



UNIVERSITÄT  
LEIPZIG

Universität Leipzig  
Fakultät für Mathematik und Informatik

Erkennung von Anthracnose anhand  
von  
Sentinel-2-Multispektralaufnahmen

Abschlussarbeit zur Erlangung des akademischen Grades  
Master of Science (M.Sc.)

vorgelegt von

Simon Hüning

Referent:

Prof. Dr. Martin Middendorf

## **Zusammenfassung**

Hier kommt ein Abstrakt hin.

# Inhaltsverzeichnis

<b>1 Einführung</b>	<b>3</b>
<b>2 Methoden</b>	<b>4</b>
2.1 Trainingsdaten . . . . .	4
2.2 Normalized Difference Vegetation Index . . . . .	5
2.3 Sentinel-2 . . . . .	5
2.4 Das trainierbare Modell . . . . .	7
2.4.1 Anforderungen . . . . .	7
2.4.2 Grundlagen . . . . .	7
2.4.3 Mask R-CNN . . . . .	9
2.5 Evaluation des Modells . . . . .	14
2.5.1 Intersection over Union . . . . .	14
2.5.2 Precision und Recall . . . . .	14
2.5.3 Average Precision . . . . .	15
2.5.4 Mean Average Precision . . . . .	16
<b>3 Overfitting</b>	<b>18</b>
3.1 Begriffserklärung . . . . .	18
3.2 Data Augmentation . . . . .	19
3.3 L2 Regularization . . . . .	20
3.4 Zusätzliche Methoden . . . . .	21
<b>4 Konzept und Implementierung</b>	<b>22</b>
4.1 Konzept . . . . .	22
4.2 Annotation . . . . .	23
4.3 Suche nach Sentinelprodukten . . . . .	25
4.4 Aufbereitung der Sentineldaten . . . . .	27
4.5 Trainings- und Validierungsdatensatz . . . . .	29
4.6 Training/Detektion . . . . .	30
4.6.1 Dataset . . . . .	31

4.6.2 Config . . . . .	32
4.6.3 Ablauf . . . . .	34
<b>5 Experimente</b>	<b>40</b>
<b>6 Zusammenfassung und Fazit</b>	<b>45</b>
<b>7 Ausblick</b>	<b>46</b>
<b>Abbildungsverzeichnis</b>	<b>47</b>
<b>Tabellenverzeichnis</b>	<b>48</b>
<b>Literaturverzeichnis</b>	<b>49</b>
<b>Erklärung</b>	<b>52</b>

# **Kapitel 1**

## **Einführung**

# Kapitel 2

## Methoden

### 2.1 Trainingsdaten



Abbildung 2.1: RGB-Sentinel-2-Aufnahme des zu untersuchenden Ackers. Die infizierten Flächen sind weiß umrandet.

Der infizierte Acker, der die Basis des Datensatzes bildet, befindet sich etwa 15 km nordwestlich von Bologna in Norditalien ( $44^{\circ}34'28.92''$  Nord,  $11^{\circ}10'21.36''$  Ost). Das Feld hat eine Fläche von  $7640,57\text{ m}^2$  und ist mit Sorghum bepflanzt. Mitarbeiter des CREA (Council for Agricultural Research and Economics) haben vor Ort am 12.07.2018 Befälle von Anthracnose und der bakteriellen Streifenkrankheit (med.: *Xanthomonas translucens*) diagnostiziert. Etwa die Hälfte der Pflanzen des Feldes sind betroffen. Dabei sind im östlichen Teil des Feldes (Abb. 2.1, innere Markierung) auf einer Fläche von  $1043,22\text{ m}^2$  von etwa 60 bis 70% der Pflanzen befallen.

## 2.2 Normalized Difference Vegetation Index

Es gibt eine starke Korrelation zwischen dem physiologischen Status einer Pflanze und deren Chlorophyllgehalt. Faktoren wie Krankheit, Dürre oder Umweltverschmutzung haben einen negativen Einfluss auf den Chlorophyllspiegel.[16] Messungen haben ergeben, dass es eine Verbindung zwischen dem Reflexionsgrad im nahen Infrarotbereich und im Rotbereich und dem Chlorophyllgehalt gibt. Das heißt, dass eine gesunde, adulte Pflanze im nahen Infrarotbereich stärker reflektiert als zum Beispiel eine pathologisch veränderte Pflanze. Jedoch bleibt die Reflexion im roten Lichtspektrum in beiden Fällen vergleichsweise schwach. Andere vegetationsfreie Oberflächen wie Acker, Straßen oder Wasser strahlen auch im nahen Infrarotbereich schwach zurück. Dadurch ergibt sich eine zerstörungsfreie Methode, mit einer Multispektralkamera die Vitalität („Grünheit“) einer oder mehrerer Pflanzen zu bestimmen.[14]

Eine multispektralen Aufnahme kann mithilfe der Formel

$$NDVI = \frac{Band_{NIR} - Band_{Red}}{Band_{NIR} + Band_{Red}} \quad (2.1)$$

dazu genutzt werden, den *Normalized Difference Vegetation Index* (NDVI) zu berechnen. Wobei  $Band_{NIR}$  der nahe Infrarotbereich (Near Infrared) und  $Band_{RED}$  der sichtbare rote Bereich des elektromagnetischen Spektrums ist. Der NDVI gibt quantifizierte Werte im Bereich von  $-1$  bis  $1$  zurück. Dabei deuten Werte, die kleiner als  $0$  sind, auf Wasseroberflächen hin.  $0$  bedeutet keine Vegetation. Bei Werten nahe  $0$  handelt es sich um spärliche oder ungesunde Vegetation. Das bedeutet je näher ein Wert an  $1$  ist, desto dichter bewachsen und gesünder ist die beobachtete Vegetationsfläche.[22] Dass bei einem niedrigen, positiven NDVI nicht unterschieden werden kann, ob eine Fläche kaum bewachsen ist oder ungesunde Vegetation besitzt, kann hier vernachlässigt werden. Das Gebiet, das in dieser Arbeit untersucht wird, ist ein bewachsenes Feld, so kann man geringe Vegetation ausschließen.

## 2.3 Sentinel-2

Die Sentinel-2-Satelliten sind eine von sechs Satellitenarten (Sentinel-1 bis -6) des Copernicus-Programms<sup>1</sup>, die zur Erdbeobachtung in einen 786 km ho-

---

<sup>1</sup>Das Copernicus-Programm wurde von der Europäischen Union zur Erdbeobachtung ins Leben gerufen. Die gesammelten Daten werden für wissenschaftliche, wirtschaftliche und

hen sonnensynchronen Orbit gebracht wurden. Die Instrumente der Sentinel-2-Satelliten können Aufnahmen in Bereichen des roten und nahen Infrarots bis hin zum Kurzwelleninfrarotspektrums. Die Aufnahmen haben Gesamtgröße von  $100 * 100$  km und je nach Band eine von Auflösung von 10m, 20m oder 60m (s. Tabelle 2.1).

Bandnummer	Auflösung	Wellenlänge (nm)	Bandbreite (nm)
B1	60	443,9	27
B2	10	496,6	98
B3	10	560	45
B4	10	664,5	38
B5	20	703,9	19
B6	20	740,2	18
B7	20	782,5	28
B8	10	835,1	145
B8a	20	864,8	33
B9	60	945	26
B10	60	1373,5	75
B11	20	1613,7	143
B12	20	2202,4	242

Tabelle 2.1: Räumliche und spektrale Auflösungen von Sentinel-2A[12]

Besonders wichtig sind die Bänder B4 (Rot) und B8 (Nahe Infrarot). Mit diesen Bändern kann der NDVI (s. Kapitel 2.2) berechnet werden.[11] Die Sentinel-2-Satelliten bieten mit  $10 * 10$  m pro Pixel eine hohe räumliche Auflösung.<sup>2</sup> Diese Eigenschaft ist wichtig, um eine mögliche Infizierung genau eingrenzen zu können.

Dabei ist es auch wichtig, dass die Satelliten regelmäßige Daten liefern können. Durch die gemeinsame Konstellation übertragen die Plattformen alle fünf Tage Daten über einen spezifischen Punkt auf der Erdoberfläche.[13] Damit ist gewährleistet, dass der Feldbesitzer ohne persönliche Inspektion ein bis zweimal in der Woche eine Gesundheitseinschätzung über seine Felder erhält.

---

private Anwendungszwecke zur Verfügung gestellt.[9]

<sup>2</sup>Im Vergleich hat zum Beispiel der Landsat-8-Satellit, dessen Daten ebenfalls frei verfügbar sind, eine relativ geringe Auflösung von  $30 * 30$  m.[26]

## **2.4 Das trainierbare Modell**

In Kapitel 2.2 und 2.3 wurde erklärt wie Daten über die möglichen Erkrankungen geliefert und verarbeitet werden können. Auf den zugrunde liegenden Bilddaten soll nun ein künstliches neuronales Netzwerk (KNN) trainiert werden. In diesem Kapitel wird darauf eingegangen, welche Anforderungen an das KNN gestellt werden, warum das Titel gebende Netz ausgewählt wurde und wie dieses funktioniert.

### **2.4.1 Anforderungen**

Das KNN muss in der Lage sein, wahrscheinliche Krankheiten in der zu untersuchenden Agrarfläche möglichst genau eingrenzen und klassifizieren zu können. Das ist besonders wichtig, wenn ein Feld von multiplen Krankheiten betroffen ist.

Es ist damit zu rechnen, dass Daten unter bewölkten Bedingungen aufgenommen werden. Nach starken Niederschlägen können Acker teils oder gänzlich überflutet sein.[21] Das sorgt selbst unter wolkenfreien Bedingungen für einen niedrigen NDVI, obwohl die Nutzpflanzen gesund sind. Das neuronale Netz muss mit solchen „Ausreißern“ umgehen können.

Daraus ergeben sich folgende Kriterien für das neuronale Netzwerk:

- Erkennung auf Pixelebene
- Robustheit
- Hohe Genauigkeit

### **2.4.2 Grundlagen**

#### **Vollständig vernetztes neuronales Netz**

Künstliche neuronale Netze sind mathematische Modelle, die nach dem Vorbild von biologischen neuronalen Netzen gebildet worden sind. So ist ein KNN ebenfalls eine Verbindung von künstlichen Neuronen. Diese Neuronen sind in Schichten angeordnet und jede die Neuronen einer Schicht sind mit den Neuronen nächsten bzw. letzten Schicht verbunden. Zwischen der ersten und der

letzten sog. Ausgangsschicht existieren  $n$  versteckte Schichten (engl.: hidden layers).

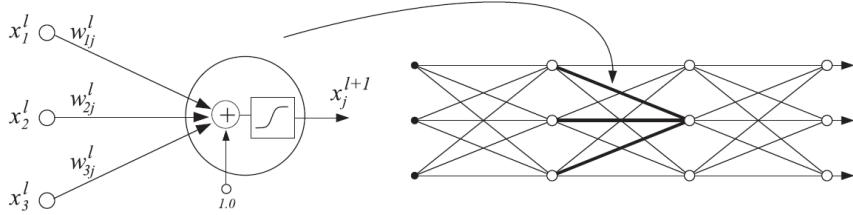


Abbildung 2.2: Künstliches neuronales Netz[27]

Ein Neuron besitzt mehrere Eingangsverbindungen (Gewichte) und ein Ausgangsneuron. Ob ein Neuron „feuert“, wird durch eine lineare oder nicht-lineare Aktivierungsfunktion bestimmt. Die Eingangsgewichte sind veränderbare Werte, die je nach Höhe einen starken oder niedrigen Einfluss auf die Aktivierungsfunktion haben.

$$x_j^{l+1} = f(\sum_i w_{ij}^l x_i^l + w_{bj}^l) \quad (2.2)$$

beschreibt das Neuron  $j$  in Schicht  $l + 1$ , wobei

- $w_{ij}^l$  die Gewichte sind, die Neuron  $i$  in Schicht  $l$  mit Neuron  $j$  verbinden.
- $w_{bj}^l$  der Biasterm des  $j$ -ten Neurons in Schicht  $l$  ist.
- $f$  die Aktivierungsfunktion ist.[27]

## Convolutional Neural Networks

*Convolutional Neural Networks* (CNN, dt.: faltendes neuronales Netzwerk) sind Kategorien von neuronalen Netzen, die besonders in der *Computer Vision* Anwendung finden. In der ersten Schicht werden mehrere Merkmale (engl.: features) durch Filter extrahiert und in separate sog. *Feature Maps* abgelegt, um größere Abstraktionsebenen zu erreichen. Diese Filter sind mathematisch mit Faltungen (engl.: convolutions) zu vergleichen und geben dem Netz den Namen.

Die Dimensionen der Feature Maps werden in einem Poolingschritt<sup>3</sup> (oder

---

<sup>3</sup>Es gibt verschiedene Arten von Pooling (Max, Average, Sum, ...). Dabei wird die  $m * m$  px große Feature Map in sich angrenzende  $n * n$  px große Felder eingeteilt ( $n < m$ ). Im Falle von Max-Pooling wird der höchste Wert aus dem Feld übernommen.

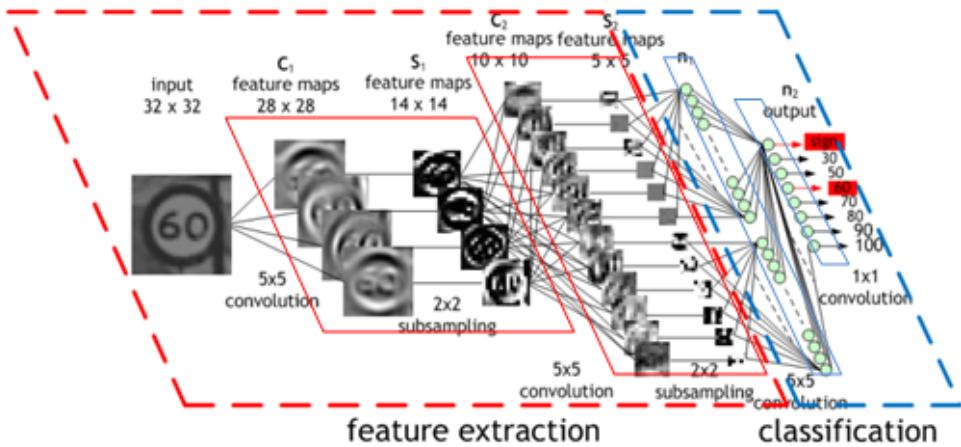


Abbildung 2.3: Architektur eines Convolutional Neural Network[10]

auch *subsampling*) reduziert. Dadurch bleiben nur relevante Informationen erhalten und das CNN wird bis zu einem gewissen Grad robust gegenüber Translationen und Rotationen. In der Regel werden die Faltungen und das das Pooling zwei Mal durchgeführt, wie es in Abb. 2.3 abgebildet ist.

Nach der Merkmalsextraktion werden die Feature Maps zur Klassifikation in eine eindimensionale Schichten geglättet. Die folgenden Schichten bis zur Ausgangsschicht sind vollständig vernetzt.

### 2.4.3 Mask R-CNN

Im Rahmen dieser Arbeit wird das *Mask Region-based Convolutional Neural Network* untersucht. Mask R-CNN ist eine von Facebook AI Research (FAIR) entwickelte Erweiterung des *Faster R-CNN* und kann verschiedene Instanzen einer Klasse in einem Bild von einander trennen. Dazu muss zuerst die Begriffe der Instanzsegmentierung definiert werden.

Einfache Klassifizierung (engl.: *classification*) ordnet Bilder als Ganzes einer Klasse zu. *Semantische Segmentierung* (engl.: *semantic segmentation*) beschreibt die Klassifizierung auf Pixelebene. Es wird erkannt zu welcher Klasse eine Menge von Pixeln gehören, aber es wird nicht zwischen einzelnen Objekten unterschieden. *Objekterkennung* (engl.: *object detection*) entdeckt und lokализiert unterschiedliche Objekte, indem es eine Bounding Box um jedes erkannte Objekt zieht. Jedoch fehlt hier die pixelgenaue Abgrenzung einzelner Objektinstanzen. *Instanzsegmentierung* (engl.: *instance segmentation*) kom-

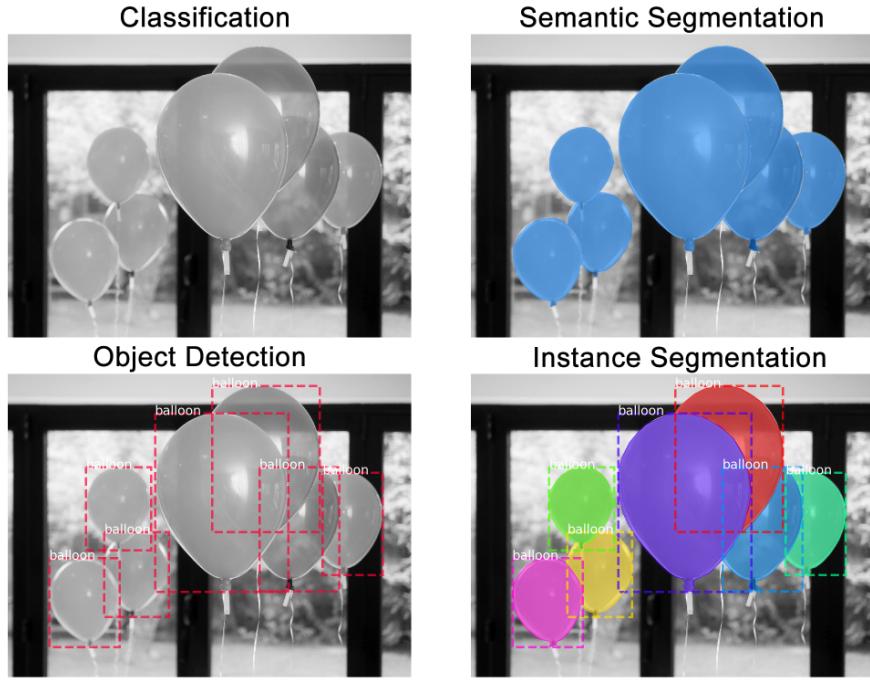


Abbildung 2.4: Unterschied Klassifizierung / semantische Segmentierung / Objekterkennung / Instanzsegmentierung[2]

biniert *Objekterkennung* und *semantische Segmentierung* und ist so in der Lage zwischen einzelnen Objekten zu unterscheiden und ihnen entsprechende Pixel zuzuordnen (s. Abb. 2.4) und ist eine der größten Herausforderungen in der Bildverarbeitung.[15]

Mask R-CNN ist wie Faster R-CNN in zwei Segmente eingeteilt. In dem ers-

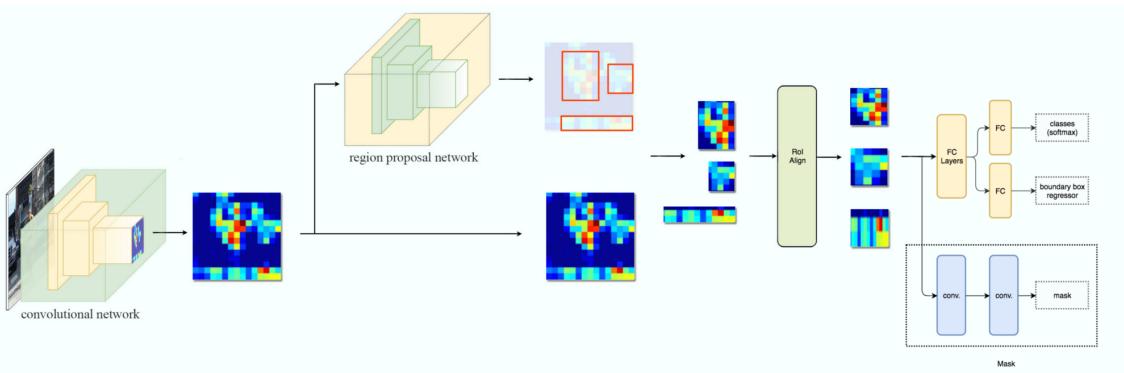


Abbildung 2.5: Mask R-CNN-Architektur[18]

ten Segment, dem *Region Proposal Network* (oder auch RPN), werden mehrere Rahmen (engl.: Bounding Boxes) innerhalb eines Bildes vorgeschlagen, die interessante Objekte beinhalten könnten. Das RPN erzeugt Rechtecke -

sog. Anker (engl.: Anchors) - von unterschiedlichen Größen und Bildverhältnissen, die sich über die Bildregion verteilen und sich überlappen. Für jeden Anker wird eine Ankerklasse und eine Bounding-Box-Verfeinerung ausgegeben. Die Klasse unterscheidet Vordergrund und Hintergrund, wobei eine Bounding-Box mit Vordergrundklassifizierung als potentielle Objekterkennung gewertet wird. Ein Anker ist möglicherweise nicht genau über ein Objekt zentriert. Die Verfeinerung ist eine geschätzte Veränderung des Ankers in Position, Höhe und Größe, um besser das Objekt umrahmen zu können. Wenn mehrere Anker sich zu sehr überschneiden, wird der Anker mit der höchsten Wahrscheinlichkeit ein Objekt zu beinhalten übernommen und der restlichen Anker werden verworfen.<sup>4</sup>[2][24] Die vorgeschlagene Regionen, die einzeln von CNNs bewertet werden, ist der Kernansatz von R-CNN. Das RPN wurde identisch von Faster R-CNN für Mask R-CNN übernommen.[15]

Im zweiten Segment werden aus den Regionen *Bounding Boxes* (dt.: Rahmen)

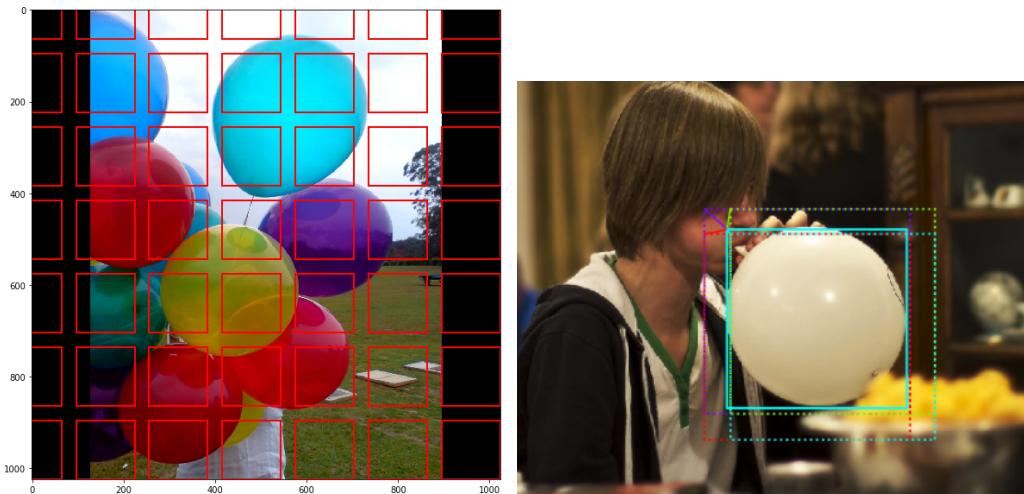


Abbildung 2.6: Links: Vereinfachte Darstellung von Ankern über ein Bild[2] / Rechts: Drei Anker (gepunktet), die das gleiche Objekt umschließen und die Verfeinerung (durchgezogen), die auf diese angewendet wird, um das Objekt genauer einzugrenzen[2]

und Masken generiert und klassifiziert. Die Rahmen haben verschiedene Größen und können Probleme bei der Klassifizierung verursachen. Daher werden die Rahmen auf eine kleine Feature Map gleicher Größe (z.B.  $7 * 7$  px) reduziert. Die Autoren von [15] schlagen eine Methode namens *RoI-Align* vor, bei der Proben aus der Feature Map entnommen werden und eine bilineare Inter-

---

<sup>4</sup>Diese Methode wird *Non-max suppression* genannt.

pulation angewendet wird. In dem bei Faster R-CNN angewandten Verfahren *RoI-Pooling* entstehen durch Quantisierung Informationsverluste und räumliche Abweichungen zwischen Bounding Box und Feature Map, was negative Auswirkungen auf die Maskengenerierung haben kann.[15]

Die oberen vollständig vernetzten Schichten (*FC Layers* in Abb. 2.5) klas-

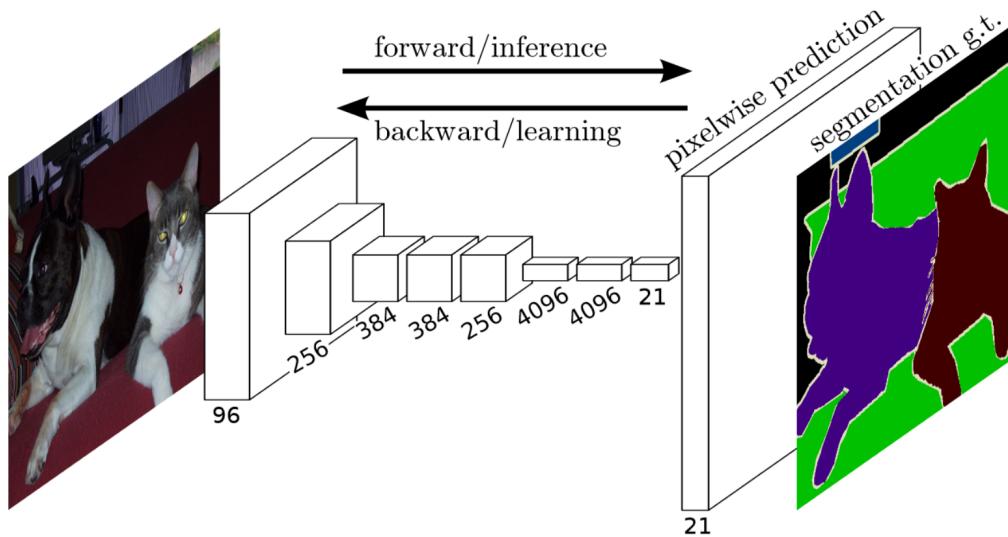


Abbildung 2.7: FCN-Architektur[18]

sifizieren die Regionen und die Bounding Boxes berechnet. Dieser Zweig ist für die Objekterkennung wichtig und noch mit Faster R-CNN gemeinsam.

Gleichzeitig werden in einem parallelen Zweig je Bounding Box  $k m * n$  große Masken zur semantischen Segmentierung erzeugt, wobei  $k$  die Anzahl der Klassen ist. Anders als in dem ersten Zweig des zweiten Segmentes werden die Masken durch *fully convolutional networks* (FCN, dt.: vollständig faltende Netzwerke) prognostiziert. Diese bestehen nur aus faltenden Schichten, wie sie in Kapitel 2.4.2 beschrieben sind. Eine Maske ist eine räumliche Kodierung eines Objektes und daher ist es wichtig räumliche Informationen beizubehalten. Diese können durch die Pixel-zu-Pixel-Übereinstimmung extrahiert werden, welche sonst durch vollständig vernetzter Schichten verloren gehen. Diese geben einen Vektor ohne räumliche Dimensionen aus.[15]

In [15] wird Mask R-CNN mit den *COCO challenge*-Gewinnern<sup>5</sup> der Jahre

---

<sup>5</sup>COCO (Common Objects in Context, dt.: Gewöhnliche Objekte im Kontext) enthält einen Datensatz von über 200000 Bildern in über 80 Kategorien. Der Datensatz ist eine oft genutzte



Abbildung 2.8: Bei FCIS entstehen Artefakte, wenn Objekte sich in einem Bild überlappen.[15]

	backbone	AP	AP <sub>50</sub>	AP <sub>75</sub>	AP <sub>S</sub>	AP <sub>M</sub>	AP <sub>L</sub>
MNC [10]	ResNet-101-C4	24.6	44.3	24.8	4.7	25.9	43.6
FCIS [26] +OHEM	ResNet-101-C5-dilated	29.2	49.5	-	7.1	31.3	50.0
FCIS+++ [26] +OHEM	ResNet-101-C5-dilated	33.6	54.5	-	-	-	-
<b>Mask R-CNN</b>	ResNet-101-C4	33.1	54.9	34.8	12.1	35.6	51.1
<b>Mask R-CNN</b>	ResNet-101-FPN	35.7	58.0	37.8	15.5	38.1	52.4
<b>Mask R-CNN</b>	ResNeXt-101-FPN	<b>37.1</b>	<b>60.0</b>	<b>39.4</b>	<b>16.9</b>	<b>39.9</b>	<b>53.5</b>

Tabelle 2.2: Instance segmentation *mask AP* auf COCO *test-dev*. MNC und FCIS sind Sieger der COCO 2015 und 2016 Challenge. Mask R-CNN erzielt deutlich bessere Ergebnisse als die komplexere FCIS+++. [15]

2015 und 2016 verglichen. Der Vergleich zeigt, dass Mask R-CNN in der Challenge bessere Werte erzielt als die Konkurrenten (s. Tab. Des Weiteren fällt *fully convolutional instance segmentation* (FCIS, dt.: vollständig faltende Instanzsegmentierung) auf, wenn es mit überlappenden Objekten konfrontiert wird. Dort erzeugt es Artefakte, welche durch Mask R-CNN nicht entstehen (s. Abb. 2.8). Durch diese Gegenüberstellungen wird gezeigt, dass Mask R-CNN alle aufgeführten Anforderungen erzielt. Es erkennt Klasseninstanzen auf Pixelebene und weist eine hohe Robustheit auf. Auch die Genauigkeit hebt sich beim direkten Vergleich ab. Aus diesen Gründen wurde Mask R-CNN im Rahmen dieser Arbeit ausgewählt.

Basis, um Objekterkennungstechniken zu evaluieren und zu bewerten.[7]

## 2.5 Evaluation des Modells

Jetzt wo gezeigt wurde, welches Modell in dieser Arbeit genutzt wird, fehlt eine Möglichkeit ein trainiertes Modell zu bewerten. *Mean average precision* (oder auch mAP) ist eine Metrik, um die Genauigkeit einer Instanzsegmentierung zu messen.<sup>6</sup> Aber bevor die erklärt werden können, muss noch die Begriffe *Precision*, *Recall* und *Intersection over Union* eingegangen werden.

### 2.5.1 Intersection over Union

*Intersection over Union* (oder auch IoU, dt: Schnitt über Vereinigung) ist eine wichtige Metrik für die semantische Segmentierung. Sie vergleicht die vorhergesagte Maske mit der Grundwahrheit<sup>7</sup>, um zu messen wie gut die Vorhersage mit der Grundwahrheit übereinstimmt.[19]

$$IoU = \frac{\text{Grundwahrheit} \cap \text{Vorhersage}}{\text{Grundwahrheit} \cup \text{Vorhersage}} \quad (2.3)$$

Die Schnittmenge beinhaltet alle Pixel, die sich in der Grundwahrheit als auch in der vorhergesagten Maske befinden. Pixel, die sich in der Grundwahrheit und in der Vorhersage befinden, werden von der Vereinigung zusammengefasst.

In der semantischen Segmentierung wird für jede Klasse ein unterschiedlicher IoU-Wert berechnet und dann wird der Mittelwert aus diesen Werten ermittelt, um einen globalen Messwert zu haben. In der Instanzsegmentierung wird für jede einzelne Objektinstanz mittels Instanzgrundwahrheit und Instanzvorhersage ein separater IoU-Wert berechnet. Wenn ein bestimmter Grenzwert überschritten wird, gilt diese Instanz als tatsächlich richtige Erkennung.[20]

### 2.5.2 Precision und Recall

*Precision* (oder auch Falsch-Positiv-Rate) sagt aus mit welcher Wahrscheinlichkeit eine Vorhersage korrekt ist. Diese Metrik wird durch die Formel

$$Precision = \frac{RP}{RP + FP} \quad (2.4)$$

---

<sup>6</sup>mAP ist nicht nur auf Instanzsegmentierung limitiert, sondern wird zum Beispiel auch als Metrik in der Objekterkennung genutzt.

<sup>7</sup>Die Grundwahrheit (engl.: ground truth) ist hier die binäre Maske, die die infizierte Fläche repräsentiert.

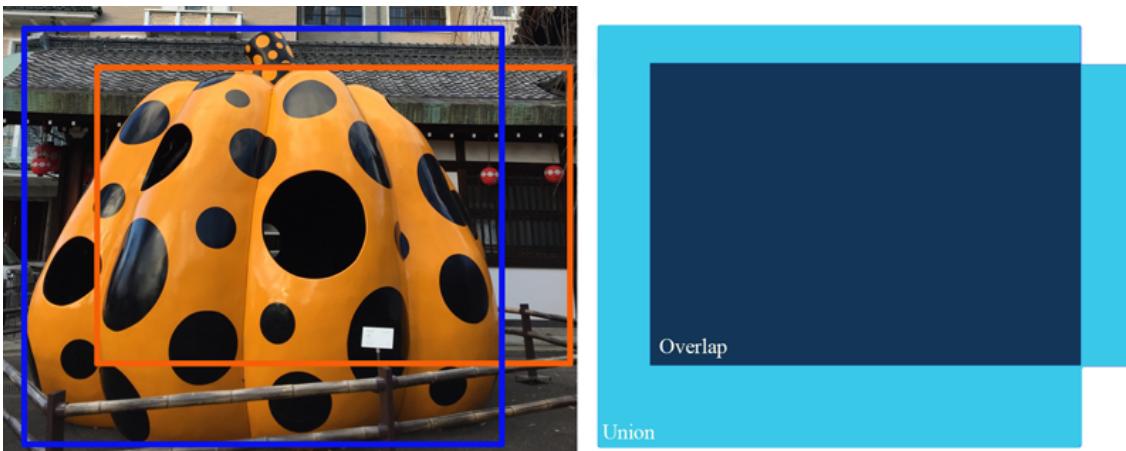


Abbildung 2.9: Beispiel Intersection over Union, Grundwahrheit in blau, Vorhersage in rot[19]

berechnet, wobei  $RP$  (Richtig-Positiv) die Anzahl der richtigen Erkennungen und  $FP$  (Falsch-Positiv) die Anzahl der falschen Erkennungen sei.[19] *Precision* ist also der Anteil von tatsächlich richtigen Erkennungen in Relation zu allen Erkennungen. In Bezug auf Instanzsegmentierung wird die Frage beantwortet, wie viele der erkannten Objekte in einem Bild tatsächlich eine passende Grundwahrheitüberschneidung und eine IoU-Grenzwertüberschreitung haben.[20]

*Recall* (oder auch Falsch-Negativ-Rate) misst die Wahrscheinlichkeit, dass alle tatsächlich wahren Detektionen korrekt erkannt wurden. Diese Metrik wird durch die Formel

$$Precision = \frac{RP}{RP + FN} \quad (2.5)$$

berechnet, wobei  $RP$  (Richtig-Positiv) die Anzahl der richtigen Erkennungen und  $FN$  (Falsch-Negativ) die Anzahl der Objekte, die fälschlicherweise nicht erkannt wurden, sei. *Recall* ist also der Anteil von tatsächlich richtigen Erkennungen in Relation zu allen Objekten im Datensatz.[19] In Bezug auf Instanzsegmentierung wird die Frage beantwortet, wie viele der Objekte mit Grundwahrheit in einem Bild als tatsächlich richtig erkannt werden und eine IoU-Grenzwertüberschreitung haben.[20]

### 2.5.3 Average Precision

Ein *Precision*- und *Recall*-Wert bezieht sich jeweils auf eine detektierte Objektinstanz einer Klasse. Bei mehreren detektierten Objekten einer Klasse in

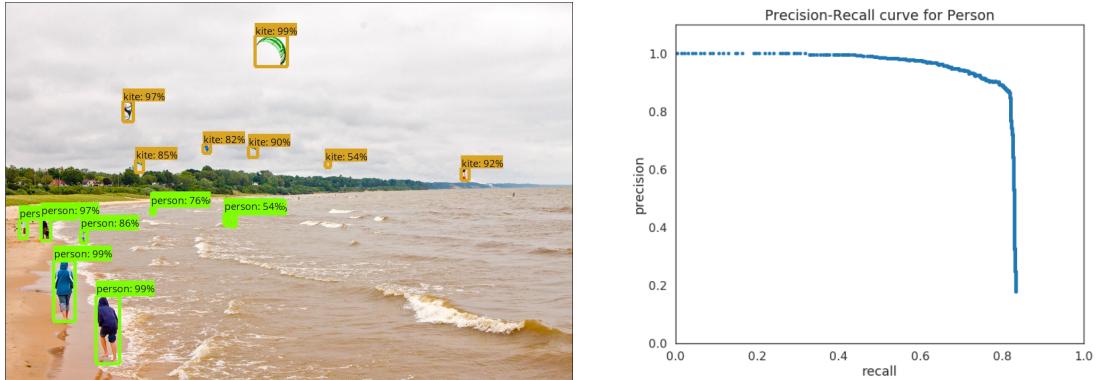


Abbildung 2.10: Links: Beispielbild mit multiplen Detektionen und Klassen[4]  
Rechts: Beispiel Precision-Recall-Kurve für die Klasse “Person”[17]

einem Bild können diese in einer Precision-Recall-Kurve visualisiert werden (s. Abb. 2.10). *Average Precision* (oder auch AP) fasst die Form der Kurve zu einem Wert zusammen, indem es den Durchschnitt der *Precision*-Werte an elf *Recall*-Werten  $[0, 0.1, \dots, 1]$  berechnet:

$$AP = \frac{1}{11} \sum_{r \in \{0, 0.1, \dots, 1\}} p_{interp}(r) \quad (2.6)$$

Ein *Precision*-Wert  $p$  an der *Recall*-Stelle  $r$  wird interpoliert, indem der Maximumwert übernommen an der *Recall*-Stelle  $\tilde{r} \geq r$  wird:

$$p_{interp}(r) = \max_{\tilde{r}: \tilde{r} \geq r} p(\tilde{r}) \quad (2.7)$$

wobei  $p(\tilde{r})$  der *Precision*-Wert  $p$  an der *Recall*-Stelle  $\tilde{r}$  sei. Die Interpolation reduziert den Einfluss kleiner, lokaler Unebenheiten in der Kurve.[17]

## 2.5.4 Mean Average Precision

*Mean Average Precision* ist der Durchschnitt aller *Average Precision*-Werte jeder Klasse in jedem Element eines (Sub-)Datensatzes.<sup>8</sup> *mAP* wird zum Beispiel auch in der COCO- oder PASCAL-VOC-Challenge benutzt, um die Resultate der Challenge-Teilnehmer zu bewerten (s. Tabelle 2.2). Aber hier kann es zu Unterschieden kommen, wie der *mAP* berechnet wird. So ist es bei der COCO-Challenge der durchschnittliche *mAP* über verschiedene *IoU*-Grenzwerte. Hier wird jeweils ein *mAP* an zehn verschiedenen *IoU*-Werten  $[0.5, 0.55, \dots, 0.95]$  berechnet und aus den Ergebnissen wird der Durchschnitt ermittelt.[8] In dieser

<sup>8</sup>*mAP* wird oft nur *AP* genannt.

Arbeit wird stets  $IoU = 0.5$  als Grenzwert benutzt, um die Auswertung einfach zu halten.

# Kapitel 3

## Overfitting

### 3.1 Begriffserklärung

Genaue Daten über Krankheitsbefäle im Agrarsektor sind rar, da diese in der Regel nicht öffentlich zugänglich sind.<sup>1</sup> Daher musste mit *Overfitting* gerechnet werden. Das künstliche neurale Netzwerk soll daraufhin trainiert werden, dass es möglichst alle Befäle, die untersucht werden, erkennt. Dafür wird es im ersten Schritt mit einem Trainingsdatensatz trainiert. Im folgenden Schritt mit einem kleineren Validierungsdatensatz überprüft, wie gut das Netz trainiert wird. Overfitting tritt auf, wenn das Netz auf die Daten aus dem Trainingsdatensatz mit sehr hoher Erfolgsquote erkennt, jedoch vergleichsweise schlechte Ergebnisse bei der Validierung bzw. bei unbekannten Daten erzielt. Das geschieht, weil sich das Netz auf nicht relevante Datenpunkte konzentriert, die im nur Trainingsdatensatz auftreten, aber nicht die allgemeine Charakteristika der Objekte widerspiegeln.

In Abb. 5.5 ist ein Beispiel wie Overfitting sich auswirken kann. Die linken zwei Bilder sind ein exemplarischer Auszug aus dem Trainingsdatensatz. Einmal eine visuelle Repräsentation der NDVI-Werte der infizierten Agrarfläche und die Binärmaske, welche die infizierte Fläche markiert. Das selbe Bild wurde nach einem erfolgreichen Trainingsdurchlauf der Mask R-CNN-Implementierung übergeben und es hat den erkrankten Bereich nahezu perfekt erkannt. Das vierte Bild zeigt zentriert das selbe Feld. Jedoch ist der Ausschnitt größer, rotiert und die Aufnahme stammt von einem anderen Datum. Der Prognose zur Folge ist die Infizierung auf die benachbarten Felder über-

<sup>1</sup>Datenschutz kann ein Grund dafür sein.

Genaues  
Da-  
tum  
nö-  
tig?

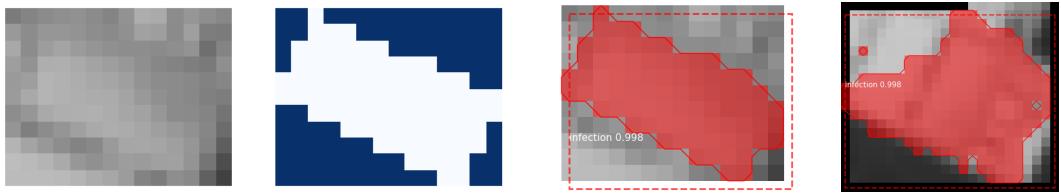


Abbildung 3.1: V.l.n.r. Bild von infizierter Agrarfläche aus Trainingsdatensatz / Binärmaske der infizierten Region, wird gemeinsam mit dem linken Bild zum Training in das KNN gespeist / Selbiges Bild, Ergebnis nach Trainingsdurchlauf, prognostizierte Ergebnisfläche in rot / Bild der selben Fläche, was nicht aus dem Trainingsdatensatz stammt, prognostizierte Ergebnisfläche in rot

gesprungen, was nicht der Realität entspricht. Overfitting ist ein bekanntes Problem im Bereich des maschinellen Lernens und es existieren multiple Methoden, um dem entgegenzuwirken.

## 3.2 Data Augmentation

Generell ist ein sehr großer Datensatz ( $\geq 10000$  Elemente) für das Training förderlich. Jedoch ist das nicht immer möglich. In diesem Fall kann der Datensatz künstlich durch *Data Augmentation* vergrößert werden. Zu Data Augmentation zählen geringe Veränderungen der Daten - hier Bildmanipulationen. Solche Operationen können unter anderem

- Rotationen,
- Spiegelungen,
- Translationen,
- zufällige Ausschnitte,
- Gauß'sches Rauschen,
- Helligkeitsveränderungen,
- oder Kombinationen davon

beinhalten. Wobei nicht jede Augmentation-Technik für jeden Anwendungsfall sinnvoll ist. Wenn zum Beispiel ein KNN auf Autoerkennung trainiert werden soll, ist es nicht nützlich oder sogar hinderlich die Bilddaten so zu rotieren, dass es von oben nach unten oder umgekehrt zeigt. Des weiteren ist

sinnvoll Data Augmentation einzusetzen, obwohl genügend Daten vorhanden sind. So besteht der Datensatz aus zwei Klassen von Autos, Marke A und Marke B. Die Front der Fahrzeuge der Marke A sind nach rechts ausgerichtet, während die Autos der Marke B nach links ausgerichtet sind. Das neuronale Netz wird diesen markanten Unterschied den Marken zuordnen und wird ein links ausgerichtetes Fahrzeug der Marke A als ein Fahrzeug der Marke B klassifizieren.[5]

Für den Anwendungsfall dieser Arbeit sind Rotationen, Spiegelungen, zufällige Ausschnitte und Translationen valide Optionen zur Vergrößerungen des Datensatzes. Wie in dem vorherigen Beispiel wird das Modell auch Merkmale wie Form, Position im Bild oder Ausrichtung der RoI untersuchen, welche in der Realität keine feste Muster haben und nicht vorhersagbar sind. Damit wird nicht nur der Datensatz vergrößert, sondern auch das Modell generalisiert.

Künstliches Rauschen und Helligkeitsveränderungen sind mit hoher Wahrscheinlichkeit nicht geeignet. Die Bilddaten bestehen aus quantifizierten Werten und die Operationen könnten diese Werte verfälschen und damit das Endergebnisse negativ beeinflussen.

### 3.3 L2 Regularization

Ein kleiner Datensatz führt zu einem komplexen Modell und komplexere Modelle neigen zu Overfitting. *L2 Regularization* (oder auch *Ridge Regression*) vereinfacht das Modell, indem es hohe Gewichte bestraft und niedrige Gewichte bevorzugt.<sup>2</sup> Hohe Gewichte können mit Anomalien korrelieren, die nur im Trainingsdatensatz auftreten und so wird das Netz Schwierigkeiten haben, fremde Daten richtig zu erkennen.

$$loss + \lambda \sum_{j=1}^p \beta_j^2 \tag{3.1}$$

Ridge Regression addiert einen zusätzlichen Bestrafungsterm (engl.: *penalty term*) zur Verlustfunktion (engl.: *loss function*), wobei *loss* die Verlustfunktion, der letzte Term der Bestrafungsterm und  $\lambda > 0$  sei. Hier ist darauf zu achten,

---

<sup>2</sup>Es sei erwähnt, dass es neben L2 Regularization noch L1 Regularization (oder auch Lasso Regression) existiert. Diese Methode wird eingesetzt, um Underfitting zu verhindern.

dass  $\lambda$  sinnvoll gewählt wird. Wenn es zu groß gewählt wird, wird zu viel Gewicht hinzugefügt und das Modell tendiert zum *Underfitting*<sup>3</sup>.[3][23]

### 3.4 Zusätzliche Methoden

Wenn ein KNN initial trainiert wird, werden die einzelnen Gewichte zufällig gewählt und dann dem Idealwert angenähert. Dieser Prozess kann zeitlich verkürzt werden indem vor-trainierte Gewichte eingesetzt werden. Je länger ein Modell trainiert werden muss, desto größer ist die Gefahr des *Overfittings*. Darum wird die Mask R-CNN-Implementierung mit Gewichten initialisiert, die auf dem COCO-Datensatz trainiert wurden.

In einem großen Datensatz beeinflussen einzelne Anomalien das Training nicht. Anders können diese Anomalien einen starken Einfluss in einem kleinen Datensatz haben. Eine hohe Anzahl von trainierbaren Parametern (Gewichten) reagieren darauf empfindlicher und es gilt das Modell durch Parameterminimierung resilenter zu machen. Um die Anzahl an Gewichten, die trainiert werden, zu minimieren, kann das *Backbone*<sup>4</sup> vereinfacht werden. Hier werden *ResNet50* und *ResNet101*, welche jeweils 50 und 101 Schichten besitzen, miteinander verglichen. Hierbei sollte *ResNet50* vorteilhafter sein, da es von beiden vorgestellten Architekturen die simplere hat.

Bei einem kleinen Datensatz hilft es die Experimente simpel zu gestalten. Das betrifft auch die Anzahl der Klassen. In dem betroffenen Region wurden zwei verschiedene Krankheiten festgestellt, die als einzelne Klassen deklariert werden können. Um die Datengröße pro Klasse zu erhöhen, werden diese beiden Klassen zu einer Klasse *infection* zusammengefasst.

---

<sup>3</sup>Underfitting bezeichnet eine schlechte Performanz des neuronalen Netzes auf sämtliche Daten inkl. dem Trainingsdatensatz.

<sup>4</sup>He et al. bezeichnen die Architektur, die für die Merkmalsextraktion verantwortlich ist, als Backbone. Das Segment, das Klassifizierung, Bounding-Box-Generierung und Maskenerkennung durchführt, wird als *network head* (oder nur *head*) definiert.[15]

# Kapitel 4

## Konzept und Implementierung

### 4.1 Konzept

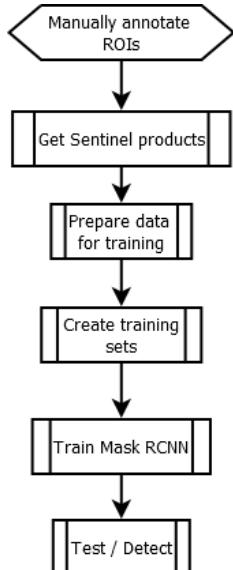


Abbildung 4.1: Gesamtablauf der Anwendung

Das Programmablauf wird in einzelne Schritte unterteilt, auf die in den nächsten Kapiteln näher eingegangen werden.

Zuerst müssen die Daten für das Training bzw. für die Erkennung manuell annotiert werden. Diese Metadaten werden dann genutzt, um automatisch Sentinelprodukte<sup>1</sup> mittels einer API, die von der Copernicus zur Verfügung gestellt wird, herunterzuladen. Aus den Produkten werden die relevanten Bänder extrahiert und unter anderem die jeweiligen NDVI-Werte berechnet. Nachdem

---

<sup>1</sup>Aufnahmenpakete der Sentinel-Plattformen werden als Produkte bezeichnet.

die Produkte für das Training vorbereitet wurden, werden die Daten in ein Trainings- und in ein Validierungsdatensatz aufgeteilt. Der folgende Trainingsprozess basiert auf diesen Datensätzen. Sobald das Training abgeschlossen ist, kann die Performanz des Modells getestet werden.

## 4.2 Annotation

Zu Beginn werden die Regionen, die entweder für das Training benutzt oder überprüft werden, manuell erfasst. Vorausgesetzte Informationen sind

- Geografische Koordinaten,
- Zeitraum des Befalls und
- Bezeichnung der Infektion.

Als Format dieser Informationen dient *GeoJSON*<sup>2</sup>. GeoJSON enthält nicht nur geografische Daten, sondern ist auch um benutzerdefinierte Eigenschaften (*properties*) erweiterbar. Die Annotationen sind also GeoJSON-Features, die ein geografisches Polygon mit Metadaten enthalten.

```
{  
  "type": "Feature",  
  "properties": {  
    "disease": 1,  
    "from": "2018-07-12T13:00:00Z-7DAYS",  
    "to": "2018-07-12T13:00:00Z+7DAYS"  
  },  
  "geometry": {  
    "type": "Polygon",  
    "coordinates": [[[11.171988617177981, 44.574291380353003],  
      [11.1726616444942, 44.574017992242283],  
      [11.17338129910439, 44.575068359984279],  
      [11.17273129334275, 44.575299863118993],  
      [11.171988617177981, 44.574291380353003]]]  
  }  
}
```

---

<sup>2</sup>GeoJSON ist eine Erweiterung des JSON-Format und beschreibt geografische Daten und Geometrien. GeoJSON wird durch den RFC7946-Standard definiert.

---

#### Listing 4.1: Annotation

`properties.disease` enthält die eindeutige, nummerische Repräsentation der Klasse bzw. Krankheit, die in dieser Region enthalten ist. Die Zuordnung der nummerischen Werte und des textuellen Bezeichners werden in einer separaten JSON als Schlüssel-Wert-Paare konfiguiert, wobei der Schlüssel nummerisch und der Wert textuell ist. Hier ist, darauf zu achten, dass der Schlüssel  $\geq 1$  ist, da 0 der implizite Schlüssel der Mask R-CNN-Implementierung für den Hintergrund ist. Diese Eigenschaft ist nur für das Training von Relevanz.

`properties.from` und `properties.to` sind jeweils Start- und Endzeitpunkt, in dem nach verfügbaren Sentinelprodukten gesucht werden soll. Das Format der jeweiligen Eigenschaften kann eine der folgenden Formen haben<sup>3</sup>:

- yyyyMMdd
- yyyy-MM-ddThh:mm:ss.SSSZ (ISO-8601)
- yyyy-MM-ddThh:mm:ssZ
- NOW
- NOW-<n>DAY(S) (oder HOUR(S), MONTH(S), usw.)
- NOW+<n>DAY(S)
- yyyy-MM-ddThh:mm:ssZ-<n>DAY(S)
- NOW/DAY (oder HOUR, MONTH usw.) - Der Wert wird entsprechend (z.B. auf den Tag) gerundet.

Es ist angebracht einen Zeitraum von mehreren Tagen bzw. Wochen zu wählen, da die Sentinel-2-Satelliten keine täglichen Daten liefern und weil eine Infektion typischerweise über einen längeren Zeitraum vorherrscht. Die Zeitspanne ist von der Krankheit abhängig. Hier wurden eine Woche vor und nach dem Aufnahmzeitpunkt genutzt, um nach Produkten zu suchen.

---

<sup>3</sup>Die Formate basieren auf der sentinel-sat-Version 0.12.2.

## 4.3 Suche nach Sentinelprodukten

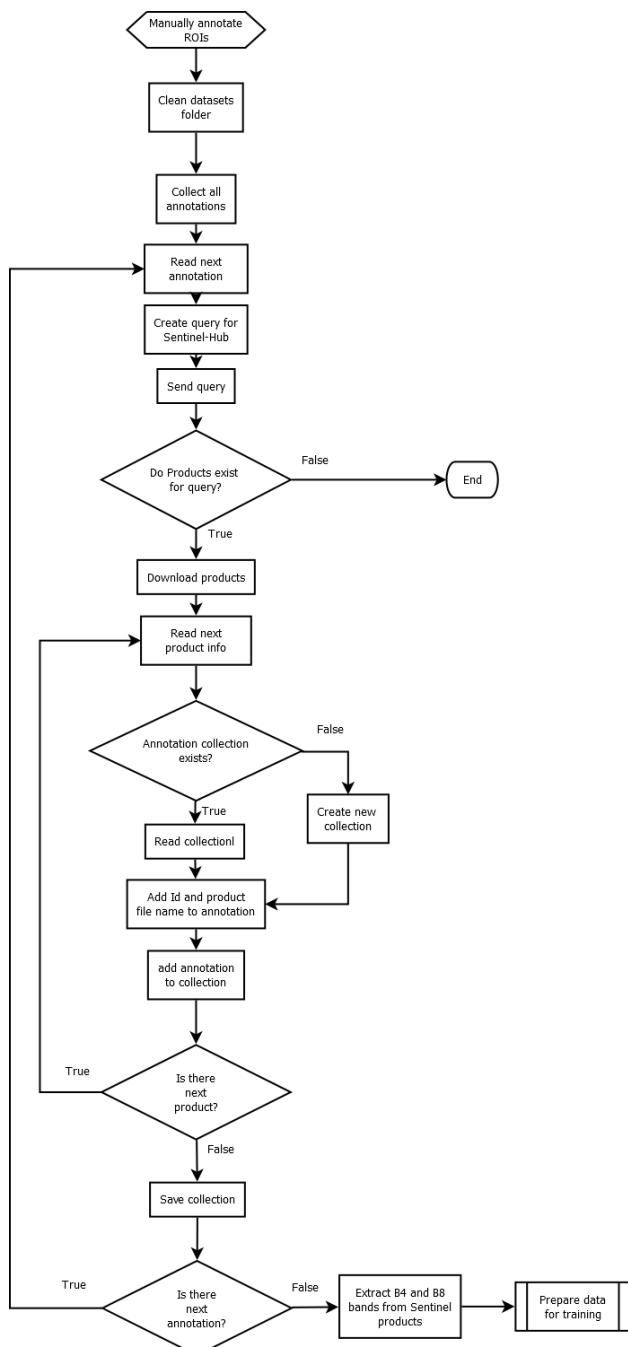


Abbildung 4.2: Ablaufdiagramm Sentineldatenaufbereitung

Der *Copernicus Open Access Hub*<sup>4</sup> ermöglicht freien und offenen Zugriff auf Sentinel-Produkte. Die Daten sind sowohl über eine grafische Oberfläche als auch über eine REST-API verfügbar. Vorausgesetzung für beide Optionen ist

<sup>4</sup><https://scihub.copernicus.eu/>

ein Account, der über die grafische Oberfläche erstellt werden kann.

Die Nutzung der Schnittstelle erfolgt über die Python-Bibliothek `sentinelsat`<sup>5</sup>. Bei einer Anfrage müssen die GeoJSON-Dateien in WKT<sup>6</sup> umgewandelt werden, was von der Bibliothek übernommen werden kann. Die WKT-Geometrie wird als *footprint* (dt.: Fußabdruck) bezeichnet. Solang es nicht anders angegeben wird, gibt die Copernicus-API Produkte zurück, die die RoI schneiden. Des Weiteren wird der Plattformname statisch als 'Sentinel-2' definiert, damit keine Produkte von den anderen Sentinelplattformen zurückgegeben werden. Der Suchzeitraum wird aus der jeweiligen GeoJSON-Datei übernommen. Sollten für die Suchanfragen keine Produkte existieren, wird die Anwendung beendet, da es keine Basis gibt, auf der das Netzwerk trainiert werden kann. Eventuell muss bei so einem Fall der Zeitraum erweitert und Prozess wiederholt werden. Bei vorhandenen Produkten lädt das Skript diese herunter. Die Produkte werden in einem komprimierten Format geliefert und enthalten neben zusätzlichen Informationen, Banddaten in separaten Dateien im JPEG2000-Format<sup>7</sup>.

Für jedes Produkt, das zur aktuellen Annotation gehört, wird ein neuer Eintrag zu einer *FeatureCollection* hinzugefügt. Für die spätere Entwicklung sind die Annotationsen so leichter zu finden und bearbeitbar. Außerdem bleiben dadurch die Originaldaten unberührt. Dieser Schritt wird für jede vorhandene Annotationsdatei wiederholt. Anschließend werden die Bilddateien für B4 und B8 aus dem Produkt extrahiert.

---

<sup>5</sup><https://sentinelsat.readthedocs.io/en/stable>

<sup>6</sup>WKT (Well-known text) ist eine Markup-Sprache zur Repräsentation von geometrischen Objekten auf Karten und räumlichen Referenzsystemen.

<sup>7</sup>JPEG 2000 genau wie GeoTIFF ist ein Bildformat in dem auch Metadaten abgelegt werden können. So sind Pixel geografischen Koordinaten zuordbar.

## 4.4 Aufbereitung der Sentineldaten

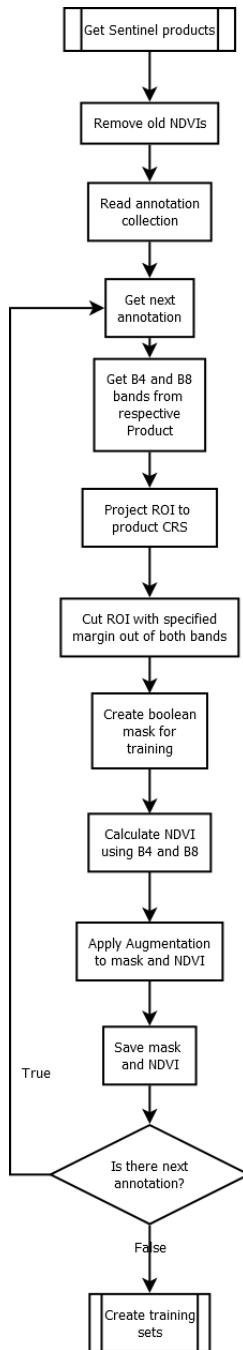


Abbildung 4.3: Ablaufdiagramm der Aufbereitung

Originale Sentinel-2-Aufnahmen haben eine Größe von 10980 \* 10980 px und müssen deshalb deutlich verkleinert werden, um die Speicherbelastung zu minimieren. Vor allem da nur kleine Ausschnitte von wenigen Pixeln benötigt werden.

Zu Beginn werden alle Elemente aus der vorher erstellten *FeatureCollection* geladen und nacheinander bearbeitet. Jedes Produkt hat ein eigenes Koordinatenreferenzsystem (engl.: Coordinate Reference System, CRS) und unter Umständen unterscheiden sich die Systeme des Produktes und des Polygons<sup>8</sup>. Diese müssen gleich sein, um miteinander agieren zu können. Das Produkt-CRS kann aus den extrahierten Bändern gelesen und dann dazu genutzt werden, um die Annotationskoordinaten in eben dieses zu projizieren.

Nachdem abgesichert wurde, dass die RoI in dem Sentinelprodukt enthalten ist, wird aus den Bändern die RoI samt einem vorher definierten Rand ausgeschnitten. Der Rand ist später bei der Data Augmentation hilfreich. Gleichzeitig erstellt die Anwendung eine gleich große binäre Maske, wobei die Elemente der Maske mit den Pixeln der RoI korrespondieren. Die Elemente, die die RoI repräsentieren, enthalten einen wahren Wert.

Nun wird der NDVI aus den B4- und B8-Ausschnitten, wie in Kapitel 2.2 ge-



Abbildung 4.4: V.l.n.r. B4 (RED), B8 (NIR), NDVI

zeigt, berechnet (s. Abb. 4.4). Das Ergebnis wird nun mittels Data Augmentation vervielfältigt. Dazu wird es vier mal um 90° gedreht. Danach wird jedes rotierte Bild horizontal und vertikal gespiegelt. Darauf werden neun mal aus den rotierten und gespiegelten Daten zufällig Bilder in der Größe von 16\*16 px ausgeschnitten. Durch diese Operationen vergrößert sich die Grundgesamtheit um den Faktor 108. Jede Operation wird ebenfalls auf die entsprechende Maske angewandt. Dabei wird darauf geachtet, dass die Maske nicht vollständig aus falschen Werten besteht und dass die manipulierten Dateien ebenfalls der *FeatureCollection* hinzugefügt werden.

---

<sup>8</sup>Nach RFC 7946 ist das geografische Referenzsystem *World Geodetic System 1984* (WGS 84) das Standardsystem.[6]

## 4.5 Trainings- und Validierungsdatensatz

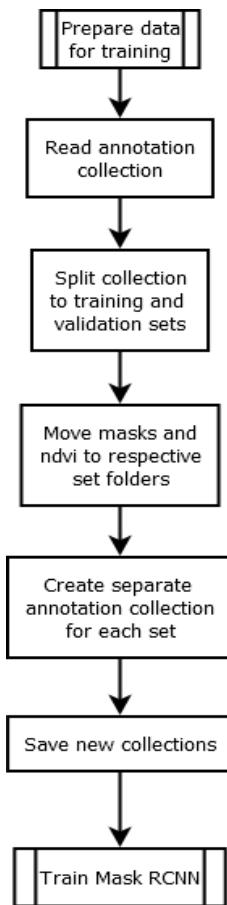


Abbildung 4.5: Ablaufdiagramm der Datensatzaufteilung

Die berechneten NDVI-Bilddateien und deren Masken bilden die Gesamtheit des Datensatzes. Um das Modell trainieren und testen zu können, muss diese Gesamtheit aufgeteilt werden.

- Trainingsdatensatz  
Hiermit werden die Gewichte des neuronalen Netzwerkes trainiert.
- Validierungsdatensatz  
Dieser Datensatz wird genutzt, um das trainierende Modell wiederholt zu evaluieren. Die Parameter in dem Modell werden auf Basis dieser Evaluation verändert, um das Idealergebnis zu approximieren.
- Testdatensatz  
Während der Validierungsdatensatz während des Trainings genutzt wird, ist der Testdatensatz zu Evaluation des fertig trainierten Modells geeig-

net. Das KNN sieht also die Elemente nicht während des Trainingprozesses und repräsentiert „fremde“ Daten. Damit ist der Testdatensatz ein besserer Indikator der Leistung des Modells als der Validierungsdatensatz.

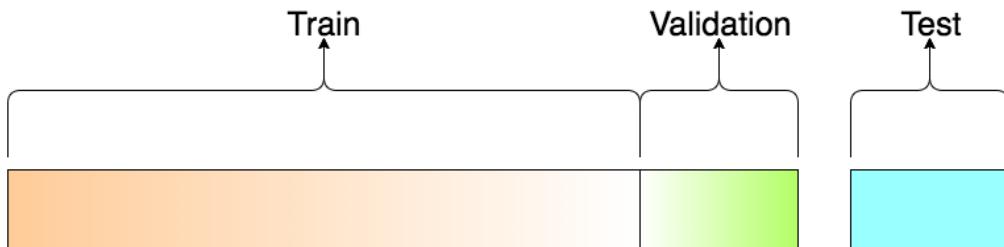


Abbildung 4.6: Darstellung der relativen Datensatzverteilung[25]

In welchem Verhältnis die Datensätze aufgeteilt werden, hat einen Einfluss auf das Training und hängt von der eigentlichen Größe des gesamten Datensatz und von dem trainierten Modell ab. Komplexere Modelle benötigen einen größeren Validierungsdatensatz, während ein Modell mit weniger Parametern mit einem kleineren auskommt.[25] Typischerweise besteht jedoch der Trainingsdatensatz aus dem größten Teil.

Hier wird zunächst die Gesamtmenge zwischen einem unbekannten Teildatensatz (80% der Gesamtheit) und dem Testdatensatz (20% der Gesamtheit) aufgeteilt. Danach teilt sich der Teildatensatz in den Trainingsdatensatz (80% des Teiles) und in den Validierungsdatensatz (20% des Teiles). Die hier angewandten prozentualen Werte sind häufig angewandte initiale Verhältnisse und können bei Bedarf angepasst werden. Die geteilten Mengen werden unterschiedlichen Orten abgelegt und für jede Sammlung wird jeweils eine neue *FeatureCollection* erzeugt, die die Annotationen der entsprechenden Elemente enthalten.

## 4.6 Training/Detektion

Mit den vorbereiteten und aufgeteilten Daten kann nun das Netzwerk angeleert werden. In dieser Arbeit wurde eine Implementierung<sup>9</sup> des kalifornischen Unternehmens Matterport, Inc. erweitert und genutzt. Matterport stellte den Quellcode 2017 unter MIT-Lizenz der Öffentlichkeit zur freien Verfü-

---

<sup>9</sup>[https://github.com/matterport/Mask\\_RCNN](https://github.com/matterport/Mask_RCNN)

gung. Das neuronale Netz wurde mittels Python 3, Tensorflow und Keras implementiert. Zum Training eigener Daten müssen die zwei Python-Basisklassen `Dataset` und `Config` erweitert werden.

#### 4.6.1 Dataset

Die Klasse `Dataset` bietet einen Weg, eigene Datensätze in das Modell zu laden, da die Daten in unterschiedlichen Formaten vorkommen können. Für die Erweiterung erbt die neue Klasse `CropDiseaseDataset` von `Dataset` und überschreibt die Methoden `Dataset.load_mask`, `Dataset.load_image` und `Dataset.image_reference`. Zusätzlich wurde die Methode `CropDiseaseDataset.load_crop_disease` implementiert. Eine Instanz von `Dataset` bzw. `CropDiseaseDataset` bildet nicht die gesamte Datenmenge ab, sondern jeweils einen spezifizierten Subdatensatz, die in Kapitel 4.5 definiert wurden. So muss jeder Teil separat instanziert werden.

`CropDiseaseDataset.load_crop_disease` fügt dem Objekt die Klassen- bzw. Bildinformationen wie eindeutige Bezeichnung und Dateipfad durch die internen Methoden `Dataset.add_class` bzw. `Dataset.add_image` hinzu.

`CropDiseaseDataset.load_image` lädt die eigentliche Bilddatei in den Arbeitsspeicher. Die NDVI-Werte wurden als Grauskalenbilder gespeichert und müssen in ein RGB-Format konvertiert werden. Es ist möglich durch Anpassungen des Mask RCNN-Codes, auch Dateien mit mehr oder weniger als drei Farbkanälen zu nutzen. Jedoch verursachte das Fehler, die zum Zeitpunkt der Verschriftlichung nicht gelöst wurde. Da es sich dabei um keinen kritischen Fehler handelt, wurde dieser durch die Konvertierung umgangen. Da die NDVI-Werte zwischen  $-1$  und  $1$  liegen und Farbwerte aus ganzzahligen Werten bestehen, werden die NDVI-Werte mit  $255$  multipliziert, bevor sie schließlich hinzugefügt werden.

`CropDiseaseDataset.load_mask` liest die binären Masken aus den Dateien und ordnet sie einer Klasse und einem Bild zu.

Die Methode `CropDiseaseDataset.image_reference` gibt den vollständigen Dateipfad einer Bilddatei zurück.

## 4.6.2 Config

Die Klasse Config ist eine Sammlung von Parametern, die zur Konfiguration des Modells genutzt werden. Diese Parameter können entsprechend angepasst werden und haben jeweils Einfluss auf das Training bzw. auf die Erkennung. Es wird hier nur auf die Parameter eingegangen, die explizit im Rahmen der Entwicklung verändert wurden, um das Ergebnis des Trainings zu verbessern. Die Parameterbezeichnung ist der Name des Parameters, so wie er in Config deklariert wurde. Die Erklärung beschreibt den Parameter näher. Die angegebenen Werte sind alle möglichen Werte, die in der Entwicklung genutzt wurden. Welche Werte bei welchem Trainingsdurchlauf genutzt wurden, wird im Kapitel ?? beschrieben.

**Parameterbezeichnung:** BACKBONE

**Erklärung:** Die Backbone-Architektur mit der das Modell gebildet wird.

**Werte:** resnet50

**Parameterbezeichnung:** DETECTION\_MIN\_CONFIDENCE

**Erklärung:** Minimalste Wahrscheinlichkeit einer Detektion, um sich als vorhergesagte Instanz zu qualifizieren. Detektionen mit einer Wahrscheinlichkeit niedriger als dieser Wert werden ignoriert.

**Werte:** 0

**Parameterbezeichnung:** GPU\_COUNT

**Erklärung:** Anzahl der Grafikkarten, auf denen das Modell trainiert wird. Wenn die CPU die Berechnungen übernimmt, muss der Parameter den Wert 1 annehmen.

**Werte:** 1

**Parameterbezeichnung:** IMAGE\_MAX\_DIM

**Erklärung:** Wichtig für IMAGE\_RESIZE\_MODE. Bildhöhe und -breite werden auf diesen Wert (in px) vergrößert.

**Werte:** 128

**Parameterbezeichnung:** IMAGE\_RESIZE\_MODE

**Erklärung:** Methode mit der ein Bild vergrößert bzw. verkleinert wird. Die gewählte Option vergrößert die Datensätze auf IMAGE\_MAX\_DIM\*IMAGE\_MAX\_DIM und füllt das Bild mit 0-Werten, sollte das Ausgangsbild kein Quadrat sein.

**Werte:** square

**Parameterbezeichnung:** IMAGES\_PER\_GPU

**Erklärung:** Bilder die pro Schritt gleichzeitig für das Training in den Speicher geladen werden. Dieser Wert ist - je nachdem man auf der CPU oder auf der GPU trainiert - abhängig von der Größe der Arbeitsspeichers oder Grafikspeichers. Für die Detektion ist nur ein Bild pro Schritt erlaubt.

**Werte:** 1, 4

**Parameterbezeichnung:** LEARNING\_RATE

**Erklärung:** Die Lernrate gibt an, wie stark die Gewichte des Netzwerkes korrigiert werden. Wenn dieser Wert zu klein ist, kann die Konvergenz zum Ideal sehr lange dauern. Ein zu hoher Wert kann das Ideal überschießen oder sogar dafür sorgen, dass die Lernkurve divergiert. He et al. nutzen in ihrer Ausarbeitung einen Wert von 0.02.[15] Dieser sorgt jedoch laut den Entwicklern von Matterport zu einem explosionartigen Anstieg der Gewichte, weswegen sie sich für 0.001 entschieden haben.[1] Hier werden beide Werte untersucht.

**Werte:** 0.001, 0.2

**Parameterbezeichnung:** NUM\_CLASSES

**Erklärung:** Die Anzahl der Klassen, die trainiert werden. Hier ist darauf zu achten, dass der Hintergrund als eigene Klasse gesehen wird, auch wenn diese nicht explizit definiert wird. Daher muss die zusätzliche Klasse mit in diesen Wert einberechnet werden.

**Werte:** 2

**Parameterbezeichnung:** RPN\_ANCHOR\_SCALES

**Erklärung:** Quadratgröße der RPN-Anker. Die Werte sollten kleiner sein als IMAGE\_MAX\_DIM. Anker, die größer sind, machen keinen Sinn, da Objektinstanzen sich innerhalb der Bildgrenzen befinden.

**Werte:** (8, 16, 32, 64, 128), (4, 8, 16, 32, 64)

**Parameterbezeichnung:** STEPS\_PER\_EPOCH

**Erklärung:** Anzahl der Trainingsschritte bis eine Epoche abgeschlossen ist. Der Wert ist abhängig von der Größe des Trainingdatensatzes und wie viele Bilder pro Schritt prozessiert werden.

**Werte:**  $\frac{SIZE_{Train}}{IMAGES\_PER\_GPU}$

**Parameterbezeichnung:** USE\_MINI\_MASK

**Erklärung:** Wenn dieser bool'sche Wert wahr ist, werden Instanzmasken zu einer spezifizierten Größe verkleinert. Hier wird das jedoch deaktiviert, da die Eingangsbilder klein genug sind und diese Operation nicht nötig ist.

**Werte:** False

**Parameterbezeichnung:** WEIGHT\_DECAY

**Erklärung:** Einflussgröße der L2 Regulization.

**Werte:** 0.01, 0.005, 0.001

Der Konfigurationsparameter BATCH\_SIZE wird automatisch aus  $GPU\_COUNT * IMAGES\_PER\_GPU$  berechnet. Während IMAGES\_PER\_GPU angibt, wie viele Bilder pro Rechnereinheit (GPU oder CPU) in das neuronale Netz geladen werden, definiert BATCH\_SIZE wie viele Bilder insgesamt pro Trainingsschritt geladen werden.

CropDiseaseConfig erbt von Config und die Werte werden vor jedem erneuteten Trainingsdurchlauf verändert und der Modell-Instanz übergeben.

### 4.6.3 Ablauf

Die Benutzer können bei Skriptaufruf als Kommandozeilenparameter spezifizieren, ob das neuronale Netz trainiert oder ob ein Bild überprüft werden soll. Für beide Fälle müssen sie spezifizieren, auf welcher Basis das neuronale Netz gebildet werden soll. Hier gibt es drei Optionen für diese Anwendung:

- vortrainierte COCO-Gewichte
- vortrainierte ImageNet-Gewichte
- Fortsetzen eines alten Trainingsdurchlaufs

Wählen die Benutzer eine der ersten beiden Möglichkeiten, werden die vortrainierten Gewichte heruntergeladen. Bei der dritten Option muss eine vorher gespeicherte Datei geladen werden, die die Gewichte des gewünschten Modells enthält. Aus den gewählten Gewichten und dem CropDiseaseConfig-Objekt wird nun eine Mask R-CNN-Instanz gebaut. Dieses Instanz kann eine von zwei Modi

- training
- inference

annehmen. Wobei *training* die Gewichte des Modells für das Training veränderbar macht und *inference* für die Detektion die Gewichte einfriert, da sie während einer Erkennung nicht trainiert werden müssen.

## **training**

Wie in Kapitel 4.6.1 beschrieben, ist für jeden Subdatensatz jeweils ein unterschiedliches `CropDiseaseDataset`-Objekt nötig. In den separaten Objekten werden nun die Daten, Klassen und Masken für das Training, die Validierung und das Testen geladen und vorbereitet.

Das Test-Objekt wird nur indirekt für das Training benutzt. Es hat keinen direkten Einfluss auf den Trainingsverlauf, sondern wird genutzt, um den mAP am Ende jeder Epoche zu berechnen. Hierzu ist eine weitere Mask R-CNN-Instanz im *inference*-Modus notwendig, dessen Gewichte nach jeder Epoche auf die selben Werte des *training*-Modells aktualisiert werden. Die Konfiguration definiert die von `CropDiseaseConfig` abgeleitete Klasse `InferenceConfig`. Anders als die Parameter von `CropDiseaseConfig` werden die Werte nur einmalig festgelegt.

```
class InferenceConfig(CropDiseaseConfig):
    # Nur ein Bild pro Detektion erlauben
    IMAGES_PER_GPU = 1
    GPU_COUNT = 1
    # Jede Erkennung wird angezeigt
    DETECTION_MIN_CONFIDENCE = 0.0
```

Listing 4.2: InferenceConfig

Auf Grund des *inference*-Modells und des Testdatensatzes kann nun am Ende einer Epoche der mAP berechnet werden, mit dessen Hilfe am Ende des Trainings das neuronale Netz evaluiert werden kann. Um Vergleichswerte zu haben, wird dieser für Trainings- und Validierungsdatensatz ebenfalls berechnet.

Bevor das Training starten kann, müssen noch die Anzahl der Epochen und die trainierbaren Schichten angegeben werden. Die Anzahl der Epochen hat ein

Einfluss darauf, wie oft ein Training wiederholt wird. Wird der Wert zu niedrig gewählt, kann es sein, dass die Gewichte nicht ausreichend genug trainiert sind. So riskiert man *Underfitting*. Ist der Wert jedoch zu hoch, werden die Gewichte zu sehr an den Trainingsdatensatz angepasst und ein *Overfitting* ist die Folge. Mit dem *training*-Modus gibt man zwar an, dass die Gewichte variabel sind, aber durch die hier angegebenen Schichten wird genau festgelegt, welche Schichten eingefroren werden und welche nicht. So wird mit `layers='heads'` lediglich der *network head* trainiert, werden alle Schichten mit `layers='all'` trainierbar sind. Bei einem kleinen Datensatz wäre es zum Beispiel sinnvoll, nur die Klassifikatoren zu trainieren, die sich im Netzwerkkopf befinden. Zusätzlich können verschiedene Schichten iterativ trainiert werden. Zum Beispiel wird in den ersten 20 Epochen der *head* trainiert. Anschließend werden alle Schichten für 80 Epochen trainiert.

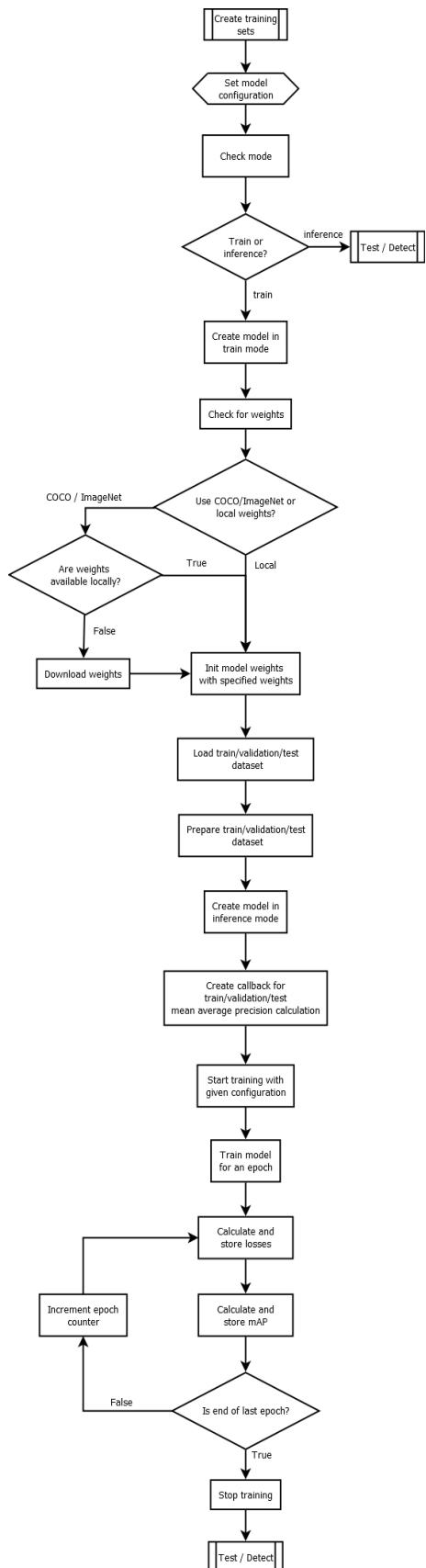


Abbildung 4.7: Ablaufdiagramm des Trainings

## inference

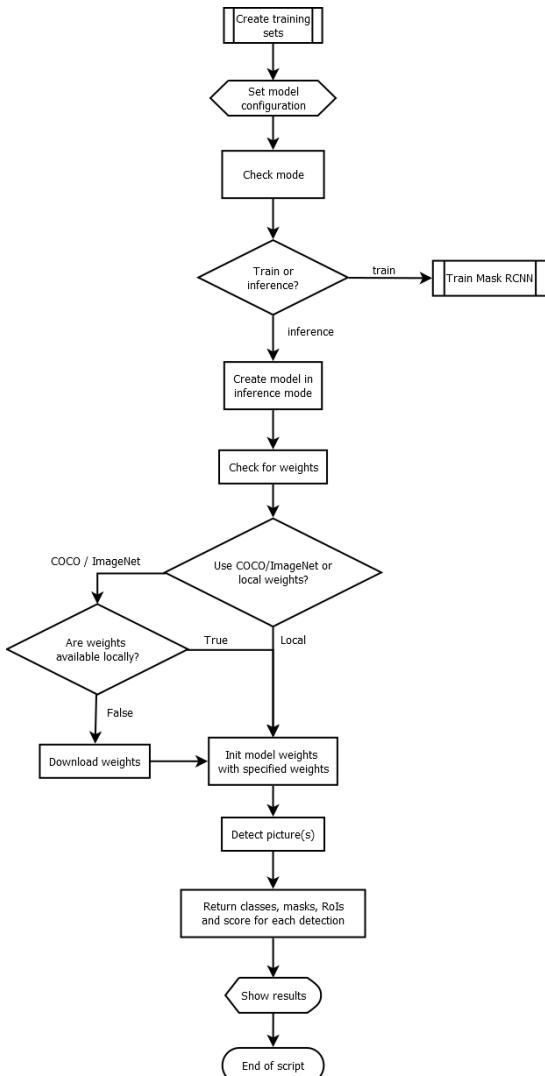


Abbildung 4.8: Ablaufdiagramm der Erkennung

Im *inference*-Mdous wird das Mask R-CNN-Objekt durch `InferenceConfig` konfiguriert. Wenn die angegebenen Gewichte in das Modell geladen wurden, können  $n$  Bilder, wobei  $n \geq 1$  sei, zur Detektion in das Netzwerk gegeben werden, ohne dass eine spezielle `Dataset`-Instanz vonnöten ist. Es ist wichtig, dass das die NDVI-Werte vor der Detektion zu RGB-Werten konvertiert werden, wie es in Kapitel 4.6.1 beschrieben ist.

Nach der Detektierung mit möglichen Objekten gibt das Modell eine Liste mit  $n$  Elementen zurück. Die Elemente dieser Listen bestehen aus  $m$  vorhergesagten *RoIs*, Masken und Klassen der jeweiligen Bilder, wobei  $m \geq 0$  sei. Diese sind in separaten Listen abgelegt und sind so angeordnet, dass die jeweiligen

Elemente der Listen an Position  $i$  zusammengehören, wobei  $0 \leq i < m$  sei. So gehören die  $i$ -te *RoI*, Klasse und Masken zu der selben vorhergesagten Objektinstanz. Zusätzlich liegt der Instanz ein Wert zwischen 0 und 1 bei, die die Wahrscheinlichkeit der Korrektheit der Vorhersage angibt.

# Kapitel 5

## Experimente

Nach der Beschreibung des Konzept und Verlauf des Python-Skripts geht dieses Kapitel auf die wichtigsten Trainingsdurchläufe ein und diskutiert die Ergebnisse.

Es wurden iterativ Trainingprozeduren durchgeführt mit jeweils unterschiedlichen Konfiguration, wie um vorherigen Kapitel gezeigt, und miteinander verglichen. Die genauen Konfigurationen werden bei den einzelnen Experimenten angeführt. Die Experimente liefen auf einem Rechner mit NVIDIA TITAN Xp 12 GB VRAM Grafikkarte, Intel i7-7800X CPU und 32 GB RAM. Da diese Arbeit zeitlich begrenzt ist, wurden die Experimente auf max. 100 Epochen begrenzt, was die durchschnittliche Laufzeit auf etwa drei bis vier Stunden je Experiment bringt. Diese Beschränkung wirkt ebenfalls Overfitting entgegen, da nicht ausreichend Zeit hat, um sich auf den Trainingsdatensatz „einzugewöhnen“.

Es wurden drei unterschiedliche Datensätze aus den Sentinelprodukten erzeugt.

1. Der originale unaugmentierte Datensatz enthält insgesamt jeweils zwölf Bilder und Masken<sup>1</sup> und dient als Basis für das Ausgangsexperiment.
2. Der augmentierte Datensatz (s. Kapitel 3.2) enthält 1291 Dateien und Masken<sup>2</sup>. Nachdem die zufällige Ausschnitte produziert wurden, kam es vor, dass einige Masken keinen *RoI*-Anteil enthielten. Diese Masken und

---

<sup>1</sup>Trainingsanteil: 7, Validationsanteil: 2, Testanteil: 3

<sup>2</sup>Trainingsanteil: 825, Validationsanteil: 207, Testanteil: 259

auch die zugehörigen Bilddateien wurden verworfen und weicht deswegen von der eigentlichen Größe von 1296 Elementen<sup>3</sup> ab.

3. In einem dritten Datensatz wurden Ausschnitte benutzt, deren Bildgrenzen sich an den äußersten Punkten der Masken befinden (s. Abb. 5.5). Dieser Datensatz wurde ausschließlich mittels Rotationen erweitert und enthält 144 Elemente<sup>4</sup>.

## Experimente 1

```
class CropDiseaseConfig(Config):
    BACKBONE = "resnet50"
    IMAGE_MAX_DIM = 128
    IMAGE_MIN_DIM = 128
    IMAGE_RESIZE_MODE = "square"
    IMAGES_PER_GPU = 4
    LEARNING_RATE = 0.001
    NUM_CLASSES = 1 + 1
    RPN_ANCHOR_SCALES = (8, 16, 32, 64, 128)
    STEPS_PER_EPOCH = 3
    USE_MINI_MASK = False
```

Listing 5.1: Konfiguration für Experimente 1

Zuerst wurden ein Modell auf Datensatz 1 und alle Schichten des Modells wurden trainiert. Diese Experimente sollen zeigen, wie ein zu kleiner Datensatz sich auswirken kann.



Abbildung 5.1: *mAP*-Graph von Experiment 1, X-Achse: Epochennummer, Y-Achse: *mAP*-Werte

---

<sup>3</sup>Die zwölf ursprünglichen Dateien multipliziert mit dem Data Augmentation-Faktor von 108.

<sup>4</sup>Trainingsanteil: 92, Validationsanteil: 23, Testanteil: 29

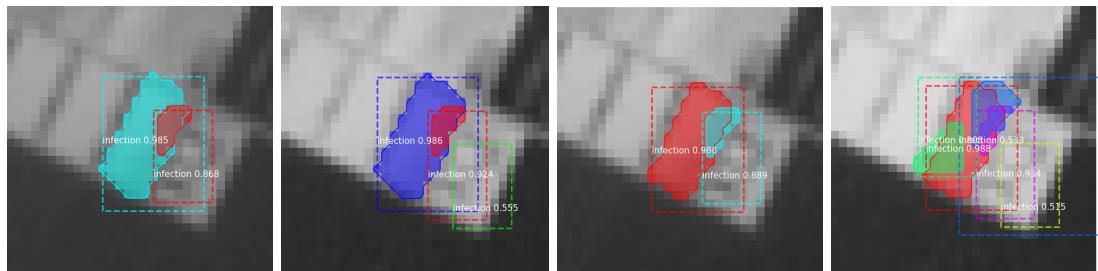


Abbildung 5.2: Beispielvorhersagen anhand der Gewichte von Epoche 55

Man sieht, dass das Modell während des Trainings unterdurchschnittliche Ergebnisse erzielt, was wenig überraschend ist, da der Datensatz sehr klein ist. Da nach jeder Epoche die Gewichte zwischengespeichert werden, können diese einzeln geladen werden. So wurde für eine Detektion ein Modell mit den Gewichten der 55. Epoche initialisiert, da sich hier der *mAP*-Wert auf dem Maximalwert befindet. Die Bilder in Abb. 5.2 entstammen einem fremden Datensatz 1. Zum Beispiel ist das Zielfeld ähnlich ausgerichtet und zentriert. Man sieht, dass sich die berechneten Masken relativ genau auf das Zielfeld eingrenzen. Jedoch sind die erzeugten Bounding-Boxen und passen nicht zu den entsprechenden Masken, falls eine Maske erkannt wurde. Auch wurden mehrere Objektinstanzen erkannt, wobei eine einzige Instanz detektiert werden sollte. Werden die Bilder in Abb. 5.2 nun vertikal gespiegelt und in das Netzwerk gegeben, erzeugt das Modell keine Masken, obwohl lediglich die Ausrichtung verändert wurde. Das ist ein Hinweis auf Overfitting, da das neuronale Netz minimale Änderungen nicht mehr erkennt.

## Experimente 2

```
class CropDiseaseConfig(Config):
    BACKBONE = "resnet50"
    IMAGE_MAX_DIM = 128
    IMAGE_MIN_DIM = 128
    IMAGE_RESIZE_MODE = "square"
    IMAGES_PER_GPU = 4
    LEARNING_RATE = 0.001
    NUM_CLASSES = 1 + 1
    RPN_ANCHOR_SCALES = (8, 16, 32, 64, 128)
    STEPS_PER_EPOCH = 3
    USE_MINI_MASK = False
```

Listing 5.2: Konfiguration für Experimente 2

Bei diesem Experiment wurde ein Modell Datensatz 2 und alle Schichten des Modells trainiert. Die Konfiguration bleibt unverändert, jedoch ist hier der augmentierte Datensatz vergleichweise größer.

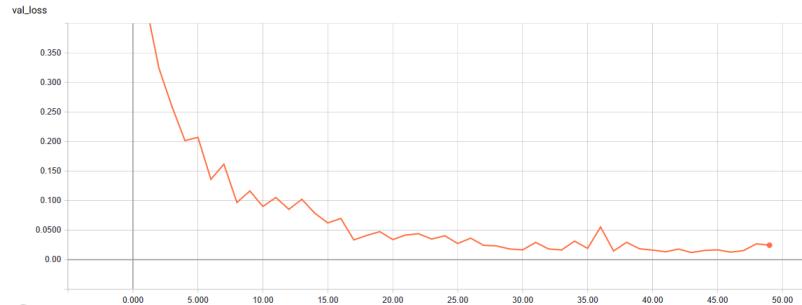


Abbildung 5.3: *loss*-Graph von Experiment 2, X-Achse: Epochennummer, Y-Achse: *loss*-Werte

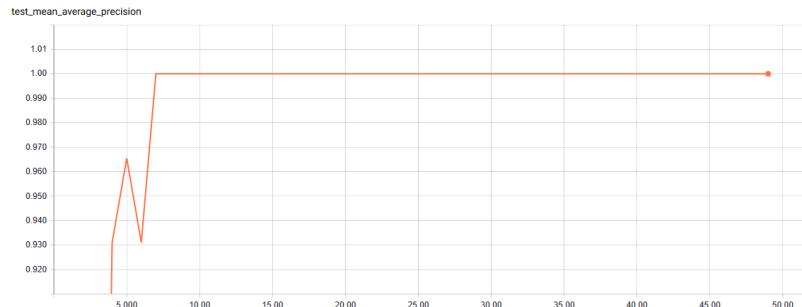


Abbildung 5.4: *mAP*-Graph von Experiment 2, X-Achse: Epochennummer, Y-Achse: *mAP*-Werte

Die *mAP*-Kurve konvergiert ab Epoche 7 gegen 1 und verweilt dort. Dagegen sinken die *loss*-Werte weiterhin

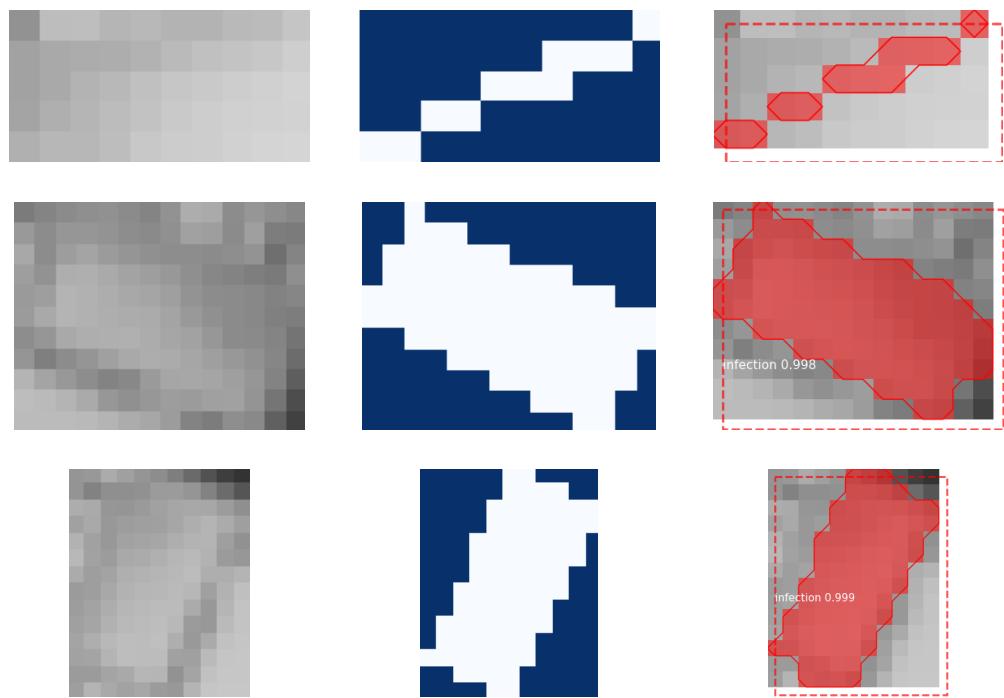


Abbildung 5.5: Beispielvorhersagen anhand der Gewichte von Epoche 37

# **Kapitel 6**

## **Zusammenfassung und Fazit**

# **Kapitel 7**

## **Ausblick**

# Abbildungsverzeichnis

2.1 Region of Interest . . . . .	4
2.2 Künstliches neuronales Netz . . . . .	8
2.3 CNN . . . . .	9
2.4 Instanzsegmentierung . . . . .	10
2.5 Mask R-CNN-Architektur . . . . .	10
2.6 RPN-Anker . . . . .	11
2.7 FCN-Architektur . . . . .	12
2.8 Mask R-CNN vs. FCIS . . . . .	13
2.9 IoU . . . . .	15
2.10 Precision-Recall-Kurve . . . . .	16
3.1 Beispiel Overfitting . . . . .	19
4.1 Gesamtablauf der Anwendung . . . . .	22
4.2 Ablaufdiagramm Sentineldatenaufbereitung . . . . .	25
4.3 Ablaufdiagramm der Aufbereitung . . . . .	27
4.4 B4 - B8 - NDVI . . . . .	28
4.5 Ablaufdiagramm der Datensatzaufteilung . . . . .	29
4.6 Datensatzverteilung . . . . .	30
4.7 Ablaufdiagramm des Trainings . . . . .	37
4.8 Ablaufdiagramm der Erkennung . . . . .	38
5.1 <i>mAP</i> -Graph von Experiment 1, X-Achse: Epochennummer, Y-Achse: <i>mAP</i> -Werte . . . