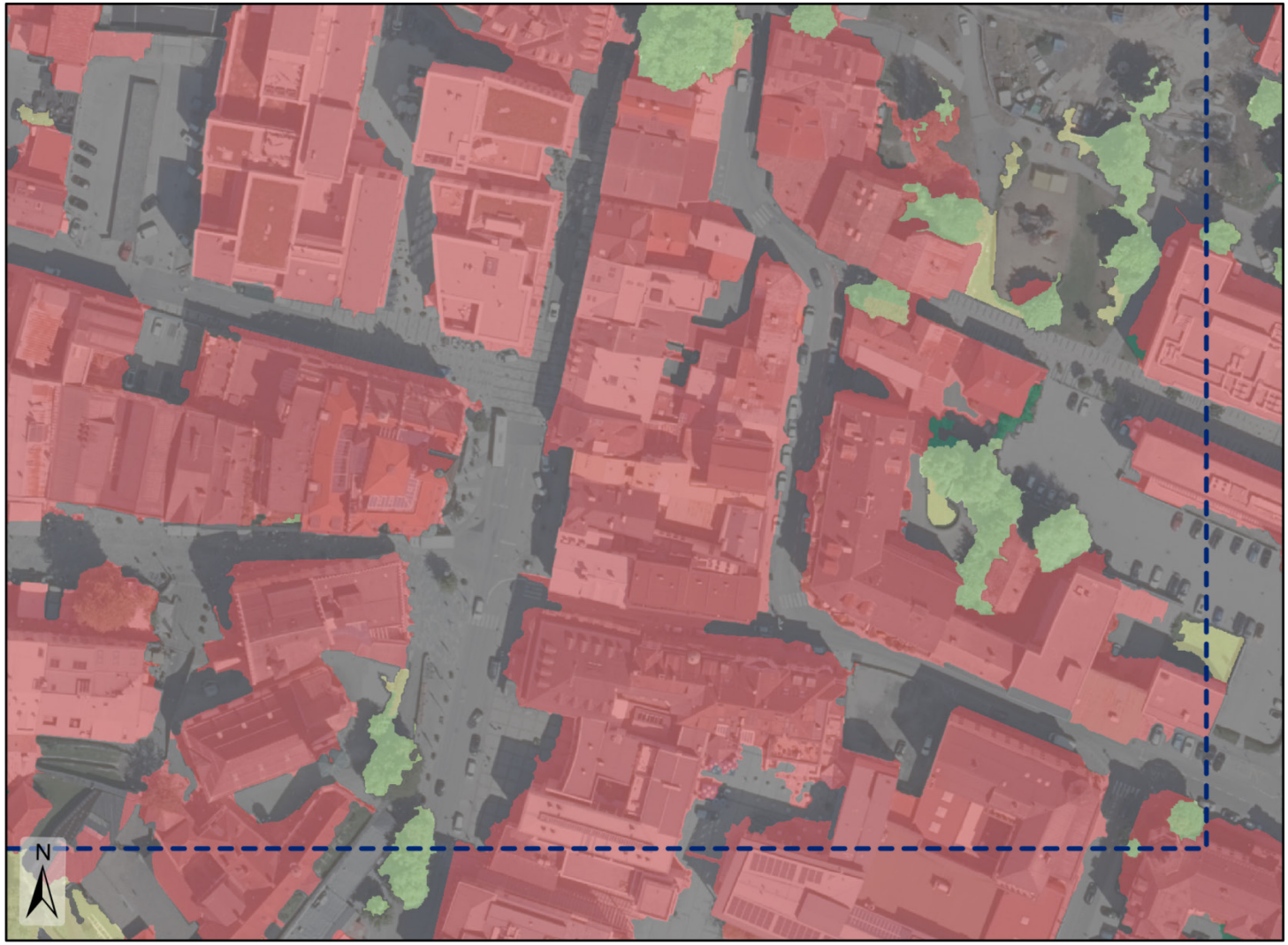
0 10 20 30 40 m

Maßstab: 1:1.500
Seite 6 von 6

Erstellung: Felix Pekárek

Datengrundlage Orthofoto: Bundesamt für Eich- und Vermessungswesen 2020

Datengrundlage Landcover-Klassifikation: Bundesamt für Eich- und Vermessungswesen 2020



Anhang XI: Klassifikationsergebnisse für das Referenzgebiet in Wien

Überblick über den Referenzbereich in Wien (DKM: 7535-68) 2020

Die Karte liefert einen Überblick über den klassifizierten Bereich in Wien und gibt die Seitennummern für die Ausschnitte in den Klassifikationsergebnissen an. In den nachfolgenden Anhängen sind die Klassifikationen durch das RGBI-Modell visualisiert.

0 80 160 240 320 m

Maßstab: 1:8.000

Erstellung: Felix Pekárek

Datengrundlage Orthofoto: Bundesamt für Eich- und Vermessungswesen 2020

Klassifikation von Kufstein (DKM: 7535-68) mithilfe eines CNN (RGBI-Modell) 2020

Landcover-Klassen

- Gebäude
- Versiegelte Flächen
- Unversiegelte Flächen
- Gewässer
- Nadelbäume
- Laubbäume
- Sonstiges

Referenzgebiet Wien

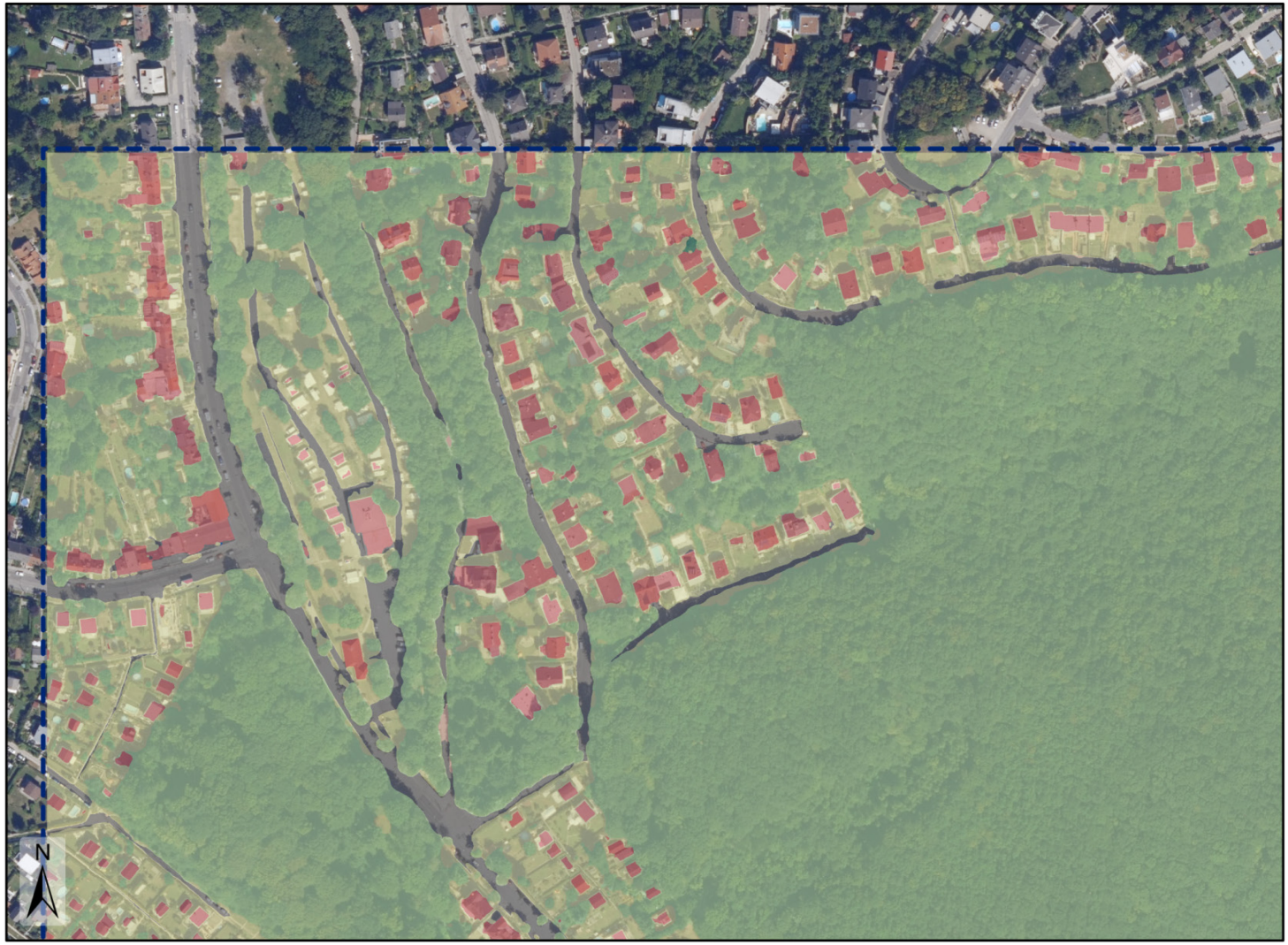
0 30 60 90 120 m

Maßstab: 1:3.500

Seite 1 von 6

Erstellung: Felix Pekárek

Datengrundlage Orthofoto: Bundesamt für Eich- und Vermessungswesen 2020



Klassifikation von Kufstein (DKM: 7535-68) mithilfe eines CNN (RGBI-Modell) 2020

Landcover-Klassen

- Gebäude
- Versiegelte Flächen
- Unversiegelte Flächen
- Gewässer
- Nadelbäume
- Laubbäume
- Sonstiges

Referenzgebiet Wien

0 30 60 90 120 m

Maßstab: 1:3.500

Seite 2 von 6

Erstellung: Felix Pekárek

Datengrundlage Orthofoto: Bundesamt für Eich- und Vermessungswesen 2020

Klassifikation von Kufstein (DKM: 7535-68) mithilfe eines CNN (RGBI-Modell) 2020

Landcover-Klassen

- Gebäude
- Versiegelte Flächen
- Unversiegelte Flächen
- Gewässer
- Nadelbäume
- Laubbäume
- Sonstiges

Referenzgebiet Wien

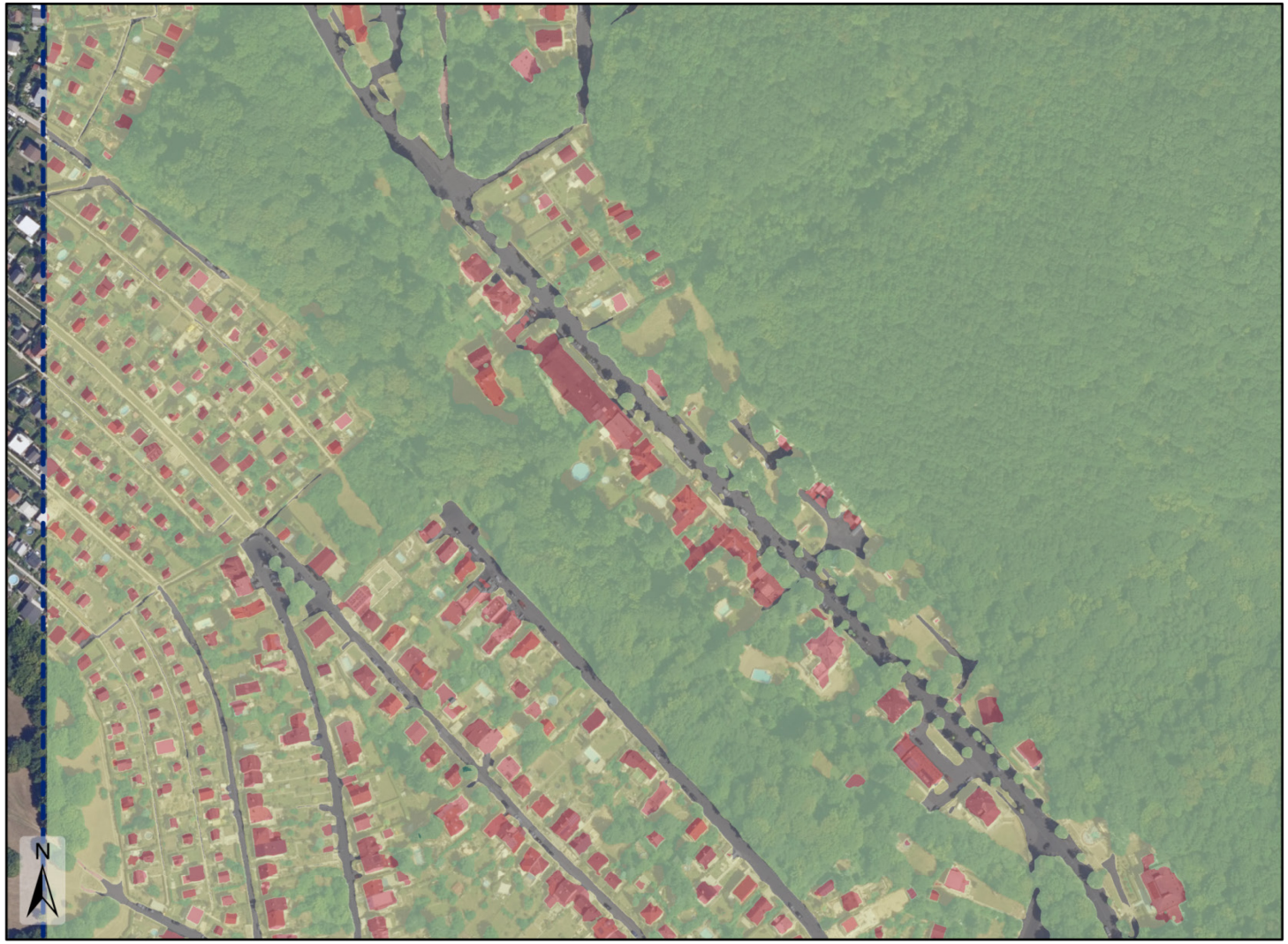
0 30 60 90 120 m

Maßstab: 1:3.500

Seite 3 von 6

Erstellung: Felix Pekárek

Datengrundlage Orthofoto: Bundesamt für Eich- und Vermessungswesen 2020



Klassifikation von Kufstein (DKM: 7535-68) mithilfe eines CNN (RGBI-Modell) 2020

Landcover-Klassen

- Gebäude
- Versiegelte Flächen
- Unversiegelte Flächen
- Gewässer
- Nadelbäume
- Laubbäume
- Sonstiges

Referenzgebiet Wien

0 30 60 90 120 m

Maßstab: 1:3.500
Seite 4 von 6

Erstellung: Felix Pekárek

Datengrundlage Orthofoto: Bundesamt für Eich- und Vermessungswesen 2020

Klassifikation von Kufstein (DKM: 7535-68) mithilfe eines CNN (RGBI-Modell) 2020

Landcover-Klassen

- Gebäude
- Versiegelte Flächen
- Unversiegelte Flächen
- Gewässer
- Nadelbäume
- Laubbäume
- Sonstiges

Referenzgebiet Wien

0 30 60 90 120 m

Maßstab: 1:3.500

Seite 5 von 6

Erstellung: Felix Pekárek

Datengrundlage Orthofoto: Bundesamt für Eich- und Vermessungswesen 2020

Klassifikation von Kufstein (DKM: 7535-68) mithilfe eines CNN (RGBI-Modell) 2020

Landcover-Klassen

- Gebäude
- Versiegelte Flächen
- Unversiegelte Flächen
- Gewässer
- Nadelbäume
- Laubbäume
- Sonstiges

Referenzgebiet Wien

0 30 60 90 120 m

Maßstab: 1:3.500

Seite 6 von 6

Erstellung: Felix Pekárek

Datengrundlage Orthofoto: Bundesamt für Eich- und Vermessungswesen 2020

Hiermit versichere ich,

- dass ich die vorliegende Masterarbeit selbstständig verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und mich auch sonst keiner unerlaubter Hilfe bedient habe,
- dass ich dieses Masterarbeitsthema bisher weder im In- noch im Ausland in irgendeiner Form als Prüfungsarbeit vorgelegt habe
- und dass diese Arbeit mit der vom Begutachter beurteilten Arbeit vollständig übereinstimmt.

Datum Unterschrift