HAW HAMBURG

Bachelorarbeit

Christof Rode

Realisierung einer cloudbasierten Anwendung zur Auswertung von Satelliten- und Luftbildern mittels Deep Learning

Fakultät Technik und Informatik
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

Christof Rode

Realisierung einer cloudbasierten Anwendung zur Auswertung von Satelliten- und Luftbildern mittels Deep Learning

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

Betreuender Prüfer: Prof. Dr. Stephan Schulz
Zweitgutachter: Prof. Dr.-Ing. Andreas Meisel

Eingereicht am: 20. September 2019

Christof Rode

Thema der Arbeit

Realisierung einer cloudbasierten Anwendung zur Auswertung von Satelliten- und Luftbildern mittels Deep Learning

Stichworte

Computer Vision, raumbezogene Datenauswertung, Fernerkundung, Erdbeobachtung, Satellitenbilder, Cloud Computing, Künstliche Neuronale Netze

Kurzzusammenfassung

Hochauflösende Satellitendaten sind heutzutage in großer Zahl verfügbar und bergen enormes Potential. Ohne tiefere Kenntnisse der Computer Vision und Zugang zu erweiterten Hardwareressourcen kann dieses jedoch nicht genutzt werden. Diese Arbeit stellt eine flexible und skalierbare cloud-basierte Lösung vor um Satellitendaten effizient auszuwerten und die Datenbeschaffung zu automatisieren. Diese umfasst auch, eine für den Laien verständliche Webanwendung.

Christof Rode

Title of Thesis

Realisation of a cloud-based application for analysis of satellite and aerial imagery by means of Deep Learning

Keywords

computer vision, geospatial analysis, remote sensing, earth observation, satellite imagery, cloud computing, artificial neural networks

Abstract

Today, high-resolution satellite data are available in large numbers and hold enormous potential. Without a deeper knowledge of computer vision and access to extended hardware resources, it cannot be used. This paper presents a flexible and scalable cloud-based solution to efficiently evaluate satellite data and automate data collection. It also includes a web application that is easy for the layperson to understand.

Inhaltsverzeichnis

1 Motivation

Nach Naturkatastrophen wie Fluten, Erdbeben oder Wirbelstürmen sind Straßen oft beschädigt, der Verkehr blockiert und weitere Folgeschäden treten auf. Sind Kommunikationsnetze betroffen ist die Datenerfassung aus den betroffenen Regionen erschwert. Dabei müssen in solchen Momenten schnell Entscheidungen getroffen werden um Schlimmeres zu verhindern. Auf Basis von Satellitendaten könnte sich schnell ein Überblick der Lage verschafft werden, hj ohne selbst vor Ort sein zu müssen. Für die Dauer der Rettungs- und Aufräumarbeiten müssen die Informationen ständig aktualisiert werden. Eine automatisierte Auswertung der wichtigsten Parameter kann hierbei eine wichtige Unterstützung sein.

Dieser und viele weitere Anwendungsfälle der Erdbeobachtung sind nur möglich, wenn ausreichend Daten erfasst und schnell zur Verfügung gestellt werden können.

Zurzeit befinden sich 5281 Objekte in einem Orbit um den Planeten [31]. Von 2062 funktionsfähigen Satelliten werden 768 zur Erdbeobachtung eingesetzt [6]. Dabei werden immense Mengen an Daten angehäuft. Das europäische Erdbeobachtungsprogramm Copernicus [33] erzeugt jeden Tag 12TB neue Daten. Das Unternehmen DigitalGlobe erweitert täglich sein Datenarchiv um weitere 80TB an hochauflösenden Satellitenbildern [4]. Heute ist es möglich auf diese Daten nur Stunden nach der Aufnahme direkt über eine API des Anbieter zuzugreifen.

In Abbildung 1.1 ist zu erkennen, dass gerade die kommerzielle Raumfahrt für den starken Anstieg der Satellitenstarts der letzten Jahre verantwortlich ist. Viele dieser neu gestarteten Satelliten sind Kleinstsatelliten die Bilder in einer Auflösung aufnehmen können, für die früher ein Vielfaches an Gewicht und Kosten notwendig waren. Dadurch ist es möglich geworden auf einem globalen Maßstab Veränderungen innerhalb kurzer Zeit zur erkennen und zu verfolgen.

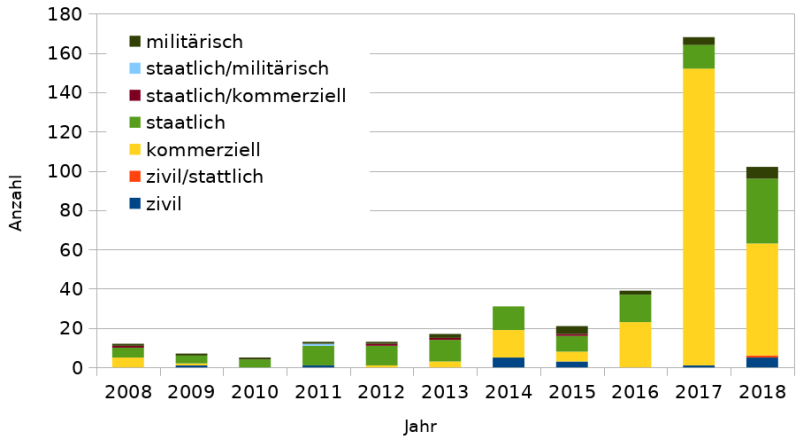


Abbildung 1.1: Anzahl der gestarteten Satelliten die Bilder der Erde anfertigen können[1].

Satellitenbilder ermöglichen eine andere Perspektive auf die Welt und die Datenbestände sind oft global und dadurch sehr divers. Die in den Bildern dargestellten Landschaften sind nicht nur sehr unterschiedlich, sondern auch ständig im Wandel und Objekte müssen oft nur anhand weniger Pixel unterschieden werden können.

Es werden also Verfahren benötigt die diese schwierigen Anforderungen erfüllen. Die Bildverarbeitung hat in den letzten Jahren enorme Fortschritte erzielt und Anwendungsfälle gelöst, die für den Menschen zwar leicht, aber seit Jahrzehnten nicht lösbar schienen. Zum Beispiel können heutzutage alltägliche Objekte oder Sehenswürdigkeiten in Bildern zuverlässig erkannt [2] oder Gesichter vollautomatisch gesucht und identifiziert werden [3]. Technologien wie Deep Learning die diesen Innovationsschub auslösten, werden seitdem in vielen verschiedenen Industrien ausprobiert und erfolgreich angewandt um Probleme der eigenen Domäne zu lösen[4]. So auch im Bereich der satellitengestützten Fernerkundung.

[1] Multispektrale oder Hyperspektrale optische Instrumente im sichtbaren und infrarotem Bereich, basierend auf Daten von [6].
[2] https://cloud.google.com/vision/
[3] https://azure.microsoft.com/en-us/services/cognitive-services/face/
[4] Beispielsweise im Bereich der Magnetresonanztomographie [25] oder der Maschinenüberwachung [51])

2 Ziel der Arbeit

Es soll ein Dienst entwickelt werden der Daten aus verschiedenen Quellen bezieht, Möglichkeiten zur automatisieren Analyse anbietet und über branchenübliche Schnittstellen zur Verfügung stellt.

Die Schnittstellen der Anbieter sind nicht einheitlich und der Umgang mit den APIs erfordert immer einen Blick in die Dokumentation. Die Daten selbst werden als unterschiedliche Produkte angeboten und unterscheiden sich im Format, den zusätzlichen Metainformationen und in qualitativen Eigenschaften wie der räumlichen Auflösung. Damit sich der Anwender nicht mit diesen Details auseinandersetzen muss, soll das System abhängig vom Ziel der Analyse, nur geeignete Datenquellen vorschlagen und die Beschaffung der Daten übernehmen.

Antworten auf Fragen wie *"Wie viele Windräder stehen in Luxemburg?"* oder *"Welche Gebäude sind nach dem Hurricane beschädigt?"* lassen sich ohne tiefere Erkenntnisse der Computer Vision nicht lösen. Das System soll es auch Laien ermöglichen Antworten auf Fragen wie diese zu bekommen.

Die Verarbeitung auf dem Desktop mit einem GIS-Programm ist bei größeren Datenmengen nicht mehr praktikabel und die computergestützte Analyse benötigt entsprechende Rechenkapazitäten. Daher soll das System webbasiert sein und die Ressourcen einer Cloud-Umgebung nutzen.

Die beschriebene Funktionalität soll durch eine API und eine Webanwendung zur Verfügung gestellt werden.

3 Inhaltliche Einschränkungen

Die Erdbeobachtung bietet einige grundsätzliche Möglichkeiten Daten zu erfassen. Die Mehrzahl der Satelliten beinhaltet optische Instrumente. Diese arbeiten im sichtbaren und nahen Infrarotbereich des elektromagnetischen Spektrums und bilden die Grundlage für die Satellitensichten von Kartendiensten wie Google Maps oder Apple Maps.

Diese Systeme werden als passiv bezeichnet, da sie keine eigene Energiequelle nutzen sondern nur die Reflexion von Sonnenstrahlen erfassen. Instrumente die im thermischen (mittel- bis langwelligem) Infrarotbereich arbeiten, sind ein weiter Vertreter dieser Kategorie und werden häufig im Bereich der Meteorologie und Klimatologie eingesetzt.

Weniger Satelliten sind aktive Systeme wie SAR, das Radarwellen nutzt oder LIDAR (Laserpulse). Diese Systeme senden selbst Energie aus und funktionieren daher auch bei Nacht wenn passive Systeme im sichtbaren Bereich keine gewünschten Daten liefern können[1]. SAR kann Frequenzen nutzen, die nicht vom Wetter beeinflusst werden. Dies ist ein großer Vorteil, da im Schnitt 68 Prozent der Erde von Wolken bedeckt ist [43]. LIDAR ermöglicht sehr genaue Höhenmessungen. Das LIDAR Instrument von ICESat-2 kann aus einer Flughöhe von etwa $500\,\mathrm{km}$ die Höhe der polaren Eisdecke im Zentimeter-Bereich messen [26].

Um den Umfang für diese Arbeit einzuschränken sollen jedoch nur zweidimensionale Daten von optischen Systemen in Betracht gezogen werden (im Folgenden als Daten bezeichnet). Wie zu Beginn in 1 bereits erwähnt wurde, werden diese Daten von einigen Anbietern im öffentlichen und privaten Sektor angeboten. Außerdem können die Daten mit Mitteln der digitalen Bildverarbeitung ausgewertet und visualisiert werden.

Das zu entwickelnde System soll abgeschlossen nutzbar sein: Es sollen alle notwendigen Schnittstellen implementiert werden, die es dem Anwender ermöglichen, das System sinnvoll für seinen Anwendungsfall zu verwenden. Dazu müssen die möglichen Anwender bestimmt und deren Anforderungen erfasst werden. Allerdings soll

[1] Außer man ist *gerade* an Nachtaufnahmen interessiert, siehe z.B. Suomi NPP VIIRS Day/Night Band https://gcmd.gsfc.nasa.gov/search/Metadata.do?entry=VNP02DNB_NRT_1&subset=gcmd

keine Produktreife erreicht werden: Wichtige Themen wie Robustheit, Geschäftsmodell, Dokumentation, Usability oder Security spielen nur eine untergeordnete Rolle.

Es sollen mehrere möglichst unterschiedliche Datenquellen und mindestens ein komplexer Anwendungsfall einer Objekterkennung ausgewählt und implementiert werden um damit den sinnvollen Einsatz von Deep Learning Techniken zu zeigen.

4 Grundlagen zu Satellitendaten

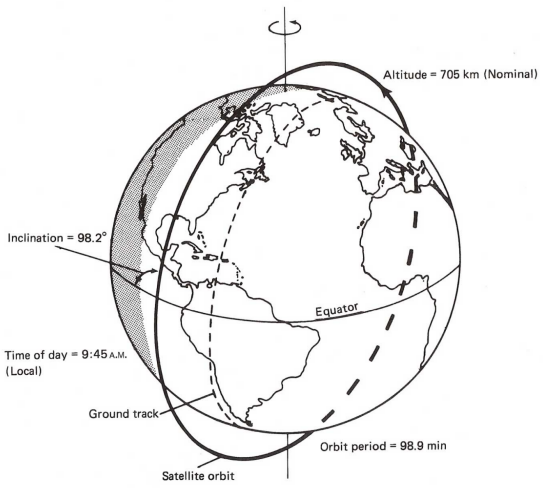
Das folgende Kapitel stellt einige wichtige Grundlagen von Satellitendaten vor, die essentiell für das Verständnis der restlichen Arbeit sind. Am Ende des Kapitels werden Vergleichskategorien eingeführt, die im darauffolgenden Teil zur Unterscheidung und Auswahl der Datenquellen des Systems genutzt werden.

4.1 Orbits

Alle für diese Arbeit relevanten Satelliten befinden sich auf einer sehr kreisförmigen erdnahen Umlaufbahn in einigen hundert Kilometern Höhe. Dabei bewegen sie sich in sonnensynchronen fast-polaren Orbits.

Eine polare Umlaufbahn bedeutet, dass der Satellit sich bei jedem Umlauf über Nord- und Südpol bewegt und damit alle Breitengrade überquert. Durch die Rotation der Erde kann somit theoretisch die gesamte Erdoberfläche beobachtet werden.

Ein sonnensynchroner Orbit ist eine geschickte Kombination von Höhe und Neigung und bewirkt, dass der Satellit einen beliebigen Punkt auf der Erde immer zur selben Ortszeit überquert. Die Position der Sonne in Relation zum Satellit und der Erde bleibt dabei gleich. Dies ist nützlich beim Vergleich mehrerer Aufnahmen zur Erkennung von Änderungen, da dadurch äußere Parameter, bestimmt durch den Sonnenstand, wie Schattenwurf oder Helligkeit zwischen zwei Aufnahmen gleich bleiben. Für die Erde sind diese Orbits aufgrund der Erdrotation, für den erdnahen Bereich fast-polar (Neigung zwischen $98\,\mathrm{und}\,99°$). In Abbildung 4.1 ist ein Orbit von Landsat Satelliten dargestellt. Landsat ist ein seit Jahrzehnten bestehendes Erdbeobachtungs-Programm der USA.

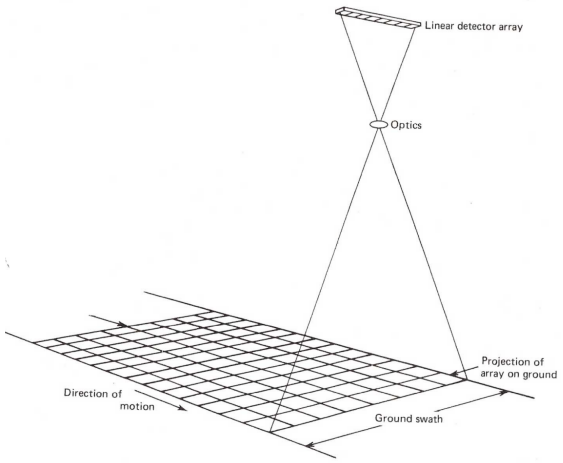


Entnommen aus: [23, p. 289]
Abbildung 4.1: Sonnensynchroner Orbit von Landsat-4, -5, -7 und -8. Dies ist ein typischer Orbit eines Erdbeobachtungssatelliten im erdnahen Umfeld mit einer Umlaufzeit von ca. $1{,}5\,\mathrm{h}$.

4.2 Räumliche Auflösung

Viele verbaute optische Systeme arbeiten nach dem Prinzip einer Zeilenkamera, wie sie beispielsweise auch in Flachbettscannern Anwendung finden. Dabei ist diese senkrecht zur Flugrichtung des Satelliten angebracht. Es gibt aber auch Instrumente die zweidimensional aufnehmen können.

Die Abbildung 4.2 veranschaulicht ein solches System. Im unteren Bereich ist zudem der sogenannte Schwad des Satelliten eingezeichnet. Dieser bezeichnet den durch das Instrument aufgenommenen Streifen der Oberfläche. Die Schwadbreite wird senkrecht zur Flugbahn gemessen. Die Länge ist technisch dadurch begrenzt, wie viele Daten zwischengespeichert werden können, bevor sie an die nächsten Bodenstation übermittelt werden.



Entnommen aus: [23, p. 225]

Abbildung 4.2: Illustration eines Zeilenkamera basierten Sensors. Jedes im unteren Bereich eingezeichnete Rechteck entspricht der Fläche eines Pixels der Zeilenkamera, die im oberen Bereich als Pixel Array abgebildet ist.

Field of View, Flughöhe und Anzahl der Pixel auf dem CCD ergeben die räumliche Auflösung am Boden (Größe eines Pixels). Diese ist jedoch nicht konstant sondern, wie in Abbildung 4.3 veranschaulicht, nimmt diese mit zunehmenden Winkel des Field of Views ab. Eine übliche Angabe ist die ground sample distance (GSD), also der Abstand zweier benachbarter Pixel, gemessen am Boden. Konzeptuell ist diese Angabe vergleichbar mit der Angabe der dots per inch (DPI) bei einem Drucker.

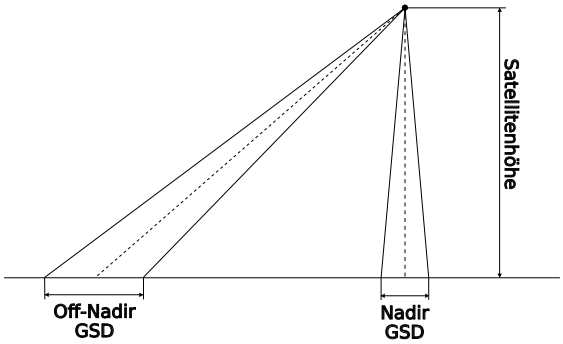


Abbildung 4.3: Räumliche Auflösung in Abhängigkeit der Flughöhe und Blickwinkel. Die beste Auflösung erhält man direkt unter dem Sensor.

4.3 Multispektrale Sensoren

Nimmt das verbaute Instrument verschiedene Bereiche des elektromagnetischen Spektrums separat auf, handelt es ich um multispektrale Daten. Neben den drei für das menschliche Auge wahrnehmbaren Bereichen (VIS) für rotes, grünes und blaues Licht gibt es meist noch weitere schmale und breite Bänder aus dem nahen (NIR) und kurzwelligen Infrarotbereich (SWIR). Von multispektral spricht man bereits ab drei Bändern (RGB), üblich sind weniger als 15. Hyperspektrale Daten beinhalten hunderte sehr schmale Bänder.

Der Grund warum möglichst viele schmale Spektren separat aufgenommen werden sollen, hat mehrere Gründe. Der naheliegendste ist zum einen zur Visualisierung, also um ein Farbbild anzufertigen. Zum anderen ist die Auflösung i.d.R. zu schlecht um kleinere Details im Bild auszumachen oder durch die Perspektive von oben nur eingeschränkt einsehbar. Um trotzdem Rückschlüsse auf das beobachtete Objekt ziehen zu können, nutzt man die Eigenschaft dass verschiedene Materialien in Abhängigkeit der Wellenlänge unterschiedlich stark reflektieren [1].

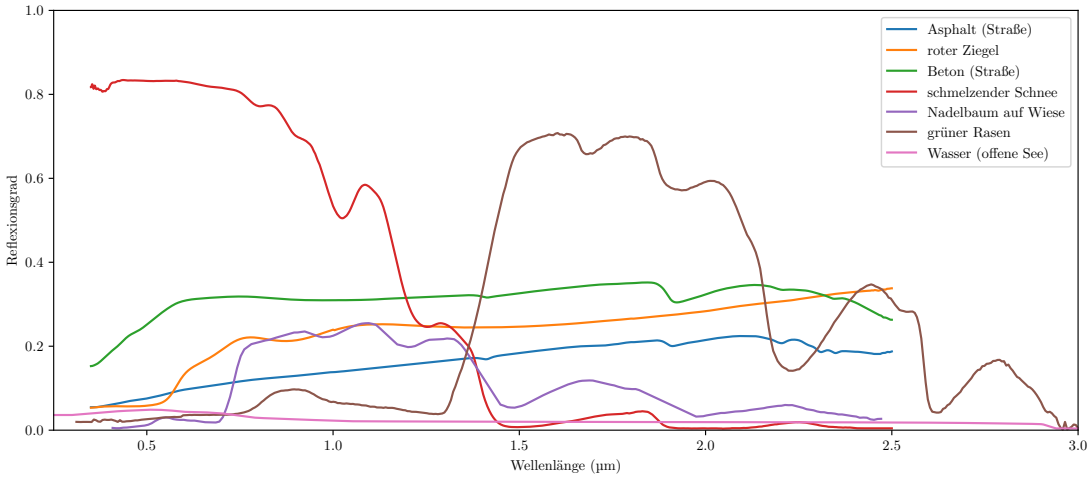


Abbildung 4.4: Spektrale Signaturen verschiedener Materialien in den VIS, NIR und SWIR Bereichen des Spektrums. Dabei zeigen Schnee und Wasser im Bereich bis $1{,}5\,\mu m$ völlig verschiedenes Reflexionsverhalten. Basierend auf Daten verschiedener Proben der USGS Spectral Library Version 7[2].

[1] Je Nach Anwendung sind diese Informationen aber auch bei hochaufgelösten RGB Bändern sehr hilfreich um weitere Unterscheidungsmerkmale nutzen zu können

[2] https://doi.org/10.3133/ds1035

Die Variation der Reflexionseigenschaften werden als spektrale Signaturen bezeichnet und können verwendet werden um ein Material[3] zu identifizieren.

In Grafik 4.4 sind für die Linie *grüner Rasen* drei größere Ausschläge im nahen Infrarotbereich auszumachen. Die hohe Reflexion in diesem Bereich ist charakteristisch bei gesunder grüner Vegetation und ist auf das in Pflanzen enthaltene Chlorophyll zurückzuführen. Bei Kenntnis der genauen Form der Kurve lässt sich auch eine einzelne Gattung der Pflanze klassifizieren. Beispielsweise wäre eine künstliche grüne Farbe, die für das menschliche Auge im Vergleich zu einer bestimmten Vegetation identisch wirkt, über ihre spektrale Signatur unterscheidbar. Die Grafik zeigt weiterhin, dass viele markante Eigenschaften einer spektralen Signatur außerhalb des sichtbaren Bereichs ($0{,}38\,\mathrm{b}$is $0{,}75\,\mu\mathrm{m}$) liegen. Daher gibt es viele Satelliteninstrumente die in diesem Bereich empfindlich sind.

Bei schmalen Bändern fällt insgesamt weniger Licht ein, da nur Wellenlängen detektiert werden, die innerhalb des Bandes liegen. Die Belichtungszeit muss daher erhöht werden, was jedoch die räumliche Auflösung reduziert, da sich der Satellit dabei in Bewegung befindet. Aus diesem Grund gibt es bei vielen multispektralen Satelliten zusätzlich ein panchromatisches Band das im gesamten sichtbaren Spektrum empfindlich ist und somit ein Schwarz/Weiß Photo darstellt. Dieses kann kürzer belichtet werden und liegt in einer vergleichsweise hohen Auflösung vor.

In der Regel bieten die Anbieter der Daten auch Produkte an, die die hohe räumliche Auflösung des panchromatischen Bandes mit der hohen spektralen Auflösung (viele schmale Bänder) der multispektralen Bänder in einem, als Pansharpening bezeichneten Verfahren kombinieren.

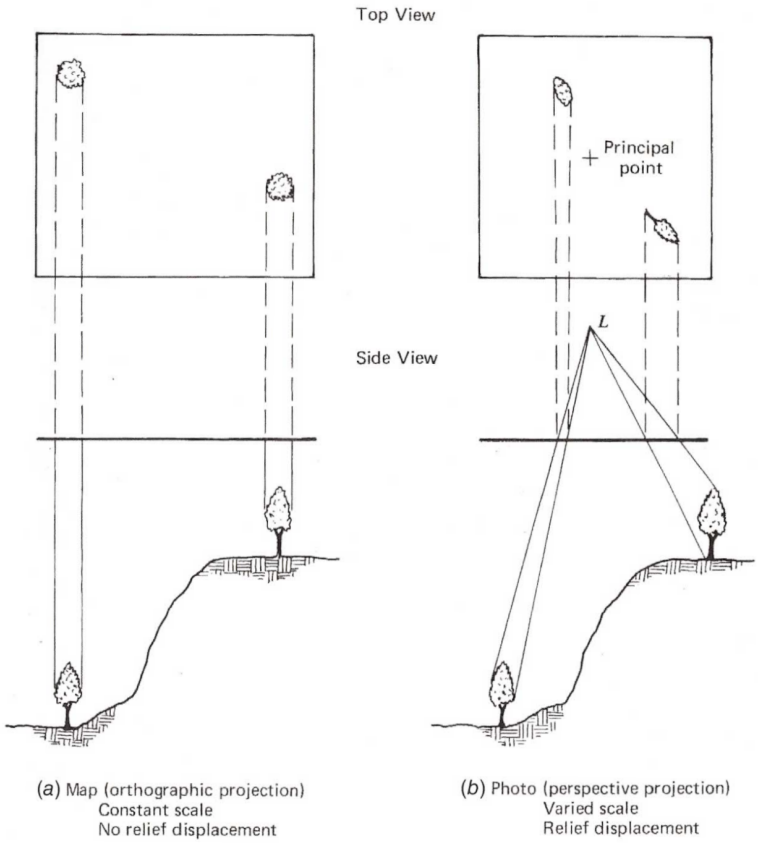
4.4 Fehlerquellen: Verzerrungen und Verschiebungen

Die Abbildung der Landschaft in den Daten enthält Fehler durch die Perspektive der Sensoroptik, die Bewegung der Scanneroptik, Geländerelief und Erdkrümmung [5, p.21].

Ein Orthophoto gleicht diese Unterschiede aus, sodass ein einheitlicher Maßstab entsteht. Dazu ist Kenntnis über Position und Neigung des Satellits bzw. des Instruments und ein Höhenmodell der Erdoberfläche[4] notwendig.

[3] Bei Satellitendaten eher eine Kombination von Materialien

[4] Wird auch als DEM (digital elevation model) bezeichnet und liegt in einem Bildformat vor, indem jeder Pixelwert die Höhe des Bodens über dem Meeresspiegel angibt.

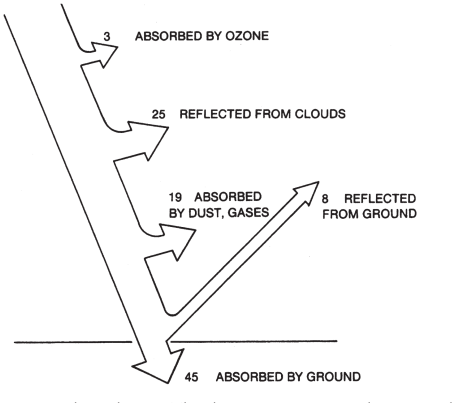


Entnommen aus: [23, p. 163]
Abbildung 4.5: Vergleich von orthographischer und perspektivischer Projektion. Erkennbar ist der Unterschied in Größe, Form und Ort der Bäume.

Eine weitere Störquelle ist die Atmosphäre, die eingehende und von der Erdoberfläche reflektierte Sonnenstrahlen beeinflusst und die Aufnahme verfälscht.

Trifft Sonnenstrahlung auf Partikel in der Atmosphäre kommt es zu verschiedenen Arten von Absorption und Reflexion. In Abhängigkeit von Wellenlänge und Größe des interagierenden Objekts treten unterschiedliche Streumuster auf. Manche Moleküle wie Ozon oder Wasserdampf sind starke Absorber und sind für bestimmte Wellenlängen undurchdringlich. Ungefähre Größenordnungen für die Verhältnisse von Absorption und Reflexion sind in Abbildung 4.6 dargestellt.

Die Zusammensetzung der Atmosphäre ändert sich durch Wetter- und Klimaänderungen laufend und muss für entsprechende Korrekturen bestimmt werden. Bereiche von Wellenlängen in denen diese Effekte nur minimal auftreten, nennt man atmosphärische Fenster. Diese sind ausschlaggebend für die Platzierung der Bänder eines multispektralen Sensors (siehe Abbildung 4.7).



Entnommen aus: [5, p. 46]

Abbildung 4.6: Darstellung von eintretender Sonnenstrahlung im und nahe des sichtbaren Spektrums. Die Werte repräsentieren ungefähre Größenordnungen für die Erde als Ganzes.

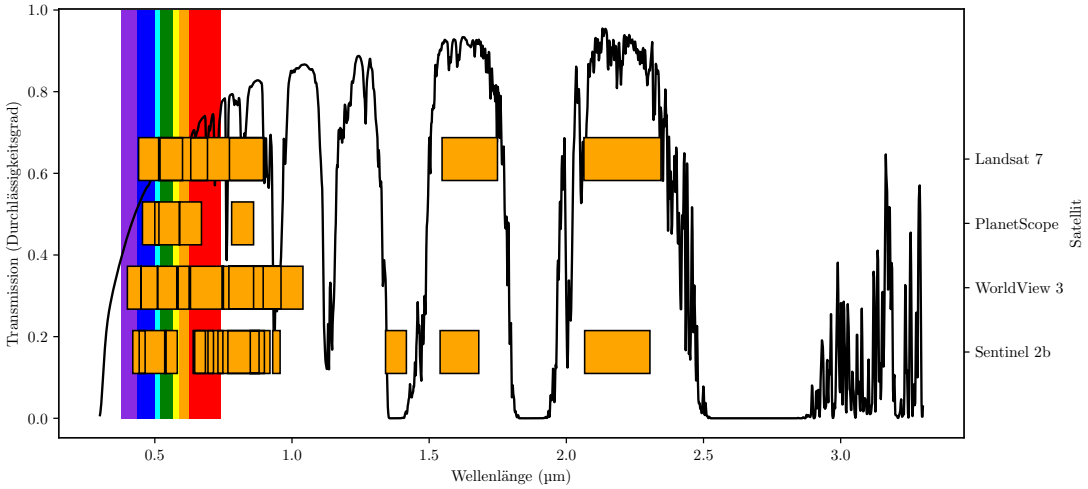


Abbildung 4.7: Darstellung der Durchlässigkeit der Atmosphäre in Abhängigkeit der Wellenlänge. Position und Länge der multispektralen Bänder von vier Satelliten sind als Balken eingezeichnet. Erkennbar sind einige Bereiche die bestimmtes Licht vollständig absorbieren und die atmosphärischen Fenster. Basierend auf Produktinformationen der Anbieter und Transmissionswerte der Software MODTRAN.

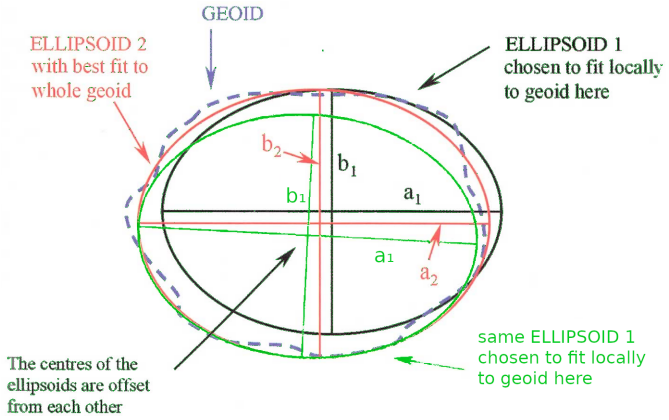
4.5 Georeferenzierung

Um sinnvoll mit den Bilddaten arbeiten zu können muss es eine Zuordnung von einem Pixel zu seinem geographischen Ort als Koordinate geben. Ein Koordinatenreferenzsystem (CRS) ist ein Koordinatensystem, das durch die Beschreibung eines Datums auf ein Objekt bezogen wurde. Ein Datum positioniert (durch Definition eines Ursprungs) und orientiert ein Koordinatensystem und macht es somit zu einem CRS. Das Objekt wäre für ein globales Koordinatensystem die Erde, allerdings ist die Form der Erde durch die sich verändernde Topografie (Erdbeben, tektonische Plattenverschiebungen u.a.) nicht trivial anzugeben.

Als Modell für die Form der Erde wird der Geoid verwendet und ist definiert durch die Form einer Oberfläche, die überall senkrecht zur Richtung der Schwerkraft ist (Äquipotentialfläche). Eine gute Approximation dieser Fläche ist der mittlere Meeresspiegel (verfälscht durch permanente Meeresströmungen die eine Neigung bezüglich der Richtung der Schwerkraft formen).

Der Geoid eignet sich durch seine Form auch nicht direkt um darauf ein Koordinatensystem zu definieren, dafür wird meist ein Ellipsoid (gestauchte oder gestreckte Kugel) verwendet. Für manche Anwendungsfälle wird auch eine Kugel als Modell verwendet. Diese ist zwar ungenauer, allerdings sind die mathematischen Operationen der Kugelgeometrie weniger rechenaufwendig und für manche Anwendungsfälle ausreichend.

Die Positionierung und Orientierung des Ellipsoiden am Geoid ist abhängig vom Anwendungszweck. Die Abbildung 4.8 zeigt den Zusammenhang an einem Beispiel von einem globalen und mehreren lokal besser passenden Modellen.



Entnommen aus: [14, p. 23]

Abbildung 4.8: Der Geoid mit mehreren Ellipsoiden die unterschiedlich positioniert und orientiert sind.

GPS verwendet mit dem CRS84 eine Orientierung, die für die gesamte Erde gut passt. Wird kein globales Koordiantensystem benötigt, gibt es besser passende Alternativen. Für Europa zum Beispiel ETRS89.

Für die Referenzierung der Bilddaten muss eine Karten Projektion durchgeführt werden, die das dreidimensionale CRS mit Längen- und Breitengraden, auf das zweidimensionale kartesische Koordiantensystem des Bildes abbildet. Diese Transformation ist immer mit Fehlern behaftet und es gibt viele verschiedene Verfahren die manche Eigenschaften wie Form oder Skalierung priorisieren, dafür aber andere wie die Richtung verzerren.

Die Überführung eines Bildes in ein CRS, durch eine Projektion wird als Georektifizierung bezeichnet und wird vom Satellitenbetreiber durchgeführt. Georeferenzierung meint allgemein den Vorgang der Verknüpfung von einer Position im Bild zu einer räumlichen Koordinate. Ein Beispiel wäre in einem GIS-Programm die Anzeige einer Koordiante in Latitude und Longitude wenn man die Maus an eine beliebige Stelle im Bild bewegt. Dafür muss das Bild georektifiziert sein, also Informationen über das verwendete CRS vorliegen.

Die Satellitenbetreiber versuchen eine möglichst genaue Positionsbestimmung zu erreichen, allerdings sind durch die oben beschriebenen Störeinflüsse Ausreißer nicht zu vermeiden. Aus diesem Grund meist ist ein statistisches Maß wie der circular error (CE) angegeben. Beispiel: Ein CE90 von $10\,\mathrm{m}$ wäre eine Genauigkeit in der 90% der gemessenen Bildpunkte statistisch innerhalb von $10\,\mathrm{m}$ von den realen Koordinaten am Boden liegen.

Wird eine höhere Genauigkeit benötigt als vom Satellitenbetreiber garantiert, kann durch die Verwendung von mehreren Referenzpunkten bzw ground control points (eins-zu-eins Mapping einer 3D Koordinate im CRS zur 2D Koordinate im Bild), die vor Ort durch einen GNSS[5] Empfänger ermittelt wurden, eine nachträgliche Transformation durchgeführt werden.

[5] global navigation satellite system (GNSS)

4.6 Angebot

Die beschriebenen Störquellen erfordern eine Aufbereitung der Rohdaten in mehreren Stufen. Die NASA definiert für Daten aus ihren Missionen vier Data Processing Levels [6] um die, bei den Verarbeitungsschritten entstandenen Produkte, einzuordnen. Dabei entspricht Level 0[7] den Rohdaten in voller Auflösung. Daten aus höheren Leveln sind in nützlichere Parameter und Formate konvertiert. Beinhalten also Georeferenzierung, geometrische und atmosphärische Korrekturen, Kalibrierungsinformationen und weitere abgeleitete Daten. Andere Anbieter orientieren sich lose an diesen Leveln oder verwenden eigene Bezeichnungen.

Üblicherweise werden mehrere Produkte aus den Aufnahmen abgeleitet, die alle georektifiziert sind und in einer konstanten Ground Sample Distance vorliegen. Dabei gibt es in der Regel mindestens die Auswahl zwischen zwei Produkten. Zum einen eine Top of Atmosphere (TOA) Variante, in der die gemessenen Werte nicht durch atmosphärische Korrekturen verändert wurden. Daneben gibt es eine Bottom of Atmosphere (BOA) Variante die diese Korrekturen enthält. Dieses Produkt wird bei manchen Anbietern auch *visual* genannt, da bei Betrachtung der RGB-Bänder ein korrektes Farbbild vorliegt. Beide Varianten werden meist in einer orthogonalen Projektion basierend auf nadir-Aufnahmen angeboten.

Ein Archivzugriff ermöglicht den Zugang zu bereits erstellten Aufnahmen. Durch Tasking kann ein Kunde zukünftige Aufnahmen für ein bestimmtes Gebiet anzufordern. Dabei werden Anfragen verschiedener Kunden in einem dynamischen Aufnahmeplan erfasst, der mit preislichem Aufschlag auch priorisiert werden kann. Befindet sich der Satellit nicht direkt über dem Gebiet und können perspektivische Verzerrungen in Kauf genommen werden, kann das Instrument auch dorthin ausgerichtet werden. Durch Tasking werden so einige Gebiete öfter aufgenommen als Gebiete, die nicht so häufig nachgefragt werden.

[6] NASA Data Processing Levels https://earthdata.nasa.gov/collaborate/open-data-services-and-software/data-information-policy/data-levels

[7] Level wird i.d.R. nicht ausgeschrieben. Üblich sind Angaben wie L1C oder L2

4.7 Vergleichskategorien

Um die Produkte der Datenanbieter besser vergleichen zu können ist eine Einteilung in die folgenden Kategorien hilfreich. Diese Einteilung wird im nächsten Kapitel verwendet um die verwendeten Datenquellen zu beschreiben.

Spektrale Auflösung Wie viele Wellenlängen lassen sich unterscheiden? Anzahl, Position und Länge der Bänder im Spektrum.

Radiometrische Auflösung Wie genau kann die Stärke des Signals angegeben werden? Anzahl verschiedener Werte die ein Pixel annehmen kann (bit depth).

Räumliche Auflösung Wie viele Details am Boden lassen sich erkennen? Größe eines Pixels gemessen am Boden.

Zeitliche Auflösung Wie viel Zeit vergeht bis der Satellit die gleiche Stelle nadir [8] erneut beobachtet? Üblich ist hier die Angabe der revisit time.

Räumliche Abdeckung Welche Fläche der Erde umfassen die Daten?

Zeitliche Abdeckung Seit wann stehen Aufnahmen zur Verfügung? [9]

[8] Nadir beschreibt die Richtung die direkt unter den Satelliten zeigt.

[9] Das US-amerikanische Landsat Programm stellt seit den 70ger Jahren Fernerkundungsdaten zur Verfügung (bis heute!). Empfehlung: Timelapse der letzten 35 Jahre https://earthengine.google.com/timelapse/

5 Auswahl der Datenquellen

Zur Demonstration des Systems sollten möglichst unterschiedliche Datenanbieter und Datenquellen gewählt und implementiert werden. Dabei viel die Wahl auf *PlanetScope*, *Sentinel-2* und *DOP20 Hamburg*. Im folgenden wird zur Einordnung zunächst ein kurzer Überblick gegeben welche Daten allgemein angeboten werden. Anschließend wird erläutert, warum sich für diese drei Datenquellen entschieden wurde. Detailinformationen der Daten können der darauffolgenden Tabelle entnommen werden.

5.1 Verfügbare Daten

Mit 30cm Auflösung des panchromatischen Bandes ist World-View-3 ein Satellit mit einer der höchsten Auflösungen die zivil bzw. kommerziell verfügbar sind.

Die PlanetScope Satelliten-Konstellation liefert mit einer mindestens täglichen revisit time, zurzeit die beste zeitliche Auflösung wenn der gesamte Planet betrachtet wird[1]. Einige ambitionierte Unternehmen bauen zur Zeit ihre Flotten gerade auf: Beispielsweise gibt das chinesische Unternehmen Chang Guang Satellite an, im Jahr 2020 eine revisit time von 30 Minuten an jedem beliebigen Punkt der Erde erreichen zu wollen [38]. Vereinzelte Gebiete der Erde werden aber durch nicht-nadir Ausrichtung, andere Orbits ohne globale Abdeckung oder durch Überschneidungen von Aufnahmen in höheren Breitengraden bei sonnensynchronen Orbits, auch deutlich häufiger beobachtet.

Kostenlos verfügbare, globale Satellitendaten werden im Rahmen des amerikanischen Landsat- oder dem europäischen Copernicus-Erdbeobachtungsprogramm frei zur Verfügung gestellt.

[1] Betrachtet werden nur hochauflösende Satelliten im erdnahen Orbit. Geostationäre Satelliten nehmen laufend Daten auf, sind mit Auflösungen im km-Bereich aber für diese Arbeit nicht relevant.

5.2 Auswahl für das System

Sentinel-2 ist ein Bestandteil des Copernicus-Programms der europäischen Union. Die Mission besteht aus zwei Satelliten auf dem gleichen sonnen-synchronen Orbit, verschoben um $180°$ und wird betrieben von der europäischen Weltraumorganisation ESA. Das von Sentinel-2 verwendete Datenmaterial ist vergleichsweise komplex und besteht aus mehreren Dateien unterschiedlicher Auflösung und einer Vielzahl von Metadaten. Außerdem ist Sentinel-2 mit $10\,\mathrm{m}$ Auflösung im sichtbaren Bereich die beste frei verfügbare Satellitendatenquelle.

PlanetScope ist eine Satelliten-Konstellation des Unternehmens Planet und besteht aus mehr als 130 Kleinstsatelliten ($10\,\mathrm{cm} \times 10\,\mathrm{cm} \times 30\,\mathrm{cm}$) die laufend erneuert und erweitert werden. PlanetScope wurde als Datenquelle wegen der hohen zeitlichen Auflösung gewählt. Der Zugriff auf die sonst kostenpflichtigen Daten wurde durch das *Planet Education and research program* ermöglicht.

Der Landesbetrieb Geoinformation und Vermessung Hamburg, sowie alle anderen Vermessungsverwaltungen in Deutschland erstellen digitale Orthophotos in $20\,\mathrm{cm}$ (DOP20) und $40\,\mathrm{cm}$ (DOP40) Auflösung und stellen diese in einheitlichen Formaten zum Teil kostenfrei bereit. Die Luftbilder liefern die nötige Auflösung für das zu implementierende Model, werden aber leider nur jährlich erhoben.

Planet - PlanetScope Ortho	
Spektrale Auflösung	4 Bänder (RGB + NIR)
Radiometrische Auflösung	12 bit
Räumliche Auflösung	$3{,}7\,\mathrm{m}$
Zeitliche Auflösung	täglich
Räumliche Abdeckung	alle Landmassen der Erde
Zeitliche Abdeckung	seit 2017 mit voller zeitlicher Auflösung von einem Tag, davor mindestens seit Juni 2016 in heutigem Technologiestand
Produkte	Basic Scene (L1B, top of atmosphere), Ortho Scene (L3B, orthorektifiziert, bottom of atmosphere), Ortho Tile (L3A, Mosaik aus mehreren Ortho Scenes), $24 \times 8\,\mathrm{km}^2$
Zugang	HTTP API des Betreibers
Preis	kostenpflichtig

ESA Copernicus Mission - Sentinel-2	
Spektrale Auflösung	12 multispektrale Bänder
Radiometrische Auflösung	12 bit
Räumliche Auflösung	$10\,\mathrm{m}$ bis $60\,\mathrm{m}$; RGB Bänder in $10\,\mathrm{m}$
Zeitliche Auflösung	alle 5 Tage
Räumliche Abdeckung	alle Landmassen und Küsten ohne Polkappen
Zeitliche Abdeckung	Sentinel-2A seit Juni 2015, Sentinel-2B seit März 2017, daher bis 2017 nur halbierte zeitliche Auflösung von 10 Tagen
Produkte	L1C (top of atmosphere) und L2A (bottom of atmosphere), Fläche von $100 \times 100\,\mathrm{km}^2$ pro Aufnahme, orthorektifiziert in UTM/WGS84 Projektion
Zugang	HTTP API des Betreibers und zusätzliche Downloadmöglichkeiten über AWS S3, bereitgestellt von einem externen Distributors
Preis	Open Data
Landesbetrieb Geoinformation und Vermessung Hamburg - DOP20	
Spektrale Auflösung	4 Bänder (RGB + NIR)
Radiometrische Auflösung	8 bit
Räumliche Auflösung	$0{,}2\,\mathrm{m}$
Zeitliche Auflösung	jährlich im Frühling und unregelmäßig zusätzlich im Sommer (belaubte Bäume)
Räumliche Abdeckung	Hamburg Stadt und Umland
Zeitliche Abdeckung	seit 2013
Produkte	RGB + IR oder CIR (3 Bänder, NIR + rot + grün) in UTM Kacheln in $1 \times 1\,\mathrm{km}^2$, GeoTIFF, orthorektifiziert; bessere Qualität mit kostenpflichtigem TrueDOP Produkt
Zugang	.zip Datei und Kartendienst (WMS, WMTS)
Preis	kostenlos

6 Produkte und Services

Im folgenden Teil wird auf Services verwiesen, die zumindest zu einem Teil optische Satellitendaten durch Machine Learning Methoden automatisiert auswerten und dem in dieser Arbeit zu entwickelndem System ähnlich sind, also eine API oder Web Applikation anbieten.

Die Vision Services [41, 28, 10] der drei großen Cloud-Provider bieten derzeit Werkzeuge wie Sentiment-Analyse, OCR, Sehenswürdigkeiten- oder Personenerkennung an, jedoch keine geospezifischen Produkte[1].

Einige Anbieter [15, 2, 9] stellen ihre Infrastruktur als Plattform mit API Zugang für eigene Auswertungen zu Verfügung. Unterschiede gibt es im Umfang der verfügbaren Satellitendaten und zusätzlich angebotene Daten anderer Art. [15] ermöglicht es Python Code ausführen zu lassen, die Lösung GDBX von [9] bietet Zugang nur zu hauseigenen Satelliten und frei verfügbaren Bilddaten, hat allerdings mehr Möglichkeiten zur Laufzeit. Es ist möglich eigenen Code in Jupyter Notebooks interaktiv oder als Docker Container ausführen zu lassen.

Services zur Objekterkennung wie Autos, Gebäude, Schiffe, Flugzeuge, Baustellen oder Pools bieten [16, 40, 9, 18]. Die Landnutzung[2] kann ebenfalls bestimmt werden. Der Zugang ist dabei sowohl über eine API, als auch über ein Webinterface möglich indem auf einer Karte gewünschte Regionen und Objektkategorien ausgewählt und erkannt werden können.

Die meisten Kategorien bietet dabei das Produkt AnswerFactory von [9]. Andere [16, 40, 18] bieten dafür auch Alarmierungsfunktionalität bei der Erkennung von Änderungen in neuen Daten. Der Kunde kann teils auch Regeln hinterlegen, so dass z.B. nur alarmiert wird, wenn innerhalb einer gewissen Zeitspanne weniger Schiffe als üblich den Hafen verlassen, oder wenn ein Fahrzeug im zu überwachenden Gebiet erkannt wird.

Zusätzlich zu einer Objekterkennung gibt es auch fertige Lösungen für spezifische Anwendungsfälle.

[1] Google bietet mit Earth Engine jedoch für Wissenschaftler Zugriff auf viele freie Satellitendaten, Rechenleistung und Algorithmen.

[2] Bei einer Landnutzungsbestimmung wird erkannt welche Flächen bspw. Wasser- oder Agrarflächen sind oder für Wohnungen genutzt werden.

[18] bietet für zwei Kategorien zusätzliche höherwertige Dienste an. Das Produkt GO Energy überwacht Raffinierien und erkennt weltweit Öltanks und errechnet das enthaltene Ölvolumen[3]. Das zweite Produkt GO Consumer zählt belegte und freie Parkplätze u.a. vor Einkaufszentren oder in Produktionsstätten der Autoindustrie. Das Unternehmen nutzt eine Vielzahl von Optischen- und Radar-Satelliten und kombiniert weitere Informationen aus geolocation Daten von Mobiltelefonen und kann so je nach Verfügbarkeit mehrmals täglich Daten erfassen. Auf dieser Basis werden Reporting-Dienste angeboten um sich aktuell über einzelne Wettbewerber oder über die gesamte Branche zu informieren.

Das Produkt Ocean Finder von [40] kann Schiffe tracken und gekaperte Schiffe wiederfinden. RefineryScanner trifft Vorhersagen zum Ausfall von Raffinieren. Mit Stack Insight ist es Minenbetreibern möglich, Volumina von abgetragenem Material zu berechnen.

Im Bereich der Waldüberwachung gibt es Produkte mehrerer Anbieter. Das Produkt Starling von [40] richtet sich eher an Unternehmen wie Plantagenbesitzer, die illegalen Waldschlag verhindern möchten. Der Anbieter [37] hat sich auf die Erkennung von Abholzung spezialisiert und richtet dabei sich eher an Regierungs- und Nicht-Regierungsorganisationen. [32] erkennt weitere Parameter eines Waldes wie Schädlingsbefall und macht Vorhersagen.

Mit EnergySights bietet [32] eine Web Application für eine Satellitenbild-unterstützte Pipelineüberwachung zur Erkennung von Diebstahl oder Leckage.

Zuletzt ist noch der Anbieter [39] zu nennen, der auf seiner Website ähnliche Anwendungsfälle beschreibt. Allerdings ist nicht klar ob dies *fertige* Services sind oder für einzelne Kunden zugeschnittenen Produkte darstellen. [2] beschreibt eine Plattform allerdings ohne weiterführende Informationen.

In der Regel entwickeln alle Anbieter auf Basis ihrer Plattformen spezifische Lösungen für einzelne Kunden.

Die Anbieter [9, 39, 40, 2] betreiben dabei auch eigene Satellitenflotten.

Auch wenn bei diesen Anwendungen viele Fortschritte durch Deep Learning erzielt werden konnten, sind manuelle Eingriffe durch Experten nach wie vor notwendig. Um die Genauigkeit bei der Erkennung von building foorprints zu erhöhen setzt [9] auf manuelle Nachkontrolle [8]. [18] vermisst einen erstmalig erkannten Öltank nach automatischer Detektion zunächst manuell [19].

[3] Floating Roof Tank (FRT). Der Deckel hebt und senkt sich je nach Füllstand. Das Volumen kann durch eine Analyse des Schattenwurfs errechnet werden.

7 Systembeschreibung

Dieses Kapitel enthält den Hauptteil dieser Arbeit und beschreibt die Anforderungen und den Entwurf des Systems mit allen Teil-Komponenten. Danach wird erläutert wie sichergestellt wurde, dass zunächst nur authorisierte User Zugriff zum System bekommen. Den Abschluss dieses Kapitels bildet eine Beschreibung der umgesetzten Cloud-Umgebung. Zunächst aber soll auf die potentiellen Anwender des Systems eingegangen werden.

7.1 Anwender

In der Recherche kristallisierten sich vier Anwendergruppen heraus die das zu entwickelnde System nutzen könnten.

GIS Poweruser

Benutzt ein GIS Programm, wie QGIS oder ArcGIS Pro und möchte das System als weitere Informationsquelle darin integrieren. Beispielsweise soll das Ergebnis eines Models[1] als ein Tilelayer in der GUI angezeigt werden können. Außerdem schreibt er Shell-Skripte zur Datenverarbeitung und möchte einige seiner Daten durch, vom System bereitgestellte Models analysieren lassen. Dazu verwendet er einen HTTP Client wie curl. Als fortgeschrittener Anwender würde er für sein GIS Programm ein Python-Plugin schreiben um komfortabler mit dem System zu interagieren. Er möchte selbst entwickelte Models im System verwenden, um von der größeren Datenbasis und den erweiterten Rechenkapazitäten zu profitieren, die ihm lokal nicht zur Verfügung stehen.

[1] Mit *Model ist im folgenden eine Komponente des Systems gemeint, die als Blackbox behandelt wird, eingegebene Daten analysiert und Ausgaben dem System zur Verfügung stellt.*

Developer

Möchte das System in einem eigenen Produkt verwenden und benötigt eine API Schnittstelle. Die Anwendung des Developers soll die vom System erzeugten Ergebnisse eines Models weiterverarbeiten oder in einer anderen Art der Präsentation darstellen.

GIS User

Möchte eigene Daten mit vom System bereitgestellten Models analysieren die er durch einen Drohnenflug aufgenommen oder bei einem Satellitenanbieter heruntergeladen hat. Möchte dabei die Analyse in einer einfachen GUI durchführen lassen und die Ausgaben im Browser einsehen, um sich schnell einen Überblick über das Ergebnis zu verschaffen. Bei Bedarf sollen die Ergebnisse auch heruntergeladen und in einem lokalen GIS Programm genutzt werden. Will interessante Ergebnisse unseres Dienstes mit externen Usern teilen, indem er eine Karte mit dem Ergebnis der Analyse auf seinem Blog einbindet. Für die Leser des Blogs soll unser Dienst transparent sein, also keine Anmeldung erfordern.

Externer User

Hat keine Kenntnis vom System und möchte Ergebnisse, die er als URL zugeschickt bekommen hat, herunterladen. Betrachtet eine von unserem System erstellte Karte über die Website eines Dritten der diese dort eingebunden hat.

7.2 Anforderungen an das System

Die Anforderungen an das System orientieren sich an den Bedürfnissen und Erwartungen der Anwender. Es sollen grundsätzlich zwei Schnittstellen zum System geschaffen werden: Eine HTTP API die alle notwendigen Operationen zur Verfügung stellt. Zusätzlich soll eine Browser Applikation geschrieben werden die einen einfacheren, für einige Anwendungsfälle optimierten, grafischen Zugriff auf die Funktionen der API bietet.

7.2.1 Funktionale Anforderungen

API

- Analyse vom User bereitgestellter Daten (Upload-Möglichkeit)
- Analyse vom System bereitgestellter Daten (Bezug der Daten von, an das System angebundener Datenanbieter)
- User kann lokal entwickelte Algorithmen dem System zur Verfügung stellen und zur Analyse einsetzen
- User kann vom System bereitgestellte Modells zur Analyse nutzen
- Möglichkeit Ergebnisse, Models und Daten mit anderen Benutzern zu teilen

Web Applikation

- Frontend für die wichtigsten API Operationen zur einfacheren Verwendung (dadurch keine Kenntnis der API erforderlich)
- Visualisierung der Analyse-Ergebnisse im Browser (kein lokales GIS Programm notwendig)

7.2.2 Nicht-funktionale Anforderungen

Erweiterbarkeit Es muss möglich sein in Zukunft weitere Model-Ausgabeformate festzulegen. Die Eingabeformate dürfen nicht in Abhängigkeit zu den Ausgaben der Datenanbieter definiert werden, um eine Anbindung weiterer Datenanbieter mit anderen Eigenschaften nicht zu erschweren. Das System ist für 2D Daten ausgelegt und muss eine Vielzahl der branchenüblichen Bild- und Metadatenformate verarbeiten können.

Fixkosten für den Betrieb Während der Implementationsphase wird das System nicht im Dauerbetrieb benötigt. Da die Infrastruktur für das Projekt bei einem Cloud-Anbieter betrieben werden wird, ist darauf zu achten, dass keine oder nur geringe Kosten anfallen wenn das System nicht verwendet wird. Die Wiederinbetriebnahme sollte möglichst schnell und automatisiert durchgeführt werden können.

Flexible Kosten in der Anwendung Je nach Anfrageparametern können sehr viele oder sehr wenige Ressourcen benötigt werden. Die Datenmenge (gemessen in Pixel oder Byte) ist linear abhängig von der zu analysierenden Fläche. Zum Beispiel wird die Analyse eines Gebiets in Größe eines Werksgeländes ungleich weniger Ressourcen benötigen, als eine Auswertung in Größe eines Bundeslands. Die genutzte Infrastruktur muss daher schnell hoch- und herunterskaliert werden können um Kosten zu sparen.

7.3 Entwurf

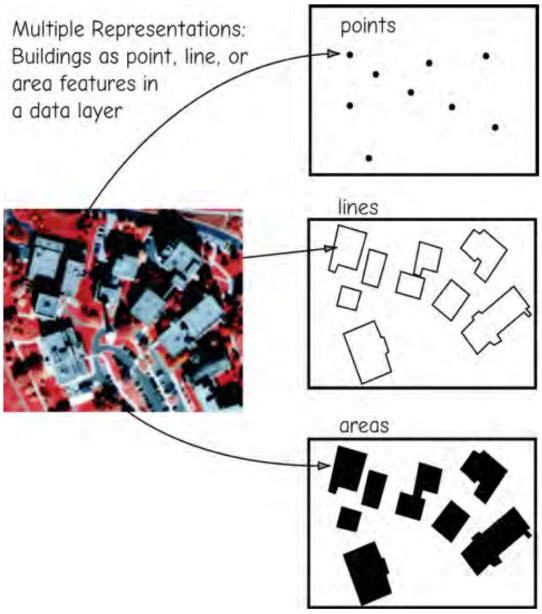
Dieses Kapitel beginnt mit einer Vorstellung der Ein- und Ausgaben des Systems und erläutert daran das implementierte Datenmodell. Danach folgt die detaillierte Beschreibung des Systemdesigns.

7.3.1 Ein- und Ausgaben des Systems

Das System verarbeitet Daten die als Rasterdaten oder als Vektordaten modelliert werden können.

Rasterdatenmodelle definieren die Welt als eine Menge von regelmäßig angeordneten Zellen in einem Gitternetz. Die zu modellierenden Entitäten oder Phänomene werden durch Attributwerte dargestellt und einer Zelle zugeordnet. Rasterdatenmodelle eignen sich für kontinuierliche, räumliche Eigenschaften wie Höhe, Steigung oder Schadstoffbelastung [3, p. 54].

Vektordatenmodelle definieren konkrete Objekte durch eine Menge von Koordinaten und assoziierten Attributen.[3, p. 42] Ein Vektordatenmodell definiert geometrische Formen wie Punkte, Linien oder Polygone und gibt Regeln für die Verbindung und Anordnung dieser vor. Zum Beispiel könnte ein Model eines Pipeline-Netzes, dass durch den Verbund mehrerer Linien dargestellt wird, einen Mindestabstand vorgeben, sodass keine Objekte näher als $10\,\mathrm{m}$ erlaubt sind.



Entnommen aus: [3, p. 43]
Abbildung 7.1: Verschiedene Darstellungen des selben Objekts. Je nach Anwendungsfall ist es sinnvoll die dargestellten Gebäude als Punkte, Linien oder ausgefüllte Polygone darzustellen.

Systemeingaben

Dateien die zuvor von einem Datenanbieter herunter- oder von einem User hochgeladen wurden umfassen gängige Bildformate wie JPEG oder PNG. Diese haben keine standardisierte Möglichkeit die, für die Branche notwendigen, zusätzlichen Informationen zur Georeferenzierung abzuspeichern.

Daher werden diese Daten entweder separat in einem sogenannten World file (Konzept eines sidecar files) oder in einem speziellen, oft proprietären Rasterdatenformat abgespeichert. Ein gängiges Format, dass Geoinformationen zusammen mit den Rasterdaten speichern kann ist zum Beispiel GeoTIFF, das auf dem Bilddateiformat TIFF basiert.

Multispektrale Daten werden bei mehr als drei Bändern meist in einzelnen Dateien ausgeliefert. Zusätzliche Metadaten über den Zustand des Aufnahmeinstruments oder atmosphärische Bedingungen werden separat in nicht standardisierten (nur für diesen Anbieter gültigen) XML oder JSON Dateien gespeichert oder können nur über eine API des Anbieters abgerufen werden.

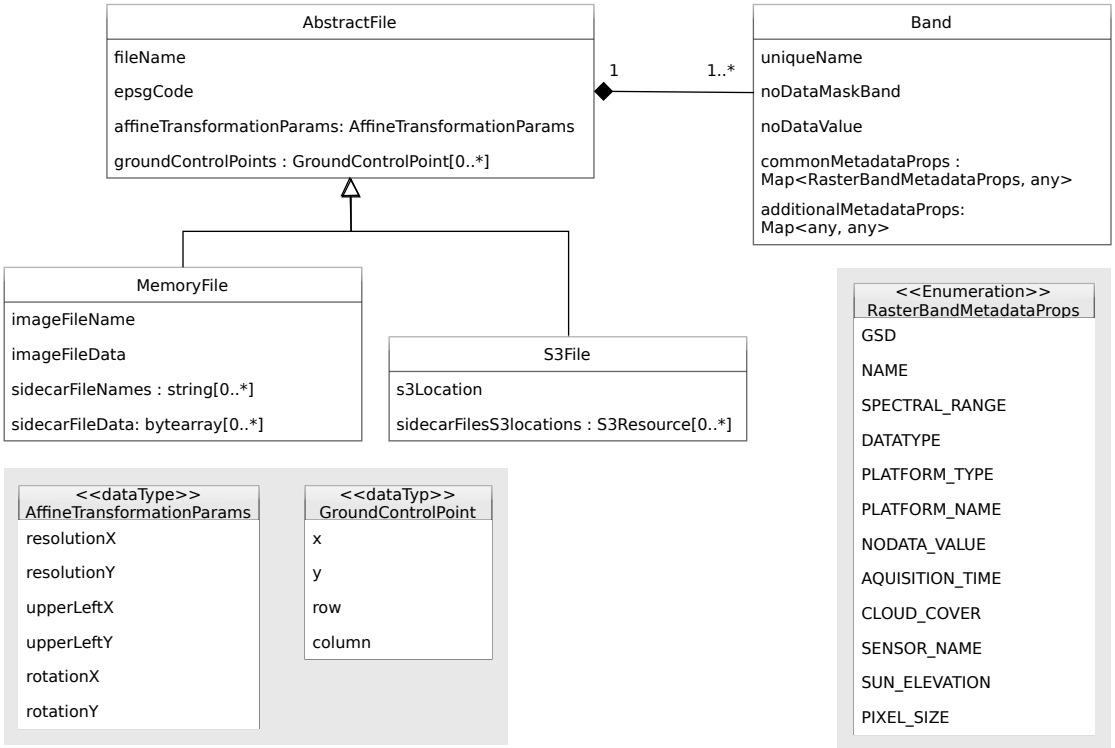


Abbildung 7.2: Datenmodell einer Bilddatei. Eine Datei kann aus mehreren Bändern mit eindeutigen Namen bestehen und kann im System im Arbeitsspeicher oder in Form einer Referenz auf eine Datei im S3 Datenspeicher vorliegen.

Die in Abbildung 7.2 dargestellte Klasse *AbstractFile* modelliert eine Bilddatei und beinhaltet Parameter für eine affine Transformation oder eine Liste von Ground Control Points. Beides sind gängige Angaben zur Georeferenzierung eines Bildes.

Der EPSG Code ist eine Referenz auf einen Eintrag des EPSG Datensatzes[2]. Dieser enthält die gebräuchlichsten Koordinatenreferenzsysteme und Koordinatentransformationen.

Ein Rasterband welches als *noDataMaskBand* gekennzeichnet ist, bestimmt für alle anderen Bänder einer Datei ungültige Pixel. Alternativ kann ein bestimmter Pixelwert (*noDataValue*) dies anzeigen. Es gibt keine einheitliche Definition von einem *ungültigem* Pixel. Meist sind dies Pixel für die keine Daten vom Sensor aufgenommen wurden und durch die Transformation in das zweidimensionale Koordinatensystem entstanden sind. Beispielsweise könnte ein ungültiger Pixel aber auch eine

[2] Der EPSG Datensatz wird von der International Association of Oil & Gas Producers gepflegt. Weitere Informationen unter http://www.epsg.org/

Angabe sein, dass die Oberfläche an dieser Stelle durch Wolken verdeckt wurde. Jedes Model legt selbst fest wie diese Werte interpretiert werden.

Das System definiert die Semantik einiger Metadaten und verarbeitet diese intern. Weitere Metadaten eines Bandes können separat abgespeichert werden (*additionalMetadataProps*), allerdings werden diese nur eingeschränkt berücksichtigt.

Im System liegen die Rasterdaten einer solchen Datei im Arbeitsspeicher vor oder als Referenz auf eine oder mehrere S3 Objekte. Der User kann Dateien entweder hochladen oder eine S3-Referenz angeben. Hostet ein Datenanbieter sein Archiv ebenfalls mit S3 kann so unnötiger Traffic vermieden werden.

Systemausgaben

Um komfortabel mit großen Datenmengen im Browser oder einem GIS Programm arbeiten zu können werden diese nicht in einem Stück heruntergeladen, sondern in kleinen Kacheln mit unterschiedlichem Detailierungsgrad (Zoomstufe) ausgeliefert (wie bspw. bei Google Maps).

Das System bietet diese Möglichkeit für Raster- und Vektordaten an um damit die eingegebenen Bilddaten und die Ergebnisse der Analyse ausliefern zu können.

Zusätzlich können die Ergebnisse in Form von segmentation maps heruntergeladen werden. Dies sind Bitmaps die für jeden Pixel eine Zugehörigkeit zu einer Klasse definieren (z.B. Pixelwert 1 = Vegetation, 2 = Wasser). Diese sind low-level Produkte des Systems und nützlich, wenn mit diesen Werten weitergearbeitet werden soll oder die höherwertigen Ausgabevarianten nicht möglich sind. Dies wäre der Fall, wenn die Daten nicht georeferenziert werden können und damit nicht auf einer Karte anzuzeigen sind.

Die folgenden Ausgabetypen wurden umgesetzt:

- Rasterkacheln werden als XYZ-Tiles[3] ausgeliefert
- Vektorkacheln werden als Mapbox vector tiles[4] ausgeliefert
- Vektordaten werden in einer Datei im GeoJSON Format[5] ausgeliefert
- Segmentation Maps werden als TIFF Dateien ausgeliefert

[3] https://wiki.openstreetmap.org/wiki/Slippy_map_tilenames
[4] https://docs.mapbox.com/vector-tiles/specification/
[5] https://tools.ietf.org/html/rfc7946

7.3.2 Komponten des Systems

Die Funktionalität des System ist in vier grundlegende Komponenten aufgeteilt. Dabei werden die zu analysierenden Satellitendaten von der Datapoint-Komponente bereitgestellt und durch die Model-Komponente analysiert. Die Aufgabe der Processing-Komponente ist es, die Verarbeitung zu starten, die Daten vorzubereiten und nach einer abgeschlossenen Analyse die Ergebnisse für den User zu erstellen. Dabei delegiert sie Aufgaben an die beiden anderen Komponenten weiter. Die Place-Komponente ist nicht an der Datenverarbeitung beteiligt und unterstützt den User nur bei der Erstellung der Anfrage zur Analyse von Bilddaten.

7.3.2.1 Datapoint-Komponente

Die Datapoint-Komponente ist der Datenlieferant des Systems und unterstützt den User die richtigen Daten für die Analyse auszuwählen indem sie *Datapoints* bereitstellt.

Ein Datapoint repräsentiert eine Menge von Bilddaten eines oder mehrerer Datenanbieter und ist aus *DatapointComponent*s aufgebaut. Eine *DatapointComponent* besitzt eine Fläche (entspricht der Fläche der Erde die in den Bilddaten abgebildet ist) und einen Zeitpunkt (genauer Zeitspanne zwischen Aufnahmebeginn und -ende) und repräsentiert eine Aufnahme eines Anbieters.

Durch Angabe eines Suchgebiets[6], einer Zeitspanne und einem Model kann eine Menge von *Datapoints* ermittelt werden. Dabei decken die *datapointComponent*s in einem Mosaik aus ihren Teilflächen das gesamte Suchgebiet ab. Die Zeitspanne des *Datapoint*s ergibt sich aus der ältesten und der neuesten *datapointComponent*.

Die Erstellung einer *DatapointComponent* ist Aufgabe eines *DatapointProvider*s. Ein *DatapointProvider* implementiert die Schnittstellen des Anbieters der Daten und kann eine zuvor herausgegebene *DatapointComponent* durch Download der Daten beim Anbieter in die tatsächlichen Bilddaten überführen.

[6] Die Fläche wird als WKT (Well-known text) angegeben. WKT ist eine einfache Beschreibungssprache von Vektorgeometrie auf einer Karte

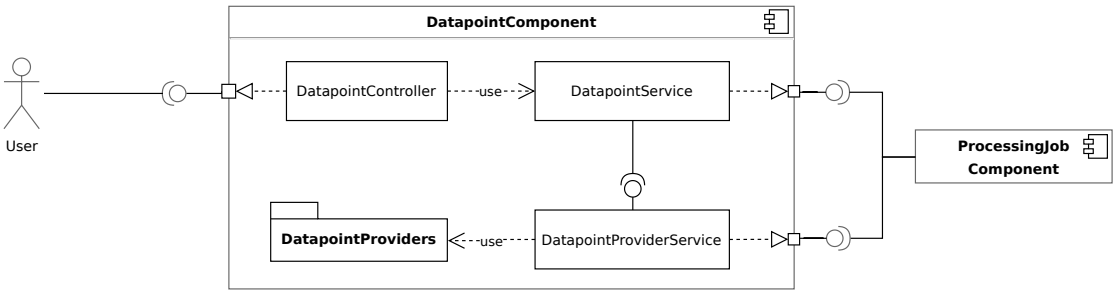


Abbildung 7.3: Die Datapoint-Komponente stellt dem User passende *Datapoint*s zur Verfügung. Hat sich der Benutzer für mindestens einen *Datapoint* entschieden, startet er die Analyse über die ProcessingJob-Komponente, die die *Datapoint*s bei der Datapoint-Komponente gegen die Bilddaten eintauscht. Die dafür notwendigen Operationen werden an anbieterspezifische Implementationen, den *DatapointProvider*n, delegiert.

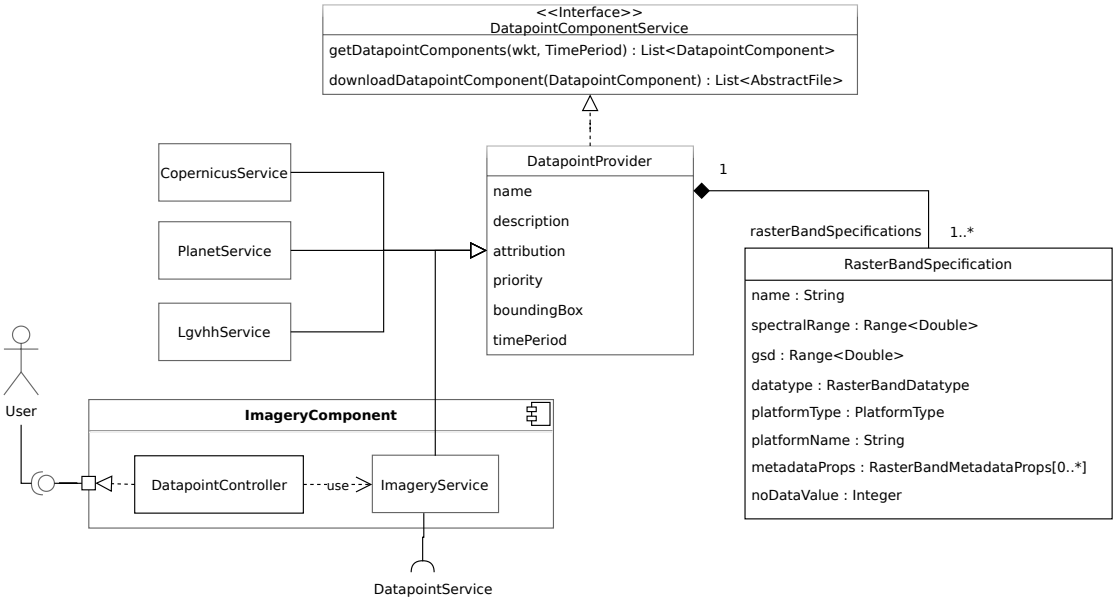


Abbildung 7.4: Datenmodell eines *DatapointProvider*s. Ein *DatapointProvider* hat die Aufgabe *DatapointComponent*s zu erstellen und Daten eines Datenanbieters herunterzuladen. Es werden bei der Erstellung nur *DatapointProvider* verwendet die mit den Anforderungen des *Model*s kompatibel sind.

DatapointProviders Ein *DatapointProvider* stellt dem System Bild- und Metadaten einer Datenquelle (z.B. Daten eines Satellitenbetreibers) zur Verfügung. Das System nutzt die Angaben der *RasterBandSpecification*s um den oder die richtigen

DatapointProvider zu finden. Außerdem muss ein *DatapointProvider* angeben, für welchen Zeitraum und für welches Gebiet er Daten liefern kann.

Zusätzlich zu den in 5 genannten externen Datenquellen gibt es mit *ImageryService* eine weitere Variante der Klasse *DatapointProvider*. Diese ist Teil der Imagery Komponente die, wie in Abbildung 7.4 angedeutet, von extern erreichbar ist und dem User den Upload von eigenen Daten ermöglicht.

Nach dem Upload von Bilddaten erkennt die Komponente soweit möglich evtl. enthaltene Metadaten automatisch. Weitere Metadaten können vom User ergänzt werden. Sind alle Metadaten vorhanden um den Contract eines *DatapointProvider*s zu erfüllen, kann die Imagery Komponente diese Daten als weiteren *DatapointProvider* im System mithilfe des *DatapointService* registrieren.

Erstellung von Datapoints Die Auswahl welche *DatapointComponent*s zusammen einen *Datapoint* ergeben, wird durch eine Variante der *DatapointCreationStrategy* bestimmt. Es wurde nur eine einfache Strategie implementiert, die im nächsten Abschnitt genauer behandelt wird. In einem Produkt müssten einige vernachlässigte Kriterien eine Rolle bei der Auswahl spielen.

Unter anderem sollte die Priorität des *DatapointProvider*s berücksichtigt werden. Die Eigenschaft *property* (siehe Abbildung 7.4) ist eine Ganzzahl und dafür gedacht eine Auswahl zu ermöglichen wenn *DatapointComponents* unterschiedlicher *DatapointProvider*s zur Verfügung stehen. Die Priorität könnte dabei den Preis, die Datenqualität, die Verfügbarkeit (Zeit bis die Daten heruntergeladen worden sind) oder andere Parameter für Bildmaterial Anbieters kodieren.

Die beste Qualität wird bei möglichst wenigen *datapointComponent*s erzielt, da jede Aufnahme einen anderen Zeitpunkt des Suchgebiets zeigt und sich Artefakte an den Schnittstellen der Aufnahmen ergeben. Es sollte für den User wählbar sein, ob bei der Auswahl möglichst große Flächen oder möglichst kurze zeitliche Abstände zwischen den Aufnahmen favorisiert werden sollen.

Auch sollten Attribute aus den Metadaten in den *RasterBandSpecifications* bei der Auswahl mit einbezogen werden. Beispielsweise kann es für manche Modelle wichtig sein, dass alle Daten möglichst identische spektrale Eigenschaften haben. Wenn nur Daten mit einer geringen Auflösung benötigt werden macht es zudem Sinn, keine hochauflösenden *DatapointProvider* zu bevorzugen, da die Daten vermutlich teurer sind, im Download länger brauchen und anschließend zunächst herunterskaliert werden müssten.

```
area := EMPTY_AREA
datapoints := EMPTY_LIST
datapoint := NULL
for all datapointComponent in datapointComponents do
    intersection = Schnittmenge von datapointComponent
    und area_target
    if area is EMPTY_AREA then
        area := intersection
        überschreibe Fläche der datapointComponent mit in-
        tersection
    else if not area enthält intersection then
        area := Vereinigungsmenge vonintersection und area

        überschreibe Fläche der datapointComponent mit
        Vereinigungsmenge von intersection und area
    else
        continue
    end if
    füge datapoint datapointComponent hizu
    if area entspricht area_target then
        füge datapoints datapoint hizu
        datapoint := NULL
        area := EMPTY_AREA
    end if
end for
```

Algorithm 1: Algorithmus zur Erzeugung eines Datapoints

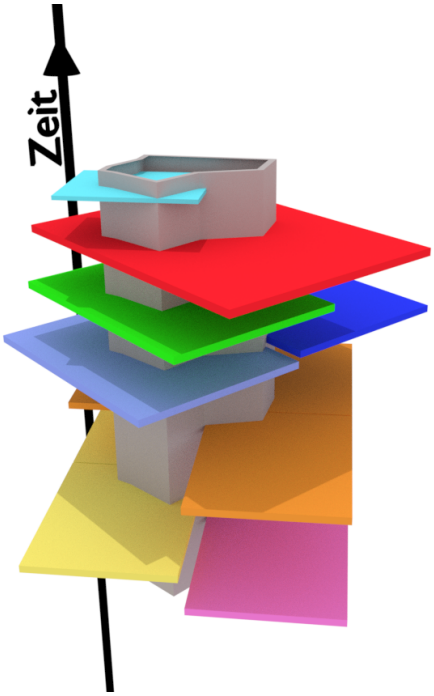


Abbildung 7.5: Erstellung eines Datapoints. Der hohle graue Körper entspricht dem Suchbereich. Die farbigen Ebenen deuten die datapointComponents an.

Beispiel Die in Abbildung Abbildung 7.5 dargestellten *datapointComponent*s werden von neu nach alt durchlaufen und bilden 3 *Datapoint*s.

Datapoint 1 bestehend aus datapointComponent 1 + 2. Datapoint 2 bestehend aus datapointComponent 3 + 4. Datapoint 3 bestehend aus datapointComponent 5 + (6 + 7 + 8) [orange]. datapointComponents 9 + 10 (gelb) und 11 (rosa) bilden zusammen keine Fläche die das graue Suchgebiet komplett einschließt und werden verworfen.

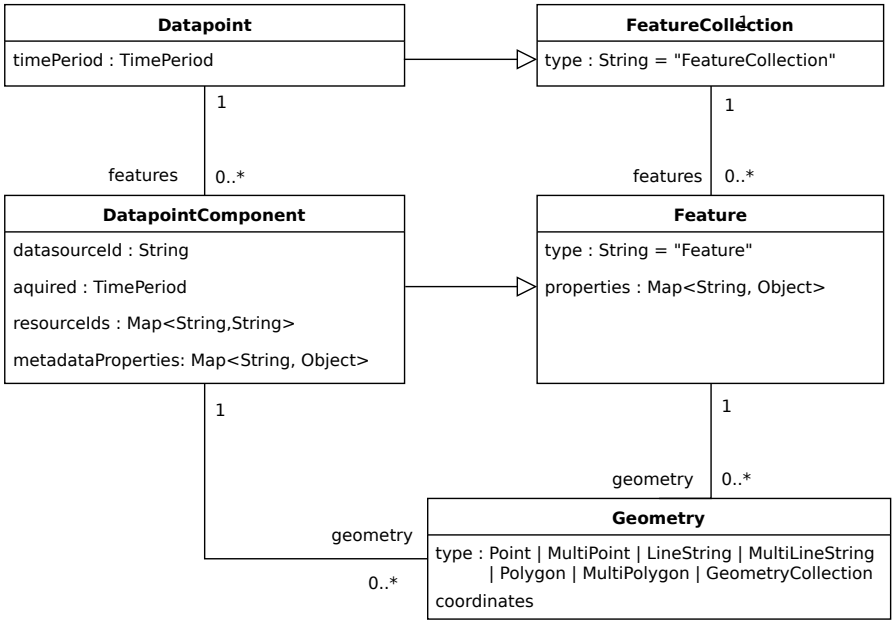


Abbildung 7.6: Datapoint Datenmodell. Ein Datapoint ist ein, um einige Properties erweitertes GeoJSON. Somit können die meisten geo-libraries und GIS Programme die als JSON serialisierten Objekte out-of-the-box anzeigen.

7.3.2.2 Model-Komponente

Die Model-Komponente ermöglicht dem User passende Models zu finden und neue Models zu erstellen. Dabei erstellt sie die nötige Infrastruktur zur Ausführung des Models.

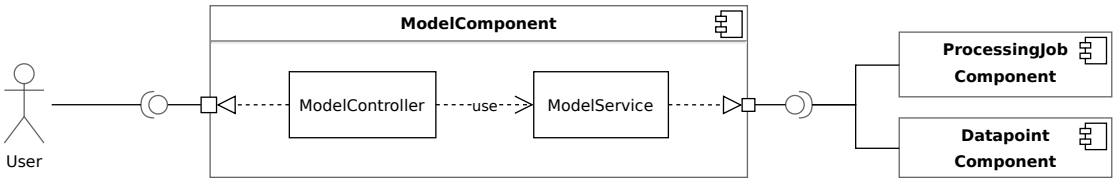


Abbildung 7.7: Die Datapoint-Komponente nutzt die Metainformationen eines Models um passende Datapoints zu erstellen, die den Anforderungen des Models genügen. Die ProcessingJob-Komponente verwendet die, durch die Model-Komponente erstellte Infrastruktur, um Daten durch ein Model verarbeiten zu lassen.

Die Datenverarbeitung eines Models wird durch einen Docker Container realisiert. Das dafür benötigte Image muss von einem User erstellt werden oder liegt im Sys-

tem bereits vor. Für jedes Model wird ein Docker Repository eingerichtet in das der User das Image hochladen kann.

Docker wurde als Userschnittstelle gewählt da es auf allen Plattformen verfügbar ist und es in der Machine Learning Community nicht unüblich ist damit zu arbeiten.

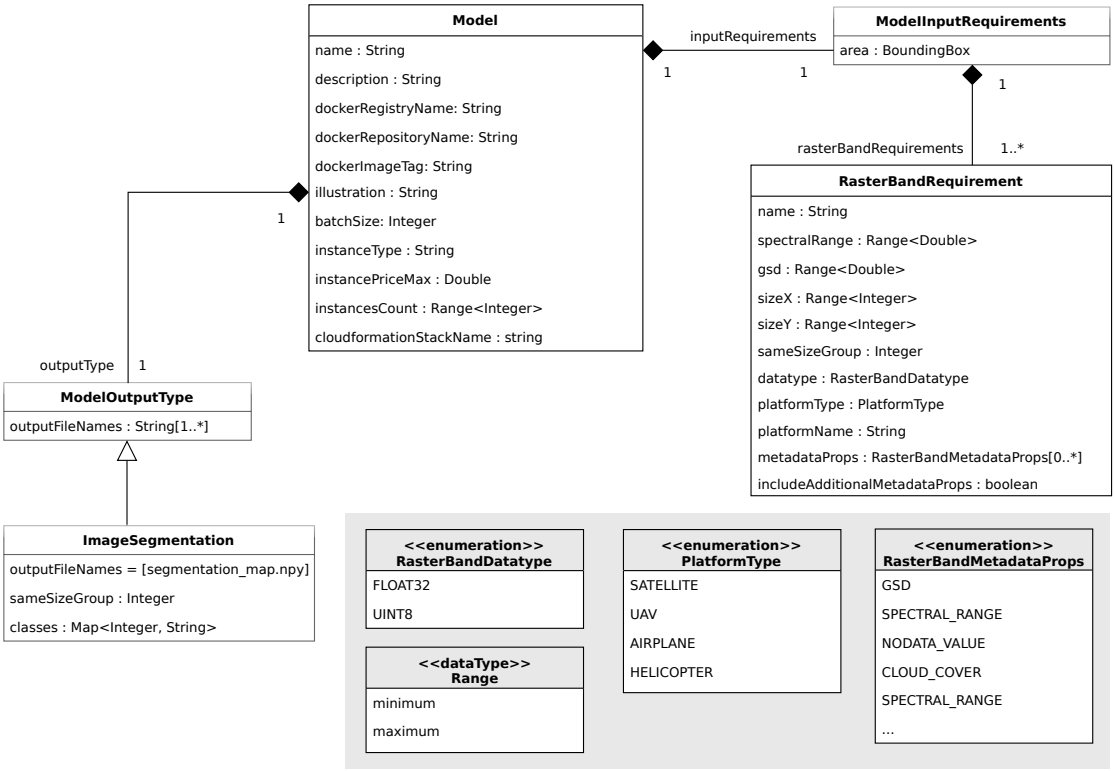


Abbildung 7.8: Model Datenmodell. Ein Modell definiert die Anforderungen an die Infrastruktur (Anzahl und Typ und Preis der virtuellen Maschine) und an die vom System bereitzustellenden Raster- und Metadaten. Zudem gibt ein Model, durch die Wahl einer *ModelOutputType* Implementation an, welche Ausgabedateien es erzeugt und wie diese zu interpretieren sind.

Schnittstelle: System ↔ Docker Container Das System startet den Container und übergibt dabei ein oder mehrere Ordner in denen sich die zu analysierenden Daten befinden. Maximal werden in einem Verarbeitungsschritt BATCH_SIZE Ordner übergeben um die Daten möglichst parallel verarbeiten zu können.

Input des Docker Containers In jedem Ordner befinden sich die folgenden drei Dateien:

tile_input.npz Enthält die Rasterdaten der Bänder. Diese sind kompatibel mit den in *rasterBandRequirement*s definierten Angaben.

tile_nodata.npz Enthält für jedes in *tile_input.npz* enthaltene Band ein NoData-Band das NODATA Pixel markiert.

tile_metadata.json Enthält für jedes in *tile_input.npz* enthaltene Band Metadaten Objekte, bestehend aus den in *metadataProps* angegebenen Key-Value Paaren und ggf. weiteren anbieterspezifischen Attributen in *includeAdditionalMetadataProps*.

tile_input.npz und *tile_nodata.npz* sind Dateien im NPZ Format, einem ZIP Archiv, bestehend aus Dateien im NPY Format [7]. Eine NPY Datei enthält die Bilddaten eines Rasterbandes. Das Format wurde gewählt da es unkompliziert ist, alle gängigen Datentypen unterstützt und Rasterbänder beliebiger Größe speichern kann.

Der Docker Container muss in der Lage sein Daten in diesem Format verarbeiten zu können. Für Python aber auch für weitere Programmiersprachen wie Java oder C++ gibt es Libraries die diese Dateien lesen können.

Beispiel Ein Model definiert in *rasterBandRequirement*s zwei Bänder. Das erste definiert die räumliche Auflösung, angegeben als *gsd* mit minimum und maximum = 10 (Meter), das zweite Band jeweils mit 20. Beide Bänder erfordern die Metadaten *CLOUD_COVER* und *GSD* durch Angabe in den *metadataProps*. Zusätzliche Metadaten des Anbieters werden erwünscht (*includeAdditionalMetadataProps*). Die minimale und maximale Größe der Pixeldaten wurde mit (256,512) in X-Richtung und (256,512) in Y-Richtung vorgeschrieben.

Das System erzeugt bei diesen Angaben drei Artefakte: *tile_input.npz* und *tile_nodata.npz*, bestehend jeweils aus zwei 2D-Arrays der Größe (512,512) als NPY Datei. Außerdem die in 7.1 dargestellten Metadaten.

```
[
  {
    "commonMetadataProps": {
      "CLOUD_COVER": 0.09,
      "GSD": 10.0
    },
    "additionalMetadataProps": {
```

[7] Ein Format um numpy Arrays zu serialisieren. numpy wird systemintern verwendet um Rasterdaten zu verarbeiten. Weitere Informationen unter https://docs.scipy.org/doc/numpy/reference/generated/numpy.lib.format.html#module-numpy.lib.format

```
      "processinglevel": "Level-2A",
      "s2datatakeid":  "GS2A_20180612T095031_015520_N02.08",
      "orbitnumber": 15520,
      "name": "relativeorbitnumber": 79
    }
  },
  {
    "commonMetadataProps": {
      "CLOUD_COVER": 0.09,
      "GSD": 20.0
    },
    "additionalMetadataProps": {
      "processinglevel": "Level-2A",
      "s2datatakeid":  "GS2A_20180612T095031_015520_N02.08",
      "orbitnumber": 15520,
      "name": "relativeorbitnumber": 79
    }
  }
]
```

Listing 7.1: Inhalt der Datei *tile_metadata.json*. Die Objekte der Liste geben Metadaten für jeweils ein Rasterband an. Die Sortierung entspricht der in *rasterBandRequirement*s.

Output des Docker Containers Der Container muss das Ergebnis der Analyse im dazugehörigen Ordner abspeichern. Es werden die in *outputFileNames* angegebenen Dateien erwartet, die das Model über die Wahl des *ModelOutputType* definiert hat.

Beispiel Das im Input Abschnitt eingeführte Model verwendet die Implementation *ImageSegmentation* (siehe Abbildung 7.8). Der Parameter *sameSizeGroup* ist mit 0 angegeben. Diese Zahl ist ein Index und bezieht sich auf *rasterBandRequirement*s. Das System hat dem Model als Eingabe das 0-te Band in den Dimensionen (512,512) ausgeliefert. Daher erwartet es eine Datei mit dem Namen *segmentation_map.npy*, bestehend aus einem Array der Größe (512,512). Die Interpretatin der Werte des Arrays sind in *classes* definiert.

Es wurde mit *ImageSegmentation* nur ein Ausgabetyp implementiert der für das später im Detail beschriebene entwickelte Model benötigt wurde.

7.3.2.3 ProcessingJob-Komponente

Die ProcessingJob-Komponente startet die Verarbeitung von Bild- und Metadaten durch ein Model. Die dafür benötigten Daten erhält sie vom User direkt in Form von Bilddateien und weiteren Parametern innerhalb der Anfrage oder beauftragt die Datapoint-Komponente mit der Bereitstellung der vom User ausgewählten Datapoints.

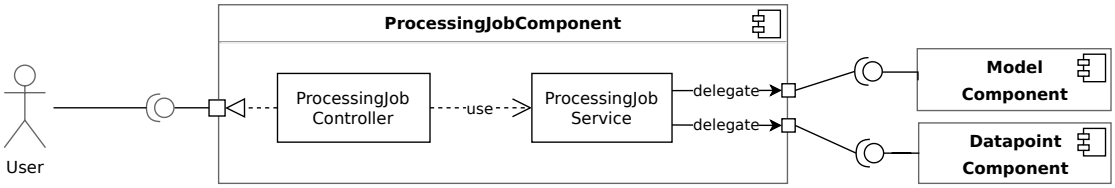


Abbildung 7.9: Schnittstellen der ProcessingJob-Komponente. Die Bereitstellung und Analyse der Daten delegiert sie an andere Komponenten. Die Datapoint-Komponente wird nicht benötigt, wenn der User die auszuwertenden Daten in der Anfrage übergibt.

Sie ist verantwortlich erhaltene Dateien im System zu persistieren und in eine geignete Struktur zu überführen. Dies umfasst den Upload in einen system-eigenen S3 Bucket und die Erfassung in der Spatial Database PostGIS[8].

Die Datenbank speichert die Ausgaben der Models als Vektordaten und zusätzlich eine Referenz zu den Eingabedaten. Weitere Anfragen die das gleiche Model benötigen und auf Dateien verweisen, die bereits verarbeitet wurden, können somit mit Daten aus der Datenbank beantwortet werden. Dies spart Ressourcen wenn ein anderer User die gleichen *datapointComponents* verwendet.

Nur Dateien der Datapoint-Komponente werden in der Datenbank erfasst, da diese prinzipiell von mehreren Usern verwendet werden können und vollständig mit Metadaten ausgezeichnet sind.

[8] PostGIS ist eine Erweiterung der PostgreSQL Datenbank und ermöglicht es geographische Objekte abzuspeichern.

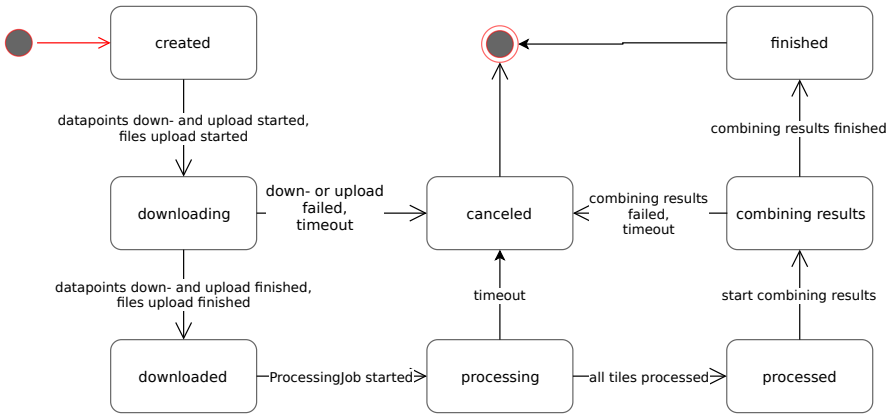


Abbildung 7.10: Lifecycle eines *ProcessingJob*s. Es besteht keine Möglichkeit einen fehlgeschlagenen *ProcessingJob* erneut zu starten. Wird allerdings ein neuer *ProcessingJob* erstellt und wurden bereits Daten eines Anbieters heruntergeladen, werden diese wiederverwendet.

Liegen alle Daten zur Verarbeitung durch ein Model vor, startet die ProcessingJob-Komponente die Verarbeitung. Sie ist weiterhin für die Aufbereitung der Daten zuständig. Dabei werden die Raster- und Metadaten in, für das jeweilige Model geeignete, Eingabedaten überführt.

Dazu gehören:

- Rasterbänder aus den Dateien extrahieren.
- Daten skalieren: Benötigt ein Model nur Daten in einer geringen Auflösung sollten höher aufgelöste Daten herunterskaliert werden um die Anzahl der zu verarbeitenden Pixel zu reduzieren.
- Daten in kleinere Einheiten stückeln: Ein Model gibt die minimale und maximale Größe der Daten an, dies es in der Lage ist zu verarbeiten. Die ProcessingJob-Komponente teilt dazu die Daten in mehrere kleinere Einheiten (bestehend aus *tile_input.npz*, *tile_nodata.npz* und *tile_metadata.json*) auf.
- Verarbeitungsforschritt überwachen: Jede Dateneinheit wird dem Model zugeführt und mit einer ID versehen um den Fortschritt der Verarbeitung zu überwachen.

Wurden alle Dateneinheiten verarbeitet werden aus den Ausgaben des Models die gewünschten Ergebnistypen abgeleitet (Übergang *processed* → *combining_results* in Abbildung7.10). Je nach Produkt entstehen dabei ein oder mehrere Dateien, die der User anschließend abrufen kann. Beispielsweise erzeugt das Produkt *Rastertiles* eine Ordnerhierarchie mit mehreren RGB-Bildern.

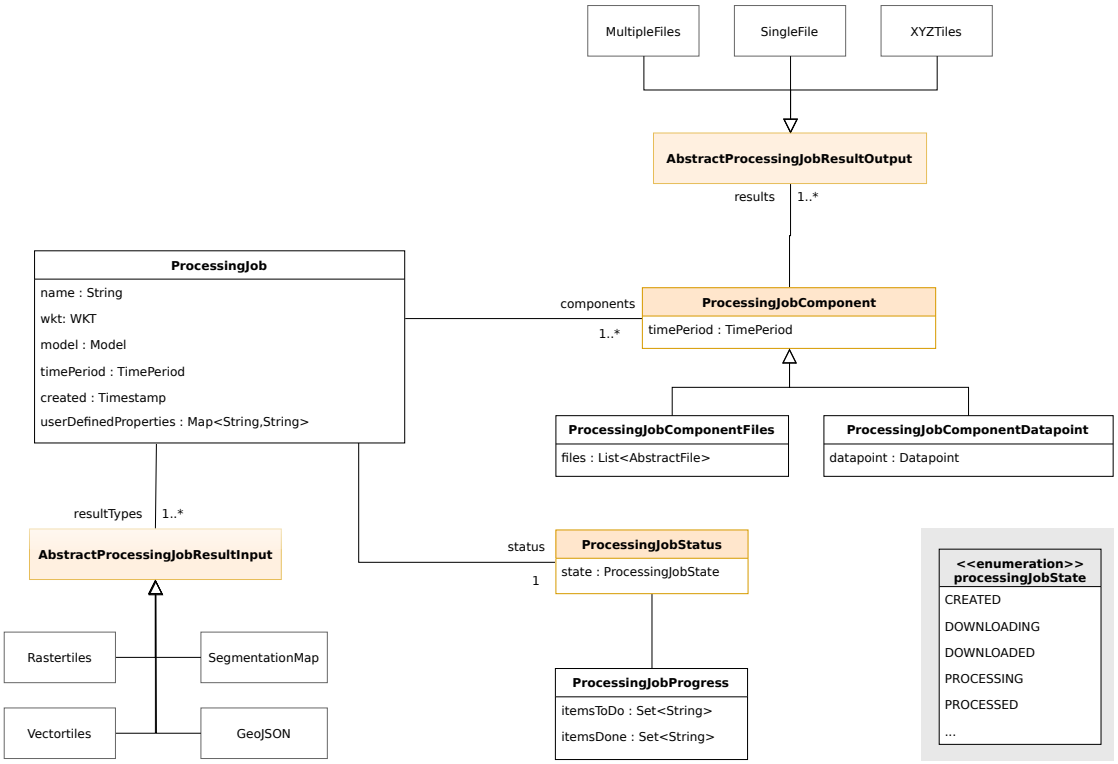


Abbildung 7.11: Datenmodell der ProcessingJob-Komponente. Ein *ProcessingJob* besteht aus mehreren *ProcessingJobComponent*s. Für jede *ProcessingJobComponent* werden alle Ergebnisdateien erzeugt, die der User durch Angabe der *resultTypes* gefordert hat.

Der in Abbildung 7.11 dargestellte Zusammenhang soll an einem Beispiel veranschaulicht werden:

Ein User erstellt einen *ProcessingJob* mit den Typen (genauer *AbstractProcessingJobResultInput*s) *Rastertiles* und *GeoJSON* und zwei *Datapoint*s. Das System erstellt für jeden Datapoint (genauer *ProcessingJobComponentDatapoint*) jeweils eine Ordnerstruktur *MultipleFiles* mit den für den Typ *Rastertiles* erwarteten Kacheln (PNG Bilddateien). Außerdem eine Datei *SingleFile* im GeoJSON Format, welches der Typ *GeoJSON* vorschreibt. Die erstellten Dateien können nun vom User heruntergeladen werden, der *ProcessingJob* befindet sich im Zustand *FINISHED*.

7.3.2.4 Place-Komponente

Die Place-Komponente stellt einen Dienst zum Suchen und Speichern von Polygonen auf einer Karte bereit. Der User kann über einen Suchbegriff, bspw. den Namen eines Stadtteils oder einer Gemeinde den Umriss des gesuchten Gebiets abfragen. Dieser kann anschließend für eine Anfrage an die Datapoint-Komponente, als Wert für das Datapoint-Suchgebiet verwendet werden. Der User muss so selbst keine Koordinaten angeben.

Die Komponente hat keine Abhängigkeiten zu anderen Systemkomponenten und dient nur dem Komfort des Users.

7.3.3 Authentifizierung und Authorisierung

Im System gibt es eine zentrale Stelle, die als Komponente außerhalb der API, für Identitätsmanagement zuständig ist. Dazu zählen Operationen wie Benutzer registrieren, Passwörter zurücksetzen oder Mail-Adressen überprüfen. Außerdem vergibt sie nach erfolgreicher Benutzerauthentifizierung zeitlich begrenzte Tokens für die Nutzung der API Ressourcen. Sie legt für jeden Client fest, welche Schritte für den Vorgang der Authentifizierung und Authorisierung notwendig sind. Die Abbildung 7.12 erläutert den Vorgang unter Verwendung der WebApp.

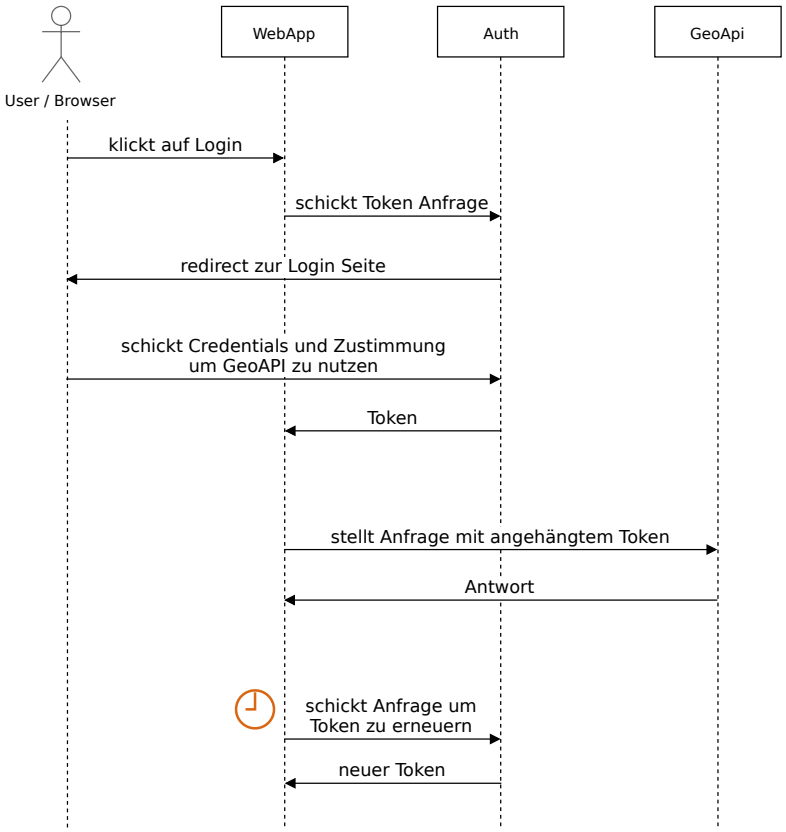


Abbildung 7.12: Login Vorgang für einen User der WebApp. Die Credentials des Users werden von der Auth Komponente geprüft und gegen zeitlich begrenze Tokens zur Nutzung der API eingetauscht.

Alle HTTP-Endpunkte der API erlauben Anfragen nur mit gültigem Token. Ein Token ist gültig wenn es von der Auth Komponente ausgestellt wurde und noch nicht abgelaufen ist. Ein Token kodiert unter anderem eine eindeutige User ID die innerhalb der API als Identifikation eines Users verwendet wird.

Dieser Aufbau hat mehrere Vorteile: Die API muss selbst keine Identitäten verwalten, dies ist Aufgabe der Auth Komponente. Um zu überprüfen ob ein API Zugriff authorisiert ist, muss die API selbst keinen State der Verbindung (z.B. Session Cookies) vorhalten, sondern nur das der Anfrage angehängte Token validieren. Außerdem können Anfragen dadurch an einen Unterdienst weiterdelegiert werden, solange auch dieser die Validität des Tokens überprüft.

Ist der API Zugriff erlaubt und ein User durch den Token eindeutig identifiziert, wird durch die ResourcePermissions Komponente innerhalb der GeoAPI festgestellt,

ob der User authorisiert ist, die angefragte Ressource (z.B. *DatapointProvider*) zu nutzen.

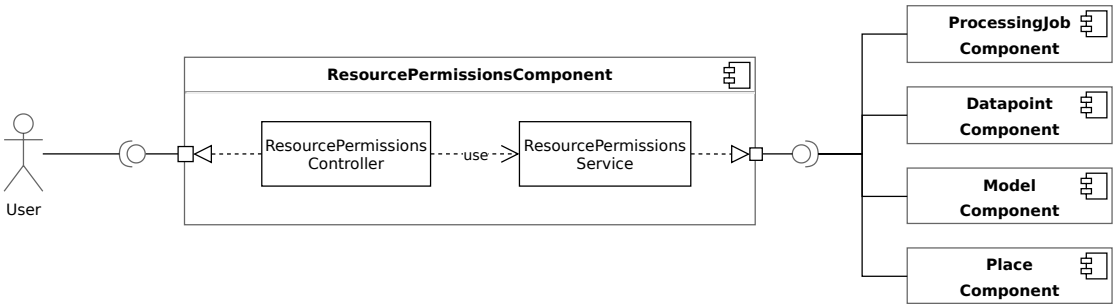


Abbildung 7.13: Alle Komponenten der API die eine *Resource* zur Verfügung stellen nutzen die ResourcePermissions Komponente zur Identifikation und Authorisierung einer Anfrage.

Damit auch externe User auf Ergebnisse einer Analyse zugreifen können, kann für die Ressource ein Geheimnis erstellt und in Form einer URL geteilt werden. Dieses Feature kann aber auch für interne User hilfreich sein, falls das GIS Programm oder das Shell-Skript keine Möglichkeit zum Token-Handling anbieten.

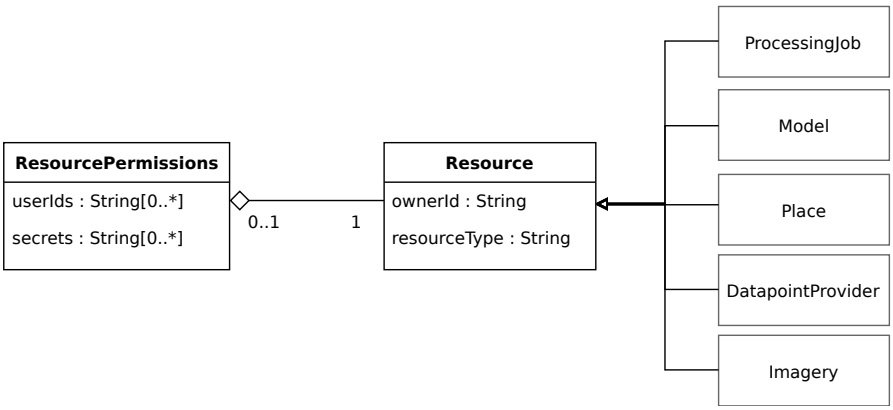


Abbildung 7.14: Datenmodell der ResourcePermissions Komponente. Zugriff auf ein *Resource* ist nur erlaubt, wenn **a)** der Zugriff vom Eigentümer stammt oder **b)** der zugreifende User authentifiziert und in den *ResourcePermissions* eingetragen ist. **c)** Hat sich der User nicht authentisiert muss er ein gültiges Geheimnis vorweisen können.

Die Auth Komponente implementiert das OpenID Connect Protokoll[9] welches auf das Authorization Framework OAuth 2.0[10] aufsetzt. Tokens sind vom Typ JSON Web Tokens[11]. Die oben dargestellte Login-Prozess entspricht dem Implicit Grant Flow[12] definiert im OAuth 2.0 Standard.

Die Funktionalität der Auth Komponente wurde nicht selbst implementiert, sondern durch eine Konfiguration des AWS Dienstes Amazon Cognito umgesetzt.

7.3.4 Deployment in der Cloud-Umgebung

Die folgende Sektion erläutert die Ausführungsumgebung des Gesamtsystems. Es werden die externen Schnittstellen und Abhängigkeiten zu AWS Diensten genannt, die für einen Betrieb des Systems notwendig sind. Dabei soll auch gezeigt werden, welche Maßnahmen ergriffen wurden um die Performance zu verbessern und die Kosten zu senken.

Zur besseren Übersichtlichkeit wird auf eine Beschreibung der Symbolik in den nachfolgenden Grafiken verzichtet und auf Abbildung 7.15 verwiesen. Die Symbole repräsentieren AWS Dienste, deren Aufgabe und Verwendung an entsprechender Stelle im Text erläutert wird.

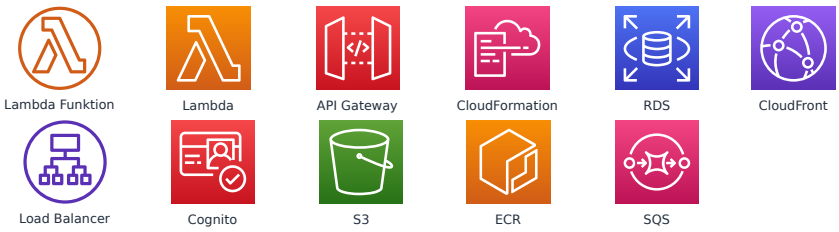


Abbildung 7.15: Alle im folgenden verwendeten AWS Symbole mit einer Beschreibung die im Fließtext verwendet wird. Rechteckige Symbole bezeichnen den Dienst, runde eine Instanz des Dienstes.

7.3.4.1 Übersicht

Das System verwendet separate Subdomains für den Zugriff auf die API, die Kommunikation mit der Auth Komponente und für das Deployment der WebApp. Wie in Abbildung 7.16 dargestellt, ist die API in zwei Einheiten geteilt, die im folgenden –

[9] https://openid.net/connect/
[10] https://oauth.net/2/
[11] https://tools.ietf.org/html/rfc7519
[12] https://tools.ietf.org/html/rfc6749#page-31

wo eine Unterscheidung sinnvoll erscheint – als Geo API und Result API bezeichnet werden.

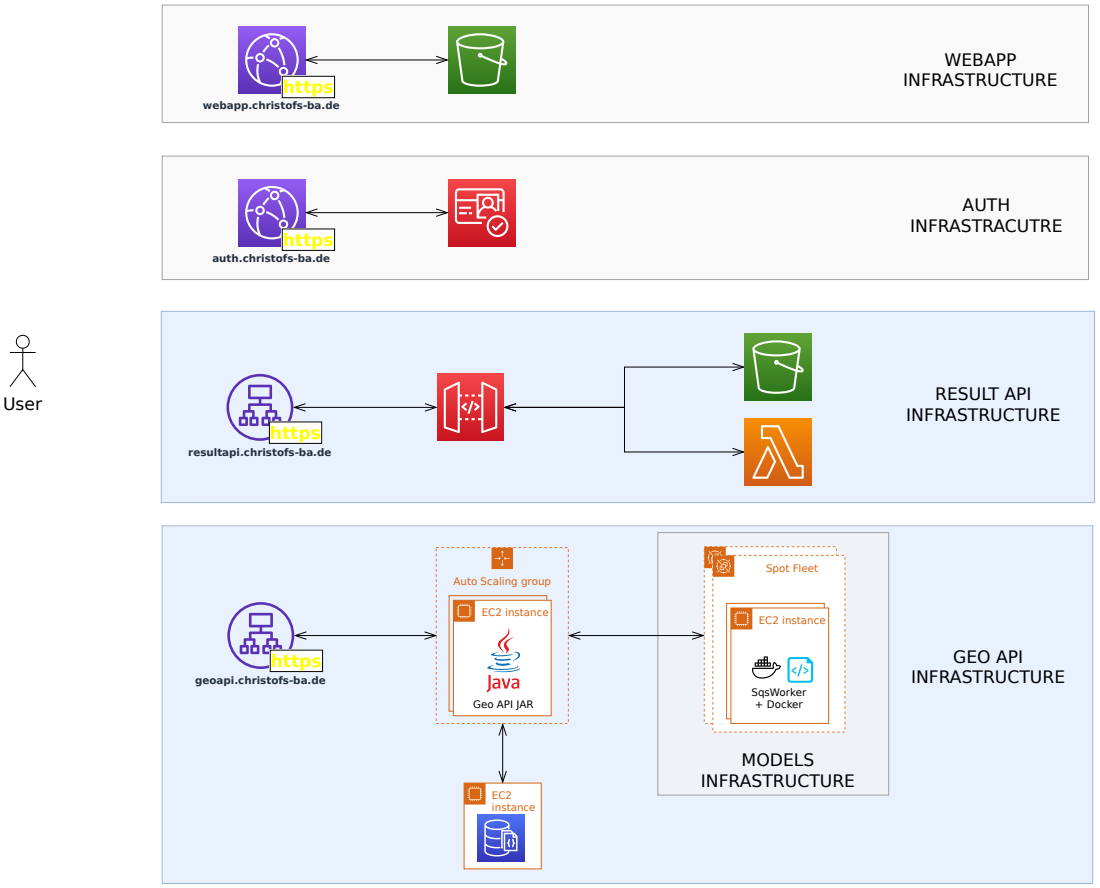


Abbildung 7.16: Deployment Übersicht des Gesamtsystems. Die blau unterlegten Flächen beinhalten Komponenten, die für die bisher als API bezeichnete Funktionalität zuständig sind. Der Zugriff des Useres auf das System erfolgt immer über HTTPS.

Die WebApp ist eine Single Page Application (SPA), daher sind im Backend nur einige statische Dateien (index.html, Javascript Dateien, Stylesheets, sowie Fonts und Bilder) vorzuhalten und über HTTPS auszuliefern. Die dafür notwendige, im Bereich *WebApp Infrastructure* der Abbildung 7.16 dargestellte Infrastruktur, ist mit einer CloudFront Distribution und einem dahinterliegenden S3 Bucket überschaubar.

S3 ist ein einfacher Datenspeicher indem die statischen Dateien in einem S3 Bucket (ermöglicht Gruppierung von Objekten) gespeichert sind. Der Dienst kann Dateien extern auch über HTTPS bereitstellen. Das System schaltet allerdings mit Cloud-

Front einen weiteren Dienst vor, um eine eigene Domain inklusive automatischem Zertifikatsaustausch nutzen zu können.

CloudFront ist der Content Delivery Network (CDN) Service von AWS. Ein CDN ist ein geographisch verteiltes System von Proxyservern und wird verwendet um global oder kontinental geringe Antwortzeiten zu erreichen, auch wenn der vom CDN auszuliefernde Inhalt buchstäblich am anderen Ende der Welt erstellt wird. Diese Funktion wird für das System derzeit nicht benötigt und nicht verwendet.

Wie in 7.3.3 bereits genannt wird die Funktionalität der Auth Komponente durch den Dienst Cognito implementiert. Dieser stellt die notwendigen Endpunkte zum Tokenaustausch bereit und hostet die Login-Seite.

Die Geo API liegt hinter einem Load Balancer der Anfragen an die, durch den EC2 Dienst bereitgestellten virtuelle Maschinen, verteilt. In einer EC2 Instanz verarbeitet ein Tomcat Server[13] die HTTP Anfragen. Zur Persistenz nutzt der Tomcat Server eine MongoDB[14], die in einer weiteren EC2 Instanz zu Anwendung kommt um eine separate Skalierung der Java-Instanzen zu ermöglichen.

Die Verarbeitung der Bilddaten findet nicht in der Java-Instanz statt, sondern ist ausgelagert (Bereich *Models Infrastructure* in Abbildung 7.16). Ebenso ist die Auslieferung der erzeugten Ergebnisartefakte aus Performance-Gründen durch andere Komponenten realisiert (Bereich *Result API Infrastructure*).

Die Zusammenhänge innerhalb der Result und Geo API im Zusammenspiel mit der Model Infrastruktur sind komplexer und werden nun im Detail erläutert.

7.3.4.2 Result API

Die Bereitstellung der Ergebnisartefakte ist eine IO-intensive Aufgabe wenn durch entsprechende Anfragen generierte, sehr große Dateien ausgeliefert werden müssen. Außerdem können Ergebnisse geteilt werden, wodurch Daten unter Umständen auch zeitgleich mehrfach bereitgestellt werden müssen. Gerade bei der Anzeige der Kacheln kommt es auf eine geringe Antwortzeiten an, damit das Scrollen der Karte flüssig wirkt. Aus diesem Grund werden alle Ergebnistypen vorgeneriert und in einem S3 Bucket abgelegt. Den Zugriff auf die Dateien ermöglicht ein AWS API Gateway. Dieser Dienst kann verschiedene AWS Komponenten integrieren und als eine HTTP API bereitstellen.

Das API Gateway nutzt einen geschützten HTTP Endpunkt der *Geo API* um von der ResourcePermissions Komponente prüfen zu lassen, ob eine Anfrage authorisiert

[13] Apache Tomcat implementiert einen HTTP Server für eine Java-Umgebung.
[14] MongoDB ist eine dokumentenorientierte NoSQL Datenbank.

ist. Nur nach erfolgreicher Prüfung werden die angeforderten S3 Objekte ausgeliefert.

Da der Zugriff auf Ergebnisse auch ohne Angabe eines Tokens möglich sein soll, gibt es jeden REST-Endpunkt doppelt. Zum Beispiel wird beim Zugriff auf */rastertiles* ein gültiges Token im Authorization Header der HTTP Anfrage erwartet, bei einem Aufruf von */s/rastertiles* wird dagegen die Authorisierung mit dem Request Parameter *secret* durchgeführt.

Das Aufrufen der Kacheln durch einen User stellt mehrere Anfragen für die gleiche *ProcessingJob* Ressource. Um die Autorisierungsabfrage nicht bei jeder Anfrage wiederholen zu müssen, cacht die *Result API* die Antwort für einige Minuten, um diese für weitere Anfragen des selben Users und der selben Ressource wiederverwenden zu können. Ein User wird dabei über das Token oder den Parameter *secret* identifiziert.

7.3.4.3 Geo API

Die bereits ausführlich beschriebene Funktionalität der Systemkomponenten wird zunächst durch eine Java-Anwendung umgesetzt. Diese erbringt die Leistung jedoch nicht allein, sondern delegiert verschiedene Aufgaben weiter. Dabei kommen die in Abbildung 7.17 dargestellten AWS Dienste zum Einsatz.

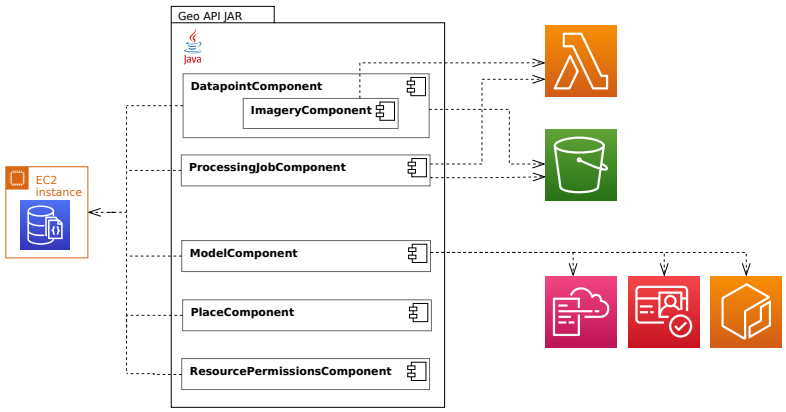


Abbildung 7.17: Schnittstellen der Geo API. Die ProcessingJob-Komponente lagert die Verarbeitung der Daten an AWS Lambda aus. Als Ablageort für Bilddateien wird ein S3 Bucket verwendet. Weitere Dienste sind zur Erstellung der Model Infrastruktur erforderlich und werden von der Model-Komponente angesprochen.

Die Model-Komponente ist zuständig für die Bereitstellung der Model Infrastruktur. Zur Erstellung, Änderung und Löschung aller für ein Model benötigter AWS Ressourcen verwendet die Model-Komponente den AWS Dienst CloudFormation.

CloudFormation gibt eine Entwurfssprache vor, in der alle benötigten AWS Komponenten detailliert beschrieben werden können um eine eigene Cloud-Umgebung zu definieren. Ein Dokument in dieser Sprache kann von CloudFormation verwendet werden um die beschriebene Umgebung bereitzustellen. Dabei werden die meisten Abhängigkeiten zwischen den Ressourcen automatisch aufgelöst.

Damit der User ein Docker Image hochladen kann, benötigt er Zugang zum entfernten Docker Repository. Dafür werden zwei weitere AWS Dienste benötigt. Mit der Identität des Users erfragt die Model-Komponente beim Cognito Dienst Credentials für den Zugriff auf den Dienst ECR an, der als private Registry das Docker Repository hostet. ECR wandelt die AWS Credentials in eine Username und Passwort Kombination um, die der User zur Anmeldung seines lokalen Docker Clients an die Registry benötigt. Dabei wird sichergestellt dass der User nur auf seine eigenen Repositories Zugriff erhält.

Die ProcessingJob-Komponente überträgt alle, von der Datapoint-Komponente lokal heruntergeladenen Dateien in einen S3 Bucket, den zentralen Datenspeicher des Systems. Die IO- und rechenintensive Vor- und Nachverarbeitung der Bilddateien erfolgt nicht innerhalb der Java-Instanz. Diese delegiert diese Aufgaben an Python-Funktionen die durch den Dienst Lambda jeweils als eine eigene Instanz gehostet werden.

Lambda Funktionen werden nur nach verbrauchter Rechenzeit abgerechnet. Benötigte Ressourcen wie CPU oder Arbeitsspeicher werden dabei von AWS verwaltet. Dadurch fallen im Gegensatz zu EC2 Instanzen keine Kosten an, wenn der Code nicht ausgeführt wird.

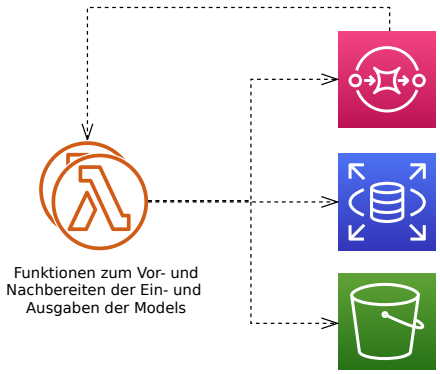


Abbildung 7.18: Die Lambda Funktionen verwenden Message Queues um mit den Models zu kommunizieren. Die eigentlichen Ein- und Ausgabedateien der Models werden in S3 abgespeichert und in den Nachrichten referenziert.

Die Lambda Funktionen stellen den Models die Daten aufbereitet zur Verfügung (in geforderter Größe, Auflösung, Format) und erstellen aus den Ausgaben der Models Ergebnisdateien die anschließend über die *Result API* abgerufen werden können.

Die Kommunikation zwischen den Lambda Funktionen der Geo API und den Models erfolgt gebuffert durch Message Queues die der AWS Dienst SQS bereitstellt. Dies entkoppelt die Komponenten und ermöglicht so u.a. Inputdaten für die Models zu erstellen ohne auf die Verarbeitung dieser warten zu müssen. Die in Abbildung 7.18 angedeutete beidseitige Kommunikation zwischen SQS und Lambda betrifft die Ein- und Ausgaben eines Models. Für jedes Model gibt es zwei Queues für Ein- und Ausgabe-Dateien und zwei Lambda Funktionen die die Eingabe-Dateien erzeugen und die Ausgabe-Dateien verarbeiten.

Bereits produzierte Ausgaben der Models sollen, wenn möglich in weiteren Anfragen genutzt werden, um sie nicht erneut erzeugen zu müssen. Um die Anzahl der Anfragen auf den S3 Storage zu reduzieren werden die Bilddateien und Model-Ausgaben in einer PostGIS Datenbank indiziert und von den Lambda Funktionen angesprochen. Das Hosting der Datenbank übernimmt dabei der AWS Dienst RDS.

7.3.4.4 Models

Der letzte Teil des Deployments behandelt die Infrastruktur eines Models die von der Model-Komponente mithilfe eines CloudFormation Templates eingerichtet wurde. Zur Ausführung eines Models werden drei Dienste benötigt: **1)** S3 zur Ablage

der Ein- und Ausgabedateien **2)** SQS zur Kommunikation mit den Lambda Funktionen und **3)** ECR, der als Container Registry das Image zur Ausführung in Docker zur Verfügung stellt.

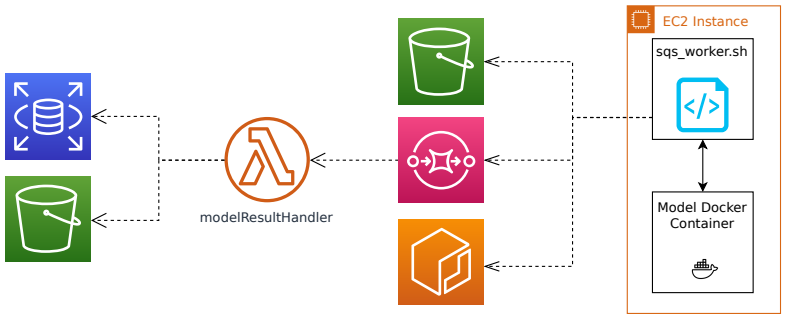


Abbildung 7.19: Der Docker Container wird innerhalb einer EC2 Instanz ausgeführt. Für die Ausführung des Containers und die Versorgung mit Daten ist ein Shell Skript zuständig.

Jedes Models besteht aus einer oder mehreren EC2 Instanzen. In diesen wird nach dem Bootvorgang ein Shell Skript gestartet, welches das Docker Image des Models herunterlädt. Das Skript nimmt Nachrichten aus der Input-Message-Queue des Models entgegen, lädt Eingabe-Dateien herunter, startet den Docker Container, überträgt die Ergebnisse an S3 und signalisiert die Verarbeitung über eine Nachricht in der Output-Message-Queue des Models.

Die in Abbildung 7.19 dargestellte Lambda Funktion wird bei jeder neuen Nachricht in der Output Queue aufgerufen. Diese meldet die Verarbeitung an die Geo API und aktualisiert ggf. die PostGIS Datenbank.

Der Hauptgrund für die Kommunikation via Message Queues ist die Skalierung der EC2 Instanzen eines Models. Das System ermittelt laufend eine Last Metrik für jede Input-Message-Queue. Diese ist definiert als

$$\textit{Rückstau pro Instanz} = \frac{\textit{Anzahl an Nachrichten in der Input-Message Queue}}{\textit{Anzahl EC2 Instanzen des Models}}$$

Weicht dieser Wert von einem Zielwert (20) ab, werden entsprechend weitere EC2 Instanzen gestartet oder vorhandene gestoppt und gelöscht (unter Beachtung der *instancesCount* Eigenschaft eines Models, die Minimal- und Maximalwerte vorgibt).

Weitere verwendete AWS Komponenten für die Netzwerkinfrastruktur wie Router, NAT-Gateways, VPC-Endpunkte oder Security Komponenten wie Rollen, Policies, Security-Groups sind in den CloudFormation Templates des Systems im Source Code einsehbar.

7.4 Anwendungsfälle des Systems

Im folgenden werden die interessantesten Anwendungsfälle des Systems erläutert. Dabei werden die Prozesse unter Verwendung der WebApp präsentiert. Da die API Anwendungsfälle funktional identisch sind, wird auf eine weitere Beschreibung dieser verzichtet. Eine detaillierte Beschreibung der API Funktionen kann in der WebApp im Menü *API* eingesehen werden.

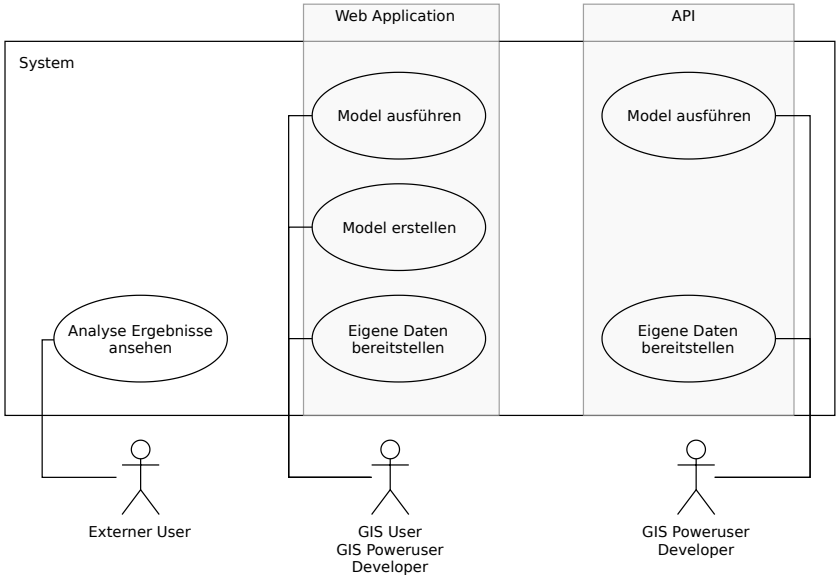


Abbildung 7.20: Anwendungsfall-Diagramm für das System. Die Fälle wurden in die Kategorien *Web Application* und *API* eingeteilt. Dies soll verdeutlichen, über welche Schnittstelle der User auf die Funktionalität des Systems zugreift. Der Anwendungsfall des externen Users ist keiner Kategorie zugeordnet, da das System für ihn transparent ist und er nicht bewusst die API verwendet.

7.4.1 Model ausführen

Dieser Anwendungsfall beschreibt die Analyse von Daten, die vom System bereitgestellt werden. Der User ist an den Ausgaben eines Models interessiert und möchte alle für sein Zielgebiet verfügbaren Satellitendaten einsehen und für ihn relevante Zeitpunkte auswählen. Das System soll die Daten selbstständig analysieren und alle Ergebnistypen erstellen, die der User zur weiteren Verarbeitung benötigt und zuvor ausgewählt hat.

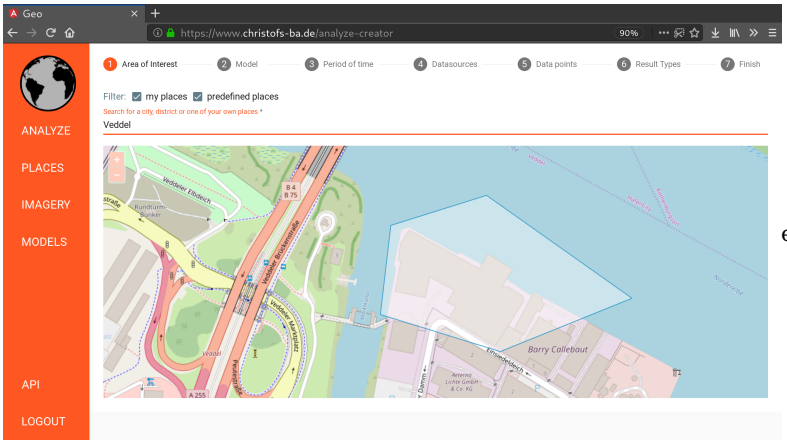


Abbildung 7.21: Der User befindet sich im Menü *ANALYZE* und gibt zunächst ein Suchgebiet ein. Die Fläche des Suchgebiets wurde zu einem früheren Zeitpunkt im Menü *PLACES* erstellt.

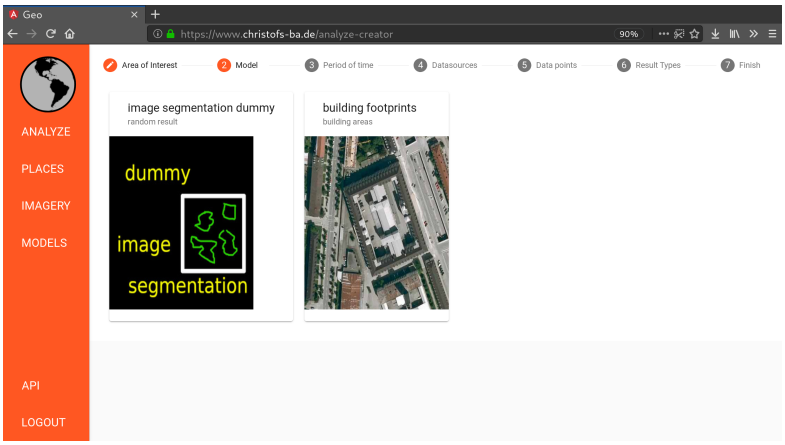


Abbildung 7.22: Der User wechselt in den nächsten Schritt und wählt das Model aus.

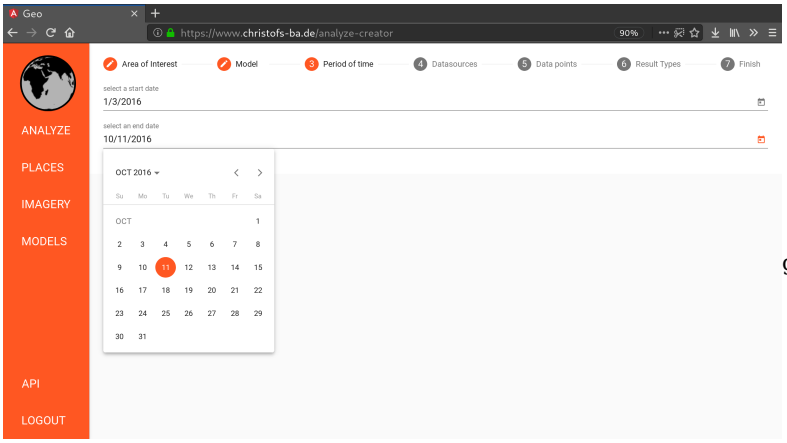


Abbildung 7.23: Der User gibt eine Zeitspanne vor, in der das System nach geeigneten Datapoints suchen soll.

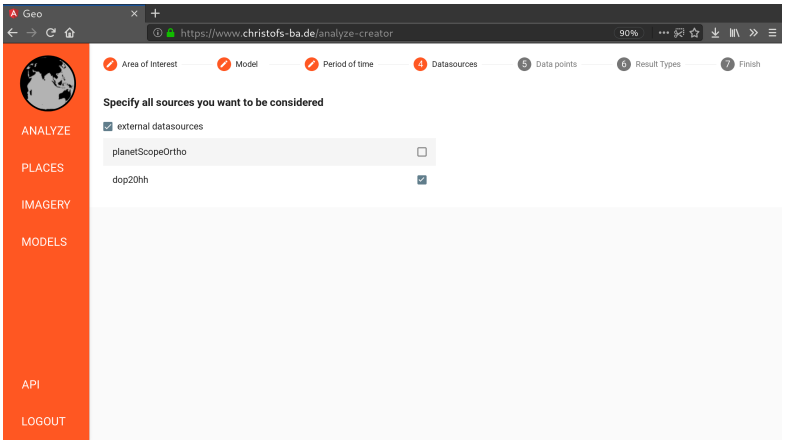


Abbildung 7.24: Das System ermittelt passende Datenquellen. Der User wählt eine Datenquelle aus.

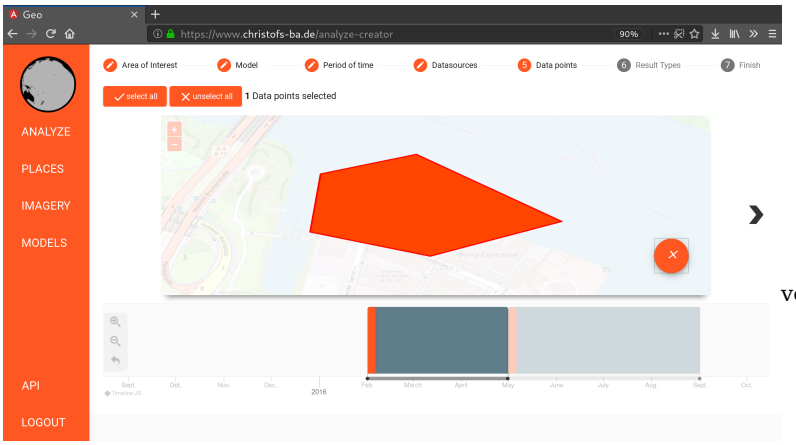


Abbildung 7.25: Der User wählt einen Datapoint aus. Der Anbieter der ausgewählten *dop20hh* Datenquelle befliegt die Fläche über einen Zeitraum von mehreren Monaten, daher sind die Datapoints entsprechend lang.

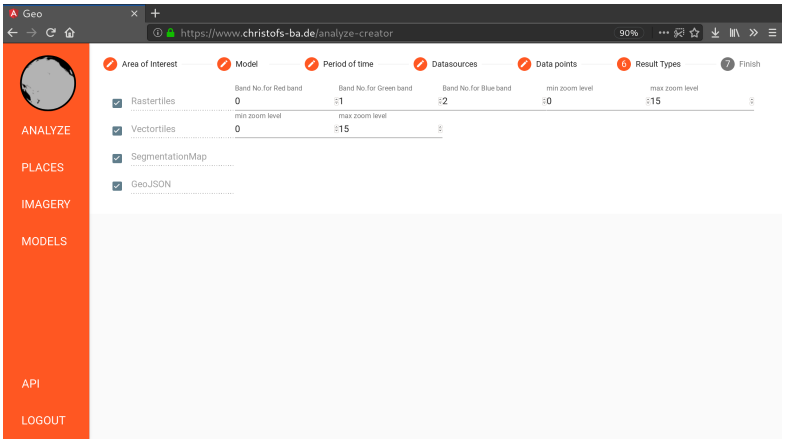


Abbildung 7.26: Der User entscheidet sich für die gewünschten Ergebnistypen. Rastertiles sind RGB Bilder und der User muss auswählen, welche Bänder auf die drei Farbkanäle abgebildet werden sollen.

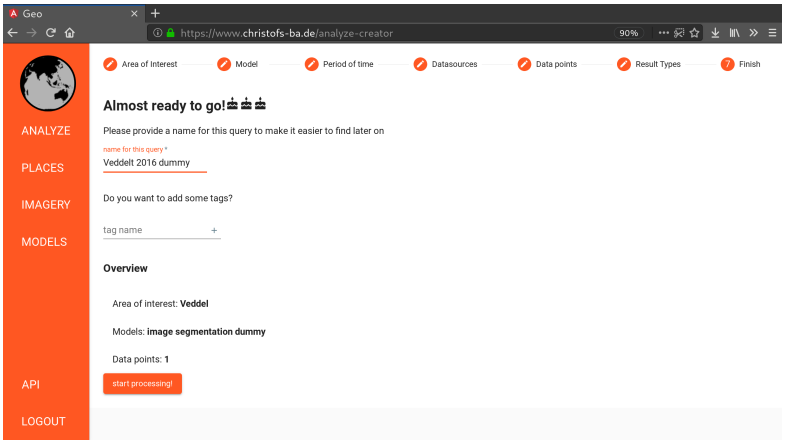


Abbildung 7.27: Nach der Angabe eines Namens für diesen ProcessingJob erstellt der User den Auftrag.

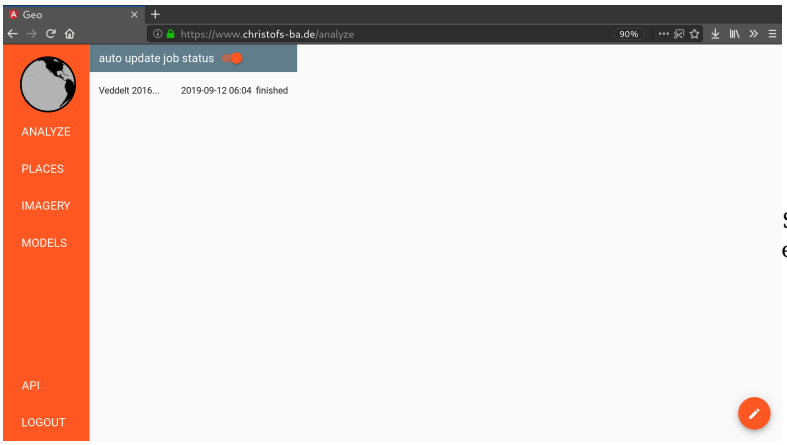


Abbildung 7.28: Der User wartet so lange bis das System die Ergebnis Produkte erstellt hat und der Status des ProcessingJobs als *finished* angezeigt wird.

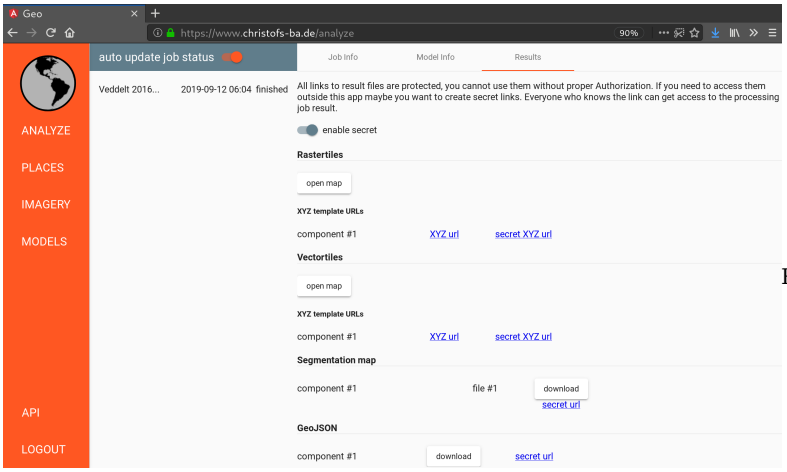


Abbildung 7.29: Nach einem Klick auf den ProcessingJob wechselt der User in den Reiter *Results* und verarbeitet die Daten weiter.

7.4.2 Model erstellen

In diesem Anwendungsfall möchte der User einen selbst implementierten Algorithmus, in einem Docker Image vorliegend, dem System zur Verfügung stellen. Das Ziel des Users ist es, eigene oder fremde Daten mit diesem Algorithmus zu analysieren und dabei von den Kapazitäten des Systems zu profitieren.

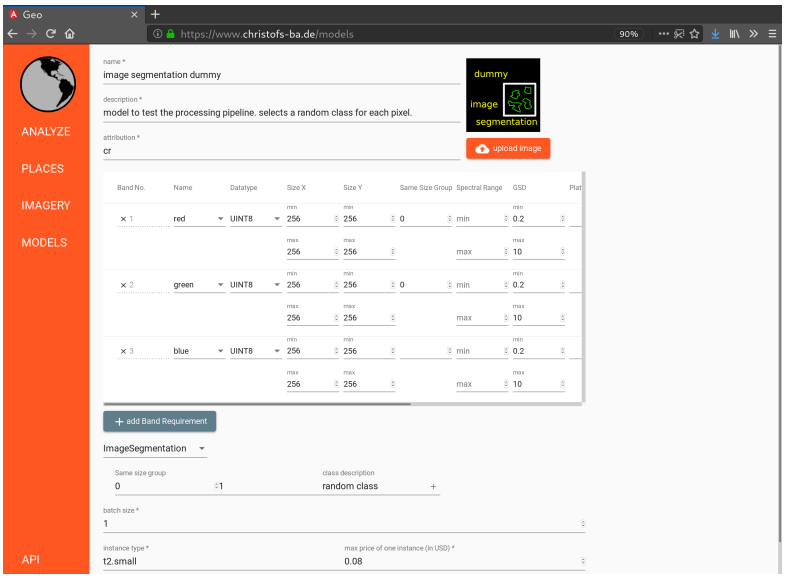


Abbildung 7.30: Der User befindet sich im Menü *MODELS* und füllt in der zunächst leeren Maske alle benötigten Daten für ein Model aus, darunter die Anforderungen der Rasterbänder und den Ausgabe-Typ.

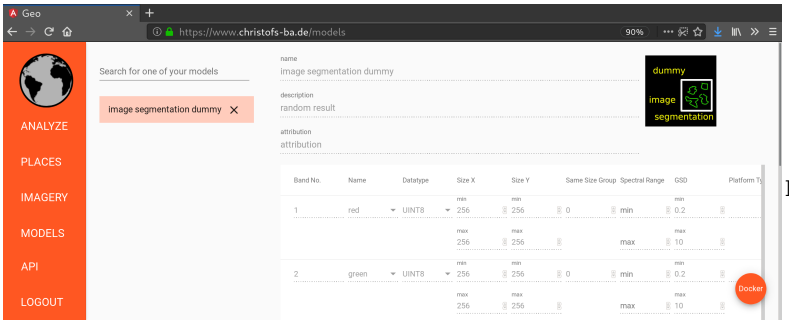


Abbildung 7.31: Der User wartet so lange, bis das System die benötigten Komponenten erstellt hat und klickt anschließend den Button *Docker*.

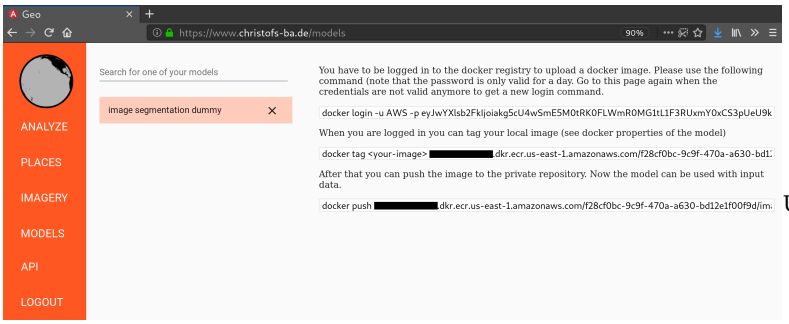


Abbildung 7.32: Das System präsentiert eine Liste von Konsolen-Befehlen die zum Upload des Images notwendig sind.

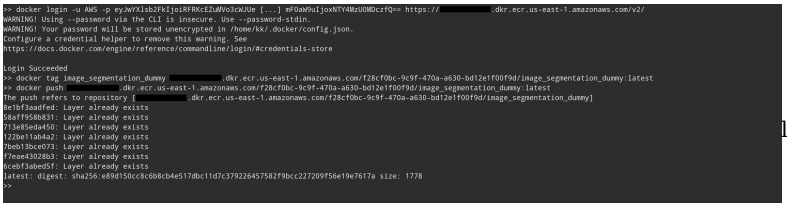


Abbildung 7.33: Der User führt die Befehle aus um sein lokales Image dem System zur Verfügung zu stellen.

7.4.3 Eigene Daten bereitstellen

Der vorliegende Anwendungsfall beschreibt die Bereitstellung von Bilddaten durch den User. Die Daten des Users stammen dabei möglicherweise von einem Dronenflug oder sie wurden bei einem Satellitenanbieter heruntergeladen. Das Ziel des Users ist es, die Daten dem System zur Verfügung zu stellen, um sie anschließend analysieren zu können (siehe Anwendungsfall 7.4.1). In diesem Anwendungsfall wird davon ausgegangen, dass der User dabei Ergebnistypen wie *Rastertiles* erstellen würde, die auch Metadaten zur Georeferenzierung voraussetzen.

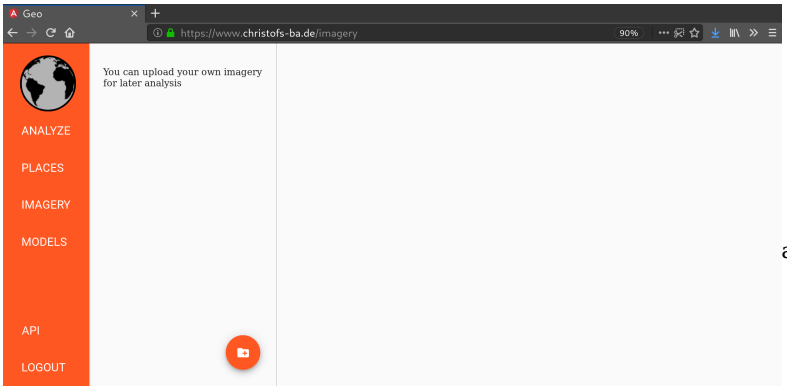


Abbildung 7.34: Der User befindet sich im Menü *IMAGERY*. Um eine neue Sammlung von Bildern anzulegen, klickt der User das Ordner-Symbol.

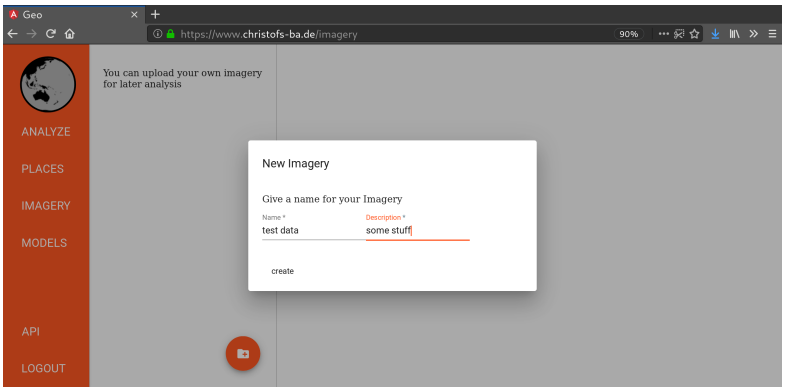


Abbildung 7.35: Ein Dialog erscheint, indem er einen Namen und eine Beschreibung der Daten angibt und bestätigt.

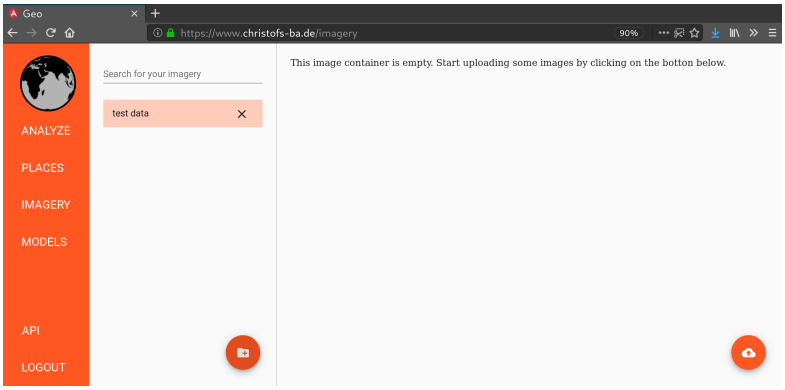


Abbildung 7.36: Um Bilddaten hochzuladen klickt der User auf das Download-Symbol und wählt eine lokale .tiff Datei aus.

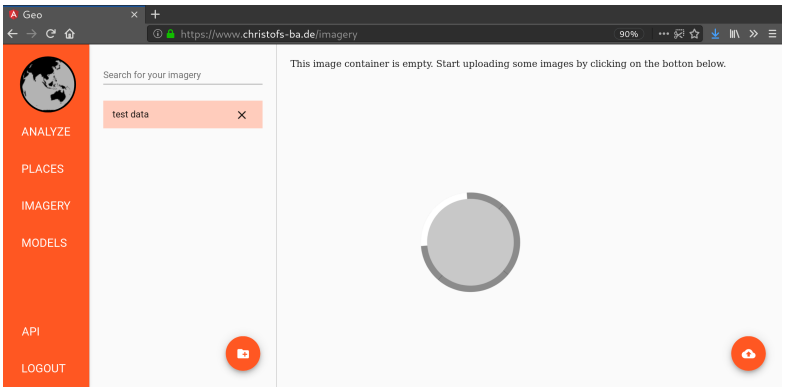


Abbildung 7.37: Der User wartet bis die Datei hochgeladen wurde.

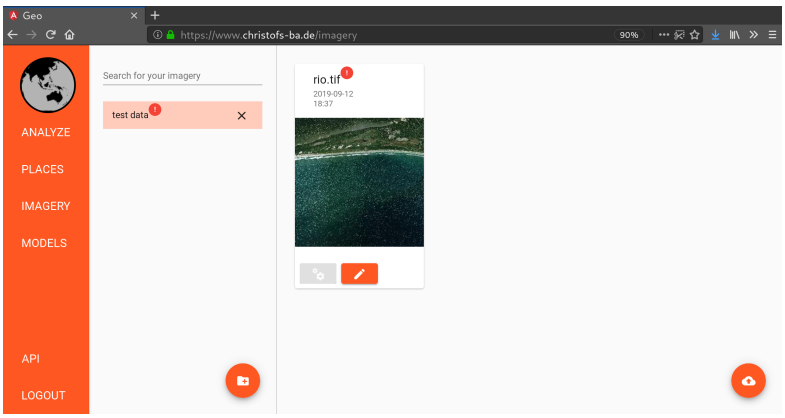


Abbildung 7.38: Der User hovert über das Vorschaubild der Aufnahme und klickt das Bearbeiten-Symbol um die Metadaten der Datei einzusehen.

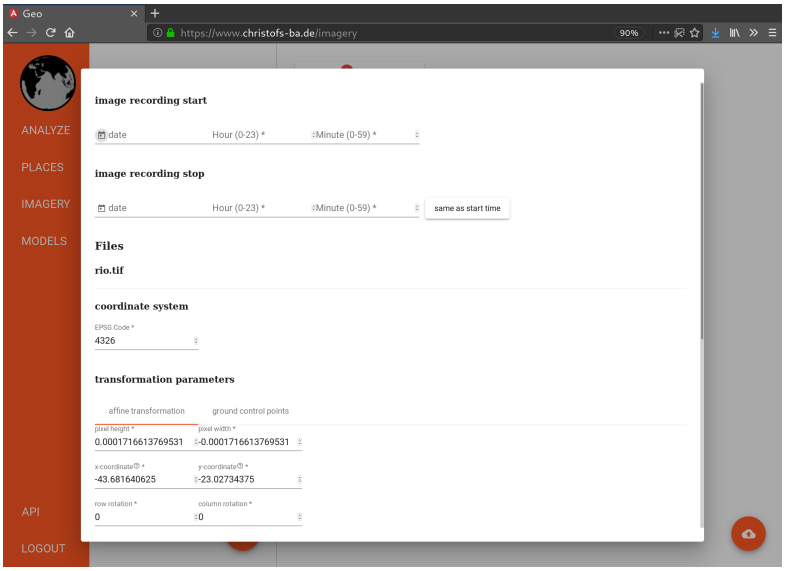


Abbildung 7.39: Das System konnte CRS und Informationen zur Georeferenzierung aus der Datei auslesen. Weitere Angaben zur Vervollständigung der Metadaten tätigt der User manuell.

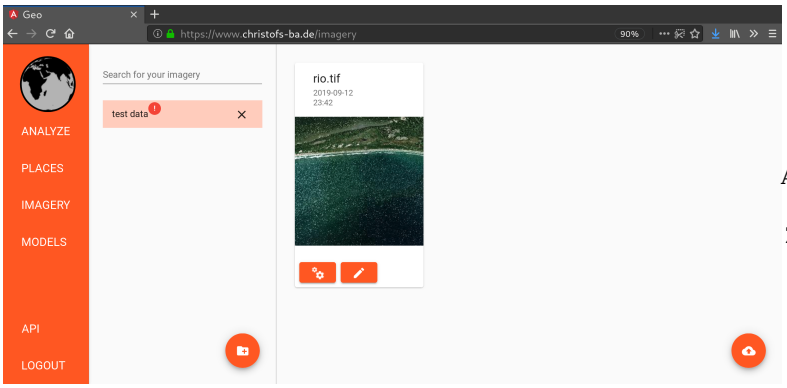


Abbildung 7.40: Sind die Angaben vollständig, kann der User über das Zahnrad-Symbol eine Analyse der Aufnahme durchführen.

7.4.4 Analyse-Ergebnisse ansehen

Die Umsetzung dieses Anwendungsfalls wurde aus zeitlichen Gründen nicht mehr abgeschlossen. Dieser sollte es ermöglichen, eine vom System generierte Website in einer externen Website (z.B. ein Blog) als HTML IFRAME einzubinden. Die Website sollte nur eine Karte präsentieren um die Ergebnistypen *Rastertiles* oder *Vectortiles* darstellen zu können. Die beiden genannten Ergebnistypen und die Möglichkeit eines externen Zugriffs wurden implementiert, die Generierung der Seite jedoch nicht mehr.

8 Entwicklung eines Models

Dieses Kapitel beschreibt die Entwicklung des Models *Building footprints*. Zunächst wird ein kurzer Einstieg in die Arbeitsweise der Machine Learning Community gegeben. Danach werden u.a. einige spannende Wettbewerbe und Datensätze beschrieben, die sich mit der Problemstellung des Models beschäftigen. Danach werden unterschiedliche Ansätze angeführt, die sich mit der Lösung der Problemstellung bereits beschäftigt haben und State-of-the-art Techniken erläutert. Das Kapitel schließt mit einer Übersicht der Experimente, die bei der Entwicklung des Models durchgeführt wurden.

8.1 Wettbewerbe und Datensätze

Um in der Machine Learning und Computer Vision Community Forschung in speziellen Gebieten anzureizen werden zeitlich befristete und thematisch begrenzte Wettbewerbe organisiert die im wissenschaftlichen Umfeld meist mit einer Konferenz enden. Dabei wird vom Veranstalter ein Datensatz und eine dazugehörige Aufgabe gestellt, die es zu lösen gilt. In der Regel müssen die Teilnehmer zusätzlich zur ihrem Ergebnis ein Paper einreichen, das den eigenen Ansatz beschreibt. Wird der Wettbewerb von einem Unternehmen abgehalten werden oft Preisgelder in Aussicht gestellt. In manchen Fällen müssen die Gewinner des Wettbewerbs ihre Lösung in Form von Source Code exklusiv dem Veranstalter zur Verfügung stellen.

Nicht selten finden solche Wettbewerbe auf Plattformen wie kaggle[1] oder topcoder[2] statt, die die nötige Infrastruktur zum Hosting der Daten, der Registrierung der Teilnehmer und zur Evaluation der (Zwischen)-Ergebnisse vorhalten und der Community eine Möglichkeit zum Austausch geben.

In viel beachteten Wettbewerben wie ImageNet oder COCO wurden oft Ansätze entwickelt, die auch bei der Lösung von Problemen in anderen Fachbereichen nützlich waren. Seit einigen Jahren werden auch in der Geo-Branche Wettbewerbe abgehalten und Datensätze entwickelt um neue Lösungen bei der Verarbeitung von Satelliten- und Luftbildern zu finden.

[1] https://www.kaggle.com/
[2] https://www.topcoder.com/

Aufgaben der Wettbewerbe lassen sich meist in eine der in Abbildung 8.1 genannten Kategorien einteilen.

Abbildung 8.1: Aufgabenkategorien im Bereich der Computer Vision. Von links nach rechts: Eingabebild, Labeling, Image Detection, Image Segmentation, Instance Segmentation

Reicht es au,s nur zu wissen ob sich eine gesuchte Klasse (in der Abbildung die Klasse *buildings*) im Bild befindet, liegt ein Labeling Problem vor. Muss zusätzlich erkannt werden an welcher Stelle im Bild sich ein Objekt befindet (beschrieben zum Beispiel durch eine Bounding Box) handelt es sich um eine Detection. Bei einer Image Segmentation muss für jeden Pixel eine Zugehörigkeit zu einer oder mehren Klassen gefunden werden. Müssen die einzelnen Instanzen einer Klasse voneinander unterscheidbar sein (Haus 1, Haus 2...) dann ist die Aufgabe eine Instance Segmentation.

Evaluationsmetriken Um die Ergebnisse der Teilnehmer bewerten und vergleichen zu können wird vom Veranstalter eine Evaluationsmetrik bestimmt. Als Bewertungsmaßstab für Aufgaben der Image Segmentation hat sich dabei die Intersection over Union (IoU) etabliert. Diese Kennzahl gibt die Ähnlichkeit von zwei Mengen an und entspricht dem Quotient der Größe der Schnittmenge und der Größe der Vereinigungsmenge. Die für andere Aufgaben verwendete Bewertung der Accuracy (Anzahl der korrekt klassifizierten Pixel an allen Pixeln) ist für das Problem der Image Segmentation nicht ausreichend, da die zu detektierenden Objekte im Vergleich zur Gesamtzahl der Pixel oft sehr klein sind. Abbildung 8.2 zeigt die Anwendung bei Bilddaten, wobei hier die Größe der Mengen durch die Anzahl der Pixel definiert ist.

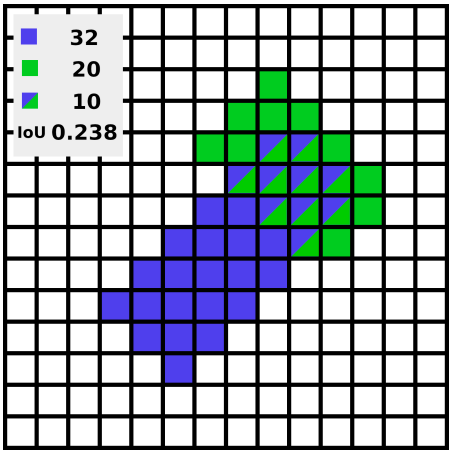


Abbildung 8.2: Anwendung der Intersection over Union. Blaue Pixel bilden die sogenannte *ground truth*, grüne Pixel die *prediction*. Die Schnittmenge ist durch Verwendung beider Farben gekennzeichnet.

Die Zahl der IoU hat einen Wertebereich von 0 bis 1, wo bei 1 eine perfekte Überlappung darstellt. Müssen einzelne Objekte unterschieden werden reicht die Angabe der IoU allein nicht aus. Ein Objekt gilt als erkannt, wenn die IoU der *ground truth* des Objekts und der *prediction* einen Schwellwert überschreitet (i.d.R. $0{,}5$). Die Ausgaben eines Algorithmus werden daran gemessen, wie viele Objekte erkannt werden (true positives, TP), wie viele nicht erkannt werden (true negatives, TN) und wie viele Objekte erkannt werden die keine sind (false positives, FN). Damit lassen sich zwei Qualitätsmetriken ermitteln:

Die Genauigkeit (precision) sagt aus wie viele gefundene Objekte relevant sind. Die Trefferquote (recall) gibt Auskunft wie viele relevante Objekte gefunden wurden. In Wettbewerben wird meist eine Kombination aus beiden Aussagen verwendet die beide Anteile gleich wichtet. Genauer wird der *F1 score* verwendet, welcher das harmonische Mittel von precision und recall angibt.

$$precision = \frac{|TP|}{|TP| \cup |FP|} \qquad recall = \frac{|TP|}{|TP| \cup |FN|} \qquad F1 = 2 * \frac{precision * recall}{precision + recall}$$

Die Werte werden je nach Wettbewerb entweder für jedes Bild einzeln, über den gesamten Datensatz oder für eine Gruppe von Bildern (z.B. der selben Region) ermittelt.

Tabelle 8.1 listet interessante Datensätze und Wettbewerbe auf. Ausgewählt wurden dafür Datensätze, die, mit denen im System implementieren Datenquellen vergleichbar sind[3] oder potentiell Testdaten für das, in der folgenden Sektion erläuterte, Model zur Erkennung von Building footprints liefern könnten.

Die Auswahl beschreibt nur einen kleinen Teil der Datensätze und Wettbewerbe die sich mit Satellitendaten beschäftigen, davon allerdings die interessantesten für Deep Learning Techniken, wenn man die Größe der Datensätze und die Komplexität der Aufgabenstellung in Betracht zieht. Für den Spezialfall der Gebäudeerkennung enthält die Aufzählung die derzeit relevantesten Datensätze des Fachgebiets.

[3] gleicher Satellit oder zumindest ähnliche Auflösung oder Anzahl an Bändern

Jahr	Name	Veranstalter	räuml. Auflösung	spektrale Auflösung	Quelle	Größe Datensatz	Kategorien	Bemerkungen
2012	Vaihingen Semantic Labeling Contest	ISPRS	0,09 m	RGB + NIR	Flugzeug 800 m, Stadt Vaihingen, Deutschland	33 Bilder	5 Kategorien, darunter buildings	zusätzliche Daten: Punktwolken + DEM
2013	Massachusetts Buildings Dataset	Volodymyr Mnih	0,66 m	RGB	Flugzeug, Stadt Massachusets, USA	151 Bilder	nur buildings	Pionierarbeit, Datensatz stammt aus Daten von OpenStreetMap + Open Data Orthophotos
2015	Potsdam Semantic Labeling Contest	ISPRS	0,05 m	RGB + NIR	Flugzeug, Stadt Potsdam, Deutschland	38 Bilder	5 Kategorien, darunter buildings	zusätzliche Daten: Punktwolken + DEM
2015	2015 IEEE GRSS Data Fusion Contest	IEEE GRSS	0,05 m	RGB	Flugzeug 300 m, Stadt Zeebruges, Belgien	7 Bilder	Keine	zusätzliche Daten: Punktwolken + DEM; offener Wettbewerb ohne Labels[4]
2016	SpaceNet Building Extractor Challenge Round 1	DigitalGlobe, CosmiQ Works, NVIDIA	0,45 m RGB (Pansharpened), 1,85 m multispektral	8 Bänder	World-View-2 Satellit, Stadt Rio de Janeiro, Brasilien	2544 km^2, 382534 Polygone	nur buildings	
2017	Dstl Satellite Imagery Feature Detection	Dstl[5]	0,3 m - 7,5 m	16 Bänder	WorldView 3 Satellit	450 Bilder	10 Kategorien, inklusive buildings	
2017	Functional Map of the World	IARPA [6]	RGB 0.3-0,6 m, 1.6-2,4 m 4 Bänder multispektral + 1.24-1,85 m 8 Bänder multispektral	RGB + NIR + 8 Bänder	Geo-Eye-1, QuickBird, World-View-2/3 satellites, globaler Datensatz, Stadt und Land	mehr als 1 Millionen Bilder	63 Kategorien, nur labels, keine Masken	ggf. auch interessant: AID Dataset [7], erstellt aus Google Earth Bildern
2017	Inria Aerial Image Labeling Challenge	Inria [8]	0,3 m	RGB	Flugzeug, 10 Städte mit niedriger und Hoher Dichte in USA und Österreich	810 km^2	nur buildings	Einreichungen auf Website weiterhin möglich

[4] Der IEEE GRSS Data Fusion Contest ist ein, seit 2006 jährlich abgehaltener Wettbewerb und beschäftigt sich explizit mit dem Bereich der Data Fusion.
[5] Defence Science and Technology Laboratory (Dstl) ist eine Organisation des britischen Verteidigungsministeriums
[6] Intelligence Advanced Research Projects Activity (IARPA) ist eine Organisation der der US Geheimdienste
[7] https://captain-whu.github.io/AID/
[8] Inria ist ein Akronym für das französisch nationale Forschungsinstitut für digitale Wissenschaften

2017	EuroSAT Dataset	TU Kaiserslautern und DFKI[9]	$10-60\,\mathrm{m}$ (Bänder $> 10\,\mathrm{m}$ wurden auf $10\,\mathrm{m}$ skaliert)	13 Bänder	Sentinel-2 Satellitenbilder 30 europäischer Städte	$11\,059\,\mathrm{km}^2$, 27000 Bilder	10 Kategorien; nur Labels, keine Masken	
2017	Understanding the Amazon from Space Challenge	Planet	$3{,}7\,\mathrm{m}$	RGB+NIR	PlanetScope Satellitenbilder des Amazonasgebiets	40000 Bilder	4 Klassen mit atmosphärischen Bedingen, 7 übliche land cover/land und 6 seltene land cover/land use Klassen	
2016	SpaceNet Building Extractor Challenge Round 2	DigitalGlobe, CosmiQ Works, NVIDIA	$0{,}3\,\mathrm{m}$ RGB (Pansharpened) + $1{,}24\,\mathrm{m}$ multispektral	RGB + 8 Bänder	World-View-3 Satellitenbilder mit Teilen von Las Vegas, Paris, Shanghai, Khartoum	$3011\,\mathrm{km}^2$, 302701 Polygone	nur buildings	bessere Qualität als Spacenet #1, da Labels durch weiteren Experten geprüft; Quelle: kein Mosaik sondern eine Aufnahme
2018	DIUx xView 2018 Detection Challenge	DIUx[10]	$0{,}3\,\mathrm{m}$ RGB (Pansharpened) + $1{,}24\,\mathrm{m}$ multispektral	RGB + 8 Bänder	World-View-3 Satellitenbilder mit globaler Abdeckung	1 Millionen Bilder, $1415\,\mathrm{km}^2$	60 Klassen, Bounding Boxes, mehrere pro Bild möglich	
2018	Ships in Satellite Imagery	Kaggle User rhammell	$3\,\mathrm{m}$	RGB	PlanetScope Satelllitendaten von Kalifornien, entnommen aus dem Planet Open California Datensatz	1000 Bilder von Schiffen, 3000 Bilder von Nicht-Schiffen	nur Schiff/Nicht-Schiff labels, keine Masken	bei Interesse: Hochauflösende Bilder von Schiffen in der Airbus Ship Detection Challenge [11]
2018	AIRS Dataset		$0{,}075\,\mathrm{m}$	RGB	Flugzeug, Christchurch, Neuseeland	$457\,\mathrm{km}^2$, 220000 Gebäude	Dach/Nicht-Dach Masken	
2018	DeepGlobe Building Detection Challenge	CVPR	$0{,}3\,\mathrm{m}$ RGB (Pansharpened) + $1{,}24\,\mathrm{m}$ multispektral	RGB + 8 Bänder	World-View-3 Satellitenbilder mit Teilen von Las Vegas, Paris, Shanghai, Khartoum	$3011\,\mathrm{km}^2$, 302701 Polygone	nur buildings	gleicher Datensatz wie SpaceNet Building Extractor Challenge Round 2
2018	SpaceNet nicht-nadir Building Detection Challenge	CosmiQ Works, Digital Globe and Radiant Solutions	$0{,}45\,\mathrm{m}$ RGB (Pansharpened), $1{,}85\,\mathrm{m}$ multispektral, bei größeren Winkeln PAN bis zu $1{,}6\,\mathrm{m}$	PAN + RGB + 8 Bänder	World-View-2 Satellitendaten mit Teilen von Atlanta, USA	$665\,\mathrm{km}^2$, 120000 Building footprints	building polygons	nicht-nadir Datensatz, nicht-nadir Winkel von 7 bis 54 Grad, nach Winkel eingeteilt in 3 Kategorien

[9] Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI)
[10] Defense Innovation Unit Experimental(DIUx) ist eine Organisation des US Verteidigungsministeriums
[11] https://www.kaggle.com/c/airbus-ship-detection

2019	xView2 Challenge	versch. US Behörden	$0{,}3\,\mathrm{m}$ RGB (Pansharpened) + $1{,}24\,\mathrm{m}$ multispectral	RGB + 8 Bänder	verschiedene Krisengebiet der Erde	$5000\,\mathrm{km}^2$, 700000 Building footprints	building Polygone jeweils mit Integer label das die Beschädigung des Gebäudes angibt	Datensatz speziell zur Schadensbewertung von Gebäuden und Change Detection nach einer Katastrophe (u.a. vertreten Vulkanausbruch, Landrutsch, Erdbeben und Waldbrände); noch nicht öffentlich, Daten werden im Herbst 2019 veröffentlicht

8.2 Model zur Erkennung von Building footprints

In der Recherche hat sich gezeigt, dass sich ein großer Teil der Literatur und der Wettbewerbe mit dem Thema der Erkennung von Building footprints beschäftigen. Des weiteren stehen mit denen in Tabelle 8.1 aufgezählten Datensätzen genügend Trainingsdaten zur Entwicklung eines eigenen Models bereit. Der folgende Abschnitt erläutert warum dies ein spannender Anwendungsfall ist.

8.2.1 Anwendungen

Eine automatisierte Erkennung von Gebäuden und deren Grundfläche (meist als Building footprint beschrieben) ist für viele Anwendungen hilfreich. Für viele Regionen gibt es gar keine oder nur veraltete Daten und eine Aktualisierung bzw. Erstellung ist durch manuellen Aufwand zu teuer und aufwendig. Mithilfe von künstlichen neuronalen Netzen konnten für Kanada OpenStreetMap Daten aktualisiert werden [12] oder innerhalb von nu drei Wochen eine aktuelle Datenbasis für Tanzania geschaffen werden [13]. Eine regelmäßig und automatisierte Erkennung von Building footprints ist hilfreich für die Erkennung von illegalen Bauten [14]. Eine automatisierte Erkennung von intakten und beschädigten Gebäuden kann eine Unterstützung sein, um nach einer Katastrophe innerhalb kurzer Zeit wichtige Entscheidungen treffen zu können wo und wie Hilfe geleistet werden soll. Auch für Versicherungen wären diese Informationen nützlich um beispielsweise nach einer Flut Hochrechnungen anstellen zu können oder um präventiv Gefahrstellen zu erkennen.

8.2.2 Ansätze

Um eine Segmentation Map, in der jeder Pixel einer Klasse zugeordnet ist erstellen zu können, müssen die Pixelwerte zunächst aufbereitet werden. Grundsätzlich lässt sich die Aufgabe dabei in zwei Bereiche trennen: Features extrahieren und anschließend klassifizieren.

Dem Problem der Erkennung von Gebäuden in Luft- und Satellitenbildern wittmeten sich bereits eine Vielzahl von Arbeiten auf unterschiedlicher Weise. Dem vorliegenden Problem widmeten sich bereits eine Vielzahl von Arbeiten auf unterschiedlicher Weise. Meist wurden dabei Features verwendet die spektrale oder geometrische

[12] https://gisuser.com/2019/03/microsoft-launches-building-footprints-in-canada/
[13] https://explore.maxar.com/Tanzania-Building-Footprints.html
[14] https://timesofindia.indiatimes.com/city/pune/satellite-imagery-identifies-illegal-buildings-in-pmrda/articleshow/69685749.cms

Informationen wie Form und Größe oder Texturinformationen des Bildes beschreiben. Auch kommen allgemein übliche Feature Descriptors wie HOG oder SIFT zum Einsatz.

[17] nutzt eine Vielzahl von generischen geometrischen Features (z.B. ermittelt durch Image Moments) um damit Orientierungen und Regionen zu beschreiben und eine SVM anzulernen die unter anderem Gebäude klassifiziert. [36] detektiert Hausdächer durch eine Erkennung und Filterung von Linien und anschließender Kombination um eine rechteckige Form zur erhalten, die einem Hausdach gleicht. Eine andere Arbeit [44] erkennt unter anderem durch Hough Transformation rechteckige, kreis- und S-förmige Gebäude. In einer Arbeit basierend auf Fuzzy Logic entwickelte [30] eine Lösung die das gerichtete räumliche Verhältnis von Gebäuden und deren Schatten modelliert. Die Segmentation Map wird durch Anwendung des GrabCut Algorithmus ermittelt. Auch [12] nutzt Charakteristiken der Gebäude-Schatten Kombination wie Form, Größe und Kontrast und kombiniert diese mit morphologischen Operationen. Dabei wird versucht Schatten von Nicht-Gebäuden wie Bäumen zu eliminieren. Diese Lösungen sind jedoch nur geeignet wenn Schatten im Bild sichtbar sind. Ein anderer Ansatz [1] basiert auf Active Contours und erstellt gute Umrisse, allerdings werden einige Gebäude, in Bereichen mit unzureichendem Kontrast mit dem Hintergrund, nicht erkannt. Eine spätere Arbeit [22] verbessert hier und nimmt weitere Komponenten im RGB und HSV Farbraum mit hinzu. [21] beschreibt einen Ansatz basierend auf higher-order Conditional Random Fields (CRF). Die higher-order CRF erstellt die Building Segmentation Maps basierend auf einem vorherigen Clustering-Schritt durch ein Gaussian Mixture Model und Vor-Segmentierung in vier Klassen (darunter auch Schatten). Die Methode funktioniert allerdings auch ohne das Vorhandensein von Schatten. Die Arbeit von [46] nutzt eine Reihe von Random Decision Trees zur Klassifikation der Pixel. Der Abstand eines Pixels zur Grenzlinie des Gebäudes wird hier explizit, als Regression Tree modelliert. [20] kombiniert Histogram of Gradients (HOG) und Local Binary Patterns (LBP), um mit diesen Features und einer SVM Rechtecke vorherzusagen. Diese Kandidaten werden mit weiteren spektralen Informationen verbessert und ein Kandidat für die Segmentation Map ausgewählt.

Für das System wird eine generische und skalierbare Lösung benötigt. Viele dieser Ansätze basieren auf Annahmen über Gebäuden, ihre Form, Farbe und Umfeld und wurden nur auf wenigen Daten getestet. Einige der Methoden benötigen keine Trainingsdaten. Allerdings gibt es durch die genannten Wettbewerbe und Datensätze mittlerweile genügend Material um eine andere Strategie zu verwenden, die im vorherigen Absatz noch nicht erwähnt wurde: Convolutional Neural Networks (CNN). Alle Teilnehmer in den oberen Plätzen der Challenges DeepGlobe Building Detection Challenge 2018 [27] oder Inria Building Detection Challenge [11] verwen-

den Varianten von CNNs. Auch in der Praxis mit Datenmengen von nationaler Größe hat sich die Technologie bewehrt [29][49].

Ein Convolutional Neural Network besteht üblicherweise aus mehreren Stufen. In jeder Stufe wird eine Menge von Feature Maps durch eine Anzahl von zu trainierenden Filtern mit einer Faltungsoperation und anschließend einer nicht-linearen Aktivierungsfunktion verarbeitet. Dadurch entstehen neue Feature Maps die durch eine Poolingoperation zunächst heruntergesampled und an die nächste Stufe weitergereicht werden. Die räumliche Auflösung eines Bildes, die in das CNN gegeben werden, verringert sich so mit jeder durchlaufenen Stufe, jedoch nimmt ebenso der Informationsgehalt der Feature Maps zu. Am Ende der letzten Stufe ist das Gesamtbild in wenigen high-level Features komprimiert. Diese sind, verglichen mit anderen Methoden und bei einer großen Anzahl an Testdaten, sehr robuste Features[15]. [45] nutzt diese um einen SVM Classifier zu trainieren. Die Umrisse extrahiert er durch ein Markov-Random-Field-basiertes Model (MRF). [50] beschreibt eine Lösung in der das CNN die Parameter der Energiefunktion eines Active Contour Models lernt.

Die vielbeachtete Arbeit [24] zeigte, dass ein reiner Full Convolutional Ansatz verwendet werden kann, um eine Segmentation Map zu erstellen. Üblicherweise die Feature Maps der letzten Stufe (hoher Informationsgehalt) mit denen der vorherigen Stufe (hohe Auflösung) kombiniert und nacheinander upgesampled bis das Ergebnis in der gleichen oder ähnlichen Auflösung wie das ursprüngliche Bild vorliegt. Die Einheit die den Input in high-level Featues überführt wird als Encoder bezeichnet. Der Decoder erstellt daraus, zusammen mit weiteren Feature Maps des Encoders, eine hochaufgelöste Segmentation Map.

Basierend auf dieser Architektur wurden viele Verbesserungen entwickelt. Viele Einreichungen bei Wettbewerben bauen auf dem U-Net [35] Model auf, welches sich durch eine einfache Architektur auszeichnet, schnell zu trainieren ist und speziell für die Anwendung bei wenigen Trainingsdaten optimiert wurde. [47] gewann durch ein Ensample aus U-Nets und zusätzlichem Input aus OpenStreetMap Layern die zweite SpaceNet Challenge. Der Gewinner der DeepGlobe Building Detection Challenge 2018 entwickelte eine Lösung mit einem geteilten ResNet-Encoder und vier Decodern zur separaten Erkennung von Straßen, kleinen, mittleren und großen Gebäuden. Der Zweig zur Erkennung der Straßen wurde durch ein vortrainiertes Netz angelernt. Dadurch wurden keine weiteren externen Daten benötigt. Um optisch ansprechendere Polygone zu erhalten greifen die Lösungen von [29] und [7] wieder auf Annahmen über Form und Größe zurück. Andere versuchen die Segmen-

[15] Empfehlung: Projekt Terrapattern nutzt die Features eines Bildes um damit ähnliche Objekte in der Umgebung zu finden. Dadurch können beliebige Objekte gefunden werden. http://www.terrapattern.com

tation Maps im Postprocessing mit klassischen Methoden wie Guided Filters [48] oder CRF[42] zu verbessern.

Bewertung In den aktuellen Wettbewerben und Arbeiten hat sich gezeigt, dass mittlere und große Gebäude in kontrastreichen Umgebungen zuverlässig extrahiert werden können. Die Qualität für andere Umgebungen scheint noch nicht ausreichend zu sein, um damit eine genaue Karte mit Umrissen erstellen zu können. Für andere Anwendungen von Building footprints, die keine so hohen Anforderungen im Bezug auf Form und genauer Position haben, ist der aktuelle Stand der Technik bereits jetzt ohne manuelle Kontrolle ausreichend um Dienste entwickeln zu können (siehe Kapitel 6). DigitalGlobe gibt auf ihrer Produktseite für Building footprints [16] an, die automatisch extrahierten Polygone manuell nachzukontrollieren, garantieren dann jedoch für eine globaler Abdeckung, einen false negative und false positive Anteil von maximal 5%.

8.2.3 Eigene Experimente

Basierend auf der U-Net Architektur wurden einige Experimente mit den Trainingsdaten der Inria Aerial Image Labeling Challenge durchgeführt. Dieser Datensatz wurden gewählt, da er von überschaubarer Größe ist und andere damit bereits erfolgreich U-Net Lösungen erarbeitet haben [11].

Ausgehend von einem einfachen Encoder (mehrere Blöcke mit Convolution und BatchNorm Layern mit Relu Activation) wurden auch andere Encoder basierend aus den vorderen Stufen vortrainierter (ImageNet classifier) ResNet50 und VGG16 Netze ausprobiert. [13] beschreibt bei einer ähnlichen Architektur Vorteile durch die Verwendung eines WideResnet-38 Netzes. Neben dem Jaccard loss (IoU) wurde auch der in [34] für eine Land Cover Classification eingesetzte Lovász-Softmax loss getestet. Wie im ursprünglichen Paper von U-Net vorgeschlagen, wurden Random Elastic Deformations und andere übliche Data Augmentation Techniken eingesetzt, um die Testdaten zu erweitern.

Dabei wurde keine systematische Auswertung durchgeführt und auf aufwändige Optimierungen verzichtet, da nur ein Testkandidat für das Model *Building footprints* benötigt wurde.

[16] Datenblatt Building footprints: https://dgv4-cms-production.s3.amazonaws.com/uploads/document/file/67/DG_Building_Footprints_DS_08-2018_.pdf

9 Ausblick

Dieses letzte Kapitel soll abschließend auf mögliche Änderungen eingehen die das System verbessern. Dabei werden einige Probleme der entwickelten Architektur genannt und Lösungen vorgeschlagen.

Ausführung der Models in einem Container Cluster

In der aktuellen Lösung wird jedes Model als eine eigenständige Gruppe von EC2 Instanzen realisiert und über die, in 7.3.4.4 beschriebene Metrik skaliert. Da die eigentliche Verarbeitung in einem Docker Container durchgeführt wird, wäre es naheliegend eine Lösung zu wählen, die Docker Container skalieren kann. Durch eine geteilte Infrastruktur der Model Container könnten so Ressourcen gespart und Zeit für Up- und Downscaling der EC2 Instanzen reduziert werden. AWS bietet mit ECS einen Dienst, der auf einem Cluster von EC2 Instanzen Docker Container verwalten kann. Damit können, unter Angabe eines Container Images und den benötigten Ressourcen wie CPU oder RAM, Tasks erstellt werden. Diese Tasks können anschließend auf dem Cluster zur Ausführung gebracht werden. Dieses Konzept würde damit auch der Semantik der ProcessingJob-Verarbeitung des Systems entsprechen. Als die Basis-Infrastruktur für das System festgelegt wurde, konnten einem Task keine GPU Ressourcen zugewiesen werden. Dadurch war es, ohne eine eigene Verwaltungslogik von Containern zu entwickeln, nicht möglich Models zu betreiben, die eine GPU zur Ausführung benötigen. Der Dienst bietet mittlerweile die Möglichkeit in einem Task GPU Ressourcen zu allokieren. Es wäre interessant zu evaluieren ob eine Lösung auf dieser Basis eine effizientere Ausführungsumgebung darstellt.

Integration eines Geocoding Services für die Place-Komponente

Mithilfe der Place-Komponente können eigene oder vom System bereitgestellte Polygone mit einem Suchbegriff gefunden werden. Zurzeit können nur statische Umrisse von Gemeinde- oder Stadtteilgrenzen geliefert werden, die zuvor manuell in

der Datenbank der Place-Komponente erfasst wurden[1]. Diese müssen manuell aktualisiert werden und es nicht praktikabel diese Informationen auch für andere Länder zu pflegen. Hier wäre es sinnvoll, einen externen Geocoding[2] Dienstleister zu verwenden. Oft wird von diesen aber nur die Möglichkeit angeboten, eine Adresse zu einem Punkt und einer Bounding Box aufzulösen um die Karte zu richtig positionieren. Für das System wären aber genauere Umrisse in Form von Polygonen wünschenswert. Beispielsweise könnte OpenStreetMap diese Daten zur Verfügung stellen.

Potentiale der Spatial Database nutzen

Alle Rasterdaten im System werden in einem S3 Bucket vorgehalten. Georeferenzierte Daten der Datapoint-Komponente werden auch in der PostGIS Datenbank erfasst, jedoch nur die Referenz auf das S3 Objekt. PostGIS bietet die Möglichkeit Rasterdaten auch direkt in der Datenbank zu speichern. Bei einer Evaluation mit Dateien verschiedener Größe wurde festgestellt, dass dieser Ansatz deutlich langsamer ist[3]. Es wurde aus Kostengründen nicht evaluiert ob eine leistungsstärkere Instanz zur Ausführung der Datenbank das Ergebnis verbessert hätte. Die benötigte CPU Zeit zur Datenaufbereitung in AWS Lambda ist deutlich günstiger als eine RDS Datenbankinstanz zu betreiben. Des weiteren kostet auch der RDS Datenbankspeicher im Vergleich mit der Speicherung der Daten in S3 deutlich mehr.

Die Datenbank könnte jedoch als wichtiges Werkzeug genutzt werden damit ein User die Ergebnisse der Verarbeitung schneller einsehen kann, noch während die Daten analysiert werden. Zurzeit werden die Ergebnistypen erst erstellt, wenn alle Eingabedaten durch das Model verarbeitet wurden. Um schnell einschätzen zu können ob die Analyse die gewünschten Ergebnisse liefert, sollte das System so früh wie möglich Zwischenergebnisse präsentieren können. Das System überwacht den Zustand der *ProcessingJob*s und weiß welche Daten bereits verarbeitet wurden. Die Ergebnisse werden fortlaufend in PostGIS aktualisiert, sobald neue Ausgaben durch das Model erzeugt wurden. Diese Information könnte genutzt werden um on-demand Kacheln mit PostGIS zu erstellen und diese in der WebApp anzuzeigen, bevor in AWS Lambda, nach Abschluss des *ProcessingJobs*, die finalen Ergebnistypen generiert werden.

[1] Zum Beispiel aus öffentlichen Quellen wie http://opendatalab.de/projects/geojson-utilities/

[2] (Forward) Geocoding bezeichnet die Auflösung einer Adresse zu Koordinaten.

[3] Der Zugriff auf die Daten erfolgte mit dem PostGIS Raster Treiber der Bibliothek GDAL https://gdal.org/drivers/raster/postgisraster.html

Erweiterung des Datenmodells um Change Detection Models zu ermöglichen

Zurzeit kann ein Model nur Daten einer Aufnahme (ein Zeitpunkt) erhalten. Hat ein User mehrere *Datapoint*s ausgewählt, muss er Veränderungen zwischen den Ergebnissen der *Datapoint*s selbst erfassen. In der WebApp ist dies nur optisch möglich. Arbeitet der User mit einem GIS Programm kann er die dortigen Möglichkeiten nutzen um Veränderungen zwischen den Vektordaten festzustellen.

Besser wäre es wenn das Datenmodell insofern angepasst wird, dass ein Model auch Daten von mehr als einer Aufnahme erhalten kann. Mit einer Erweiterung der zeitlichen Dimension wäre es möglich Models zu implementieren, die Veränderungen zwischen mehreren Zeitpunkten detektieren können.

Skalierbarkeit der API verbessern

Die rechen- und IO-intensiven Aufgaben der Daten Vor- und Nachverarbeitung, sowie die Analyse wurden bereits als eigenständige Einheiten ausgelagert, die den Anforderungen entsprechend skalierbar sind. Allerdings sind die Implementationen der *DatapointProvider* ein Teil der Java Instanz. Diese bietet die HTTP User-Schnittstellen der API an und sollte daher skaliert werden, sobald die Anzahl der Instanzen durch Überlastung nicht mehr ausreicht, um HTTP Anfragen innerhalb einer relativ kurzen Zeitspanne zu beantworten.

Es sollten jedoch andere Regeln für den Betrieb der *DatapointProvider* gelten, da die Anforderungen andere sind. Eine Implementation eines *DatapointProvider*s ist i.d.R. auf Daten einer externen Quelle angewiesen. Externe APIs begrenzen die Nutzung beispielsweise durch eine Beschränkung der Anzahl gleichzeitiger Anfragen oder der Menge an Anfragen innerhalb eines Zeitraums. Um diesen Anforderungen gerecht zu werden müsste jeder *DatapointProvider* separat Regeln für den Betrieb vorgeben und kann daher nicht Teil der Java Instanz sein.

Eine Lösung könnte ein eigenständiger Datenservice sein, der den Zustand und die Regeln der *DatapointProvider* kennt und geeignet reagieren kann[4]. Damit müsste jedoch auch das Persistieren der Daten im zentralen S3 Storage, durch diesen Datenservice erfolgen.

[4] Beispiele: **1)** zwei parallele Anfragen möglich, derzeit steht eine Antwort aus; **2)** die letzte Anfrage wurde abgelehnt und sollte in 5 Sekunden wiederholt werden; **3)** drei fehlgeschlagene Versuche, beende Anfrage mit einem Fehler.

Literaturverzeichnis

[1] Ahmadi, Salman ; Zoej, M.J. V. ; Ebadi, Hamid ; Moghaddam, Hamid A. ; Mohammadzadeh, Ali: Automatic urban building boundary extraction from high resolution aerial images using an innovative model of active contours. In: *International Journal of Applied Earth Observation and Geoinformation* 12 (2010), Juni, Nr. 3, S. 150–157. – URL https://doi.org/10.1016/j.jag.2010.02.001

[2] BlackSky: *Homepage*. – URL https://www.blacksky.com/. – Zugriffsdatum: 2019.09.17

[3] Bolstad, Paul: *GIS Fundamentals: A First Text on Geographic Information Systems*. 5. 2016. – ISBN 9781506695877

[4] Bridge, Space I.: *In brief: DigitalGlobe 80+ TB/day, Descartes Labs out of beta, Astro Digital Landmapper first images*. – URL https://www.spaceitbridge.com/in-brief-digitalglobe-80-tbday-descartes-labs-out-of-beta-astro-digital-landmapper-first-images.htm. – Zugriffsdatum: 2019.09.09

[5] Campbell, James B. ; Wynne, Randolph H.: *Introduction to Remote Sensing*. 5. Guilford Press, 6 2011. – ISBN 9781609181765

[6] Concerned Scientists, Union of: *UCS Satellite Database*. – URL https://www.ucsusa.org/nuclear-weapons/space-weapons/satellite-database. – Zugriffsdatum: 2019.09.09

[7] Dickenson, Matt ; Gueguen, Lionel: Rotated Rectangles for Symbolized Building Footprint Extraction. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2018, S. 215–2153

[8] DigitalGlobe: *Datasheet: Ecopia Building Footprints powered by DigitalGlobe*. Oktober 2018. – URL https://dgv4-cms-production.s3.amazonaws.com/uploads/document/file/67/DG_Building_Footprints_DS_08-2018_.pdf. – Zugriffsdatum: 2019.09.17

[9] DigitalGlobe, Unternehmen der Maxar Technologies G.: *Homepage*. – URL https://www.digitalglobe.com/. – Zugriffsdatum: 2019.09.17

[10] Google: *Google Cloud Vision AI*. – URL https://cloud.google.com/vision/. – Zugriffsdatum: 2019.09.17

[11] Huang, Bohao ; Lu, Kangkang ; Audeberr, Nicolas ; Khalel, Andrew ; Tarabalka, Yuliya ; Malof, Jordan ; Boulch, Alexandre ; Saux, Bertr L. ; Collins, Leslie ; Bradbury, Kyle ; Lefevre, Sebastien ; El-Saban, Motaz: Large-Scale Semantic Classification: Outcome of the First Year of Inria Aerial Image Labeling Benchmark. In: *IGARSS 2018 - 2018 IEEE International Geoscience and Remote Sensing Symposium*, IEEE, Juli 2018, S. 6947–6950. – URL https://doi.org/10.1109/igarss.2018.8518525

[12] Huang, Xin ; Zhang, Liangpei: A Multidirectional and Multiscale Morphological Index for Automatic Building Extraction from Multispectral GeoEye-1 Imagery. In: *Photogrammetric Engineering & Remote Sensing* 77 (2011), Juli, Nr. 7, S. 721–732. – URL https://doi.org/10.14358/pers.77.7.721

[13] Iglovikov, V. ; Seferbekov, S. ; Buslaev, A. ; Shvets, A.: TernausNetV2: Fully Convolutional Network for Instance Segmentation. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, June 2018, S. 228–2284

[14] Iliffe, Jonathan C. ; Lott, Roger: *Datums and Map Projections: For Remote Sensing, GIS and Surveying*. 2. Whittles Publishing, 4 2008. – ISBN 9781904445470

[15] Inc., Descartes L.: *Homepage*. – URL https://www.descarteslabs.com/. – Zugriffsdatum: 2019.09.17

[16] Inc., SpaceKnow: *Homepage*. – URL https://spaceknow.com/. – Zugriffsdatum: 2019.09.17

[17] Inglada, Jordi: Automatic recognition of man-made objects in high resolution optical remote sensing images by SVM classification of geometric image features. In: *ISPRS Journal of Photogrammetry and Remote Sensing* 62 (2007), August, Nr. 3, S. 236–248. – URL https://doi.org/10.1016/j.isprsjprs.2007.05.011

[18] Insight, Orbital: *Homepage*. – URL https://orbitalinsight.com/. – Zugriffsdatum: 2019.09.17

[19] Insight, Orbital: *Orbital Insight Energy: Oil Storage v5.1 Methodologies and Data Documentation*. November 2018. – URL https://www.cmegroup.com/market-data/files/orbital-insight-oil-storage-data-methodology-guide.pdf. – Zugriffsdatum: 2019.09.17

[20] Konstantinidis, Dimitrios ; Stathaki, Tania ; Argyriou, Vasileios ; Grammalidis, Nikolaos: Building Detection Using Enhanced HOG–LBP Features and Region Refinement Processes. In: *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 10 (2017), März, Nr. 3, S. 888–905. – URL https://doi.org/10.1109/jstars.2016.2602439

[21] Li, Er ; Femiani, John ; Xu, Shibiao ; Zhang, Xiaopeng ; Wonka, Peter: Robust Rooftop Extraction From Visible Band Images Using Higher Order CRF. In: *IEEE Transactions on Geoscience and Remote Sensing* 53 (2015), August, Nr. 8, S. 4483–4495. – URL https://doi.org/10.1109/tgrs.2015.2400462

[22] Liasis, Gregoris ; Stavrou, Stavros: Building extraction in satellite images using active contours and colour features. In: *International Journal of Remote Sensing* 37 (2016), Februar, Nr. 5, S. 1127–1153. – URL https://doi.org/10.1080/01431161.2016.1148283

[23] Lillesand, Thomas ; Kiefer, Ralph W. ; Chipman, Jonathan: *Remote Sensing and Image Interpretation*. 7. Wiley, 2 2015. – ISBN 9781118343289

[24] Long, J. ; Shelhamer, E. ; Darrell, T.: Fully convolutional networks for semantic segmentation. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015, S. 3431–3440

[25] Lundervold, Alexander S. ; Lundervold, Arvid: An overview of deep learning in medical imaging focusing on MRI. In: *Zeitschrift für Medizinische Physik* 29 (2019), Mai, Nr. 2, S. 102–127. – URL https://doi.org/10.1016/j.zemedi.2018.11.002

[26] Markus, Thorsten ; Neumann, Tom ; Martino, Anthony ; Abdalati, Waleed ; Brunt, Kelly ; Csatho, Beata ; Farrell, Sinead ; Fricker, Helen ; Gardner, Alex ; Harding, David ; Jasinski, Michael ; Kwok, Ron ; Magruder, Lori ; Lubin, Dan ; Luthcke, Scott ; Morison, James ; Nelson, Ross ; Neuenschwander, Amy ; Palm, Stephen ; Popescu, Sorin ; Shum, CK ; Schutz, Bob E. ; Smith, Benjamin ; Yang, Yuekui ; Zwally, Jay: The Ice, Cloud, and land Elevation Satellite-2 (ICESat-2): Science requirements, concept, and implementation. In: *Remote Sensing of Environment* 190 (2017), März, S. 260–273. – URL https://doi.org/10.1016/j.rse.2016.12.029

[27] Microsoft: *DeepGlobe Building Detection Challenge 2018*. – URL http://deepglobe.org/leaderboard.html. – Zugriffsdatum: 2019.09.15

[28] Microsoft: *Microsoft Cognitive Vision Service*. – URL https://azure.microsoft.com/en-us/services/cognitive-services/directory/vision/. – Zugriffsdatum: 2019.09.17

[29] Microsoft: *US Building Footprints*. – URL https://github.com/Microsoft/USBuildingFootprints. – Zugriffsdatum: 2019.09.15

[30] Ok, Ali O. ; Senaras, Caglar ; Yuksel, Baris: Automated Detection of Arbitrarily Shaped Buildings in Complex Environments From Monocular VHR Optical Satellite Imagery. In: *IEEE Transactions on Geoscience and Remote Sensing* 51 (2013), März, Nr. 3, S. 1701–1717. – URL https://doi.org/10.1109/tgrs.2012.2207123

[31] Outer Space Affairs, United Nations O. for: *Space Object Register*. – URL http://www.unoosa.org/oosa/osoindex/search-ng.jspx. – Zugriffsdatum: 2019.09.09

[32] PlanetWatchers: *Homepage*. – URL https://www.planetwatchers.com/. – Zugriffsdatum: 2019.09.17

[33] Programm, Copernicus: *Access to Data*. – URL https://www.copernicus.eu/en/access-data. – Zugriffsdatum: 2019.09.09

[34] Rakhlin, A. ; Davydow, A. ; Nikolenko, S.: Land Cover Classification from Satellite Imagery with U-Net and Lovász-Softmax Loss. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, June 2018, S. 257–2574

[35] Ronneberger, Olaf ; Fischer, Philipp ; Brox, Thomas: U-net: Convolutional networks for biomedical image segmentation. In: *International Conference on Medical image computing and computer-assisted intervention* Springer (Veranst.), URL https://doi.org/10.1007/978-3-319-24574-4_28, 2015, S. 234–241

[36] Saeedi, Parvaneh ; Zwick, Harold: Automatic building detection in aerial and satellite images. In: *2008 10th International Conference on Control, Automation, Robotics and Vision*, IEEE, Dezember 2008, S. 623–629. – URL https://doi.org/10.1109/icarcv.2008.4795590

[37] Satelligence: *Homepage*. – URL https://satelligence.com/. – Zugriffsdatum: 2019.09.17

[38] Satellite, Chang G.: *Chang Guang Satellite: About Us*. – URL https://www.cgsatellite.com/about-us/. – Zugriffsdatum: 2019.09.17

[39] Satellogic: *Satellogic Homepage*. – URL https://satellogic.com/. – Zugriffsdatum: 2019.09.17

[40] SE, Airbus: *Homepage*. – URL https://oneatlas.airbus.com/. – Zugriffsdatum: 2019.09.17

[41] Services, Amazon W.: *AWS Amazon Rekognition*. – URL https://aws.amazon.com/rekognition/. – Zugriffsdatum: 2019.09.17

[42] Shrestha, Sanjeevan ; Vanneschi, Leonardo: Improved Fully Convolutional Network with Conditional Random Fields for Building Extraction. In: *Remote Sensing* 10 (2018), Juli, Nr. 7, S. 1135. – URL https://doi.org/10.3390/rs10071135

[43] Stubenrauch, C. J. ; Rossow, W. B. ; Kinne, S. ; Ackerman, S. ; Cesana, G. ; Chepfer, H. ; Girolamo, L. D. ; Getzewich, B. ; Guignard, A. ; Heidinger, A. ; Maddux, B. C. ; Menzel, W. P. ; Minnis, P. ; Pearl, C. ; Platnick, S. ; Poulsen, C. ; Riedi, J. ; Sun-Mack, S. ; Walther, A. ; Winker, D. ; Zeng, S. ; Zhao, G.: Assessment of Global Cloud Datasets from Satellites: Project and Database Initiated by the GEWEX Radiation Panel. In: *Bulletin of the American Meteorological Society* 94 (2013), Juli, Nr. 7, S. 1031–1049. – URL https://doi.org/10.1175/bams-d-12-00117.1

[44] Turker, Mustafa ; Koc-San, Dilek: Building extraction from high-resolution optical spaceborne images using the integration of support vector machine (SVM) classification, Hough transformation and perceptual grouping. In: *International Journal of Applied Earth Observation and Geoinformation* 34 (2015), Februar, S. 58–69. – URL https://doi.org/10.1016/j.jag.2014.06.016

[45] Vakalopoulou, M. ; Karantzalos, K. ; Komodakis, N. ; Paragios, N.: Building detection in very high resolution multispectral data with deep learning features. In: *2015 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, IEEE, Juli 2015, S. 1873–1876. – URL https://doi.org/10.1109/igarss.2015.7326158

[46] wleite: *SpaceNet Challenge 1 - Building footprint extraction - implementation report by 1st place wleite*. 2017. – URL https://github.com/SpaceNetChallenge/BuildingDetectors/tree/master/wleite. – Zugriffsdatum: 2019.09.15

[47] XD_XD: *SpaceNet Challenge 2 - Building footprint extraction - implementation report by 1st place XD_XD*. – URL https://github.com/SpaceNetChallenge/BuildingDetectors_Round2/tree/master/1-XD_XD. – Zugriffsdatum: 2019.09.15

[48] Xu, Yongyang ; Wu, Liang ; Xie, Zhong ; Chen, Zhanlong: Building Extraction in Very High Resolution Remote Sensing Imagery Using Deep Learning and Guided Filters. In: *Remote Sensing* 10 (2018), Januar, Nr. 1, S. 144. – URL https://doi.org/10.3390/rs10010144

[49] Yang, Hsiuhan L. ; Yuan, Jiangye ; Lunga, Dalton ; Laverdiere, Melanie ; Rose, Amy ; Bhaduri, Budhendra: Building Extraction at Scale Using Convolutional Neural Network: Mapping of the United States. In: *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 11 (2018), August, Nr. 8, S. 2600–2614. – URL https://doi.org/10.1109/jstars.2018.2835377

[50] Zhang, L. ; Bai, M. ; Liao, R. ; Urtasun, R. ; Marcos, D. ; Tuia, D. ; Kellenberger, B.: Learning Deep Structured Active Contours End-to-End. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018, S. 8877–8885

[51] Zhao, Rui ; Yan, Ruqiang ; Chen, Zhenghua ; Mao, Kezhi ; Wang, Peng ; Gao, Robert X.: Deep learning and its applications to machine health monitoring. In: *Mechanical Systems and Signal Processing* 115 (2019), S. 213–237

A Anhang

Abbildung A.1: Blick auf die Alster in Hamburg. Daten von DOP20, LGV HH.

Abbildung A.2: Blick auf die Alster in Hamburg. Daten von PlanetScope Analytic Ortho, Planet. (hier ohne Farbkorrektur)

Abbildung A.3: Blick auf die Alster in Hamburg. Daten von Sentinel-2B, ESA.

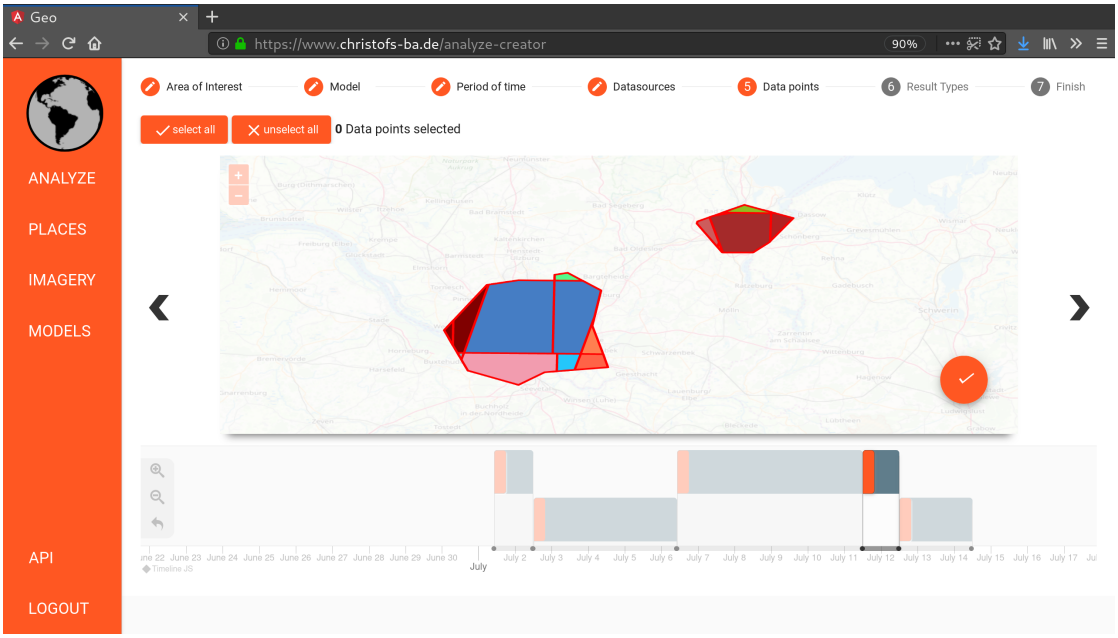


Abbildung A.4: Screenshot der Webanwendung bei Angabe der PlanetScope Datenquelle und einem mehrwöchigen Suchzeitraum für die Region Hamburg und Lübeck. Der markierte Datenpunkt setzt sich aus mehreren Aufnahmen zusammen, die innerhalb eines Tages entstanden sind.

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

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

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

Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: _____

Vorname: _____

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

Realisierung einer cloudbasierten Anwendung zur Auswertung von Satelliten- und Luftbildern mittels Deep Learning

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

_____	_____	_____
Ort	Datum	Unterschrift im Original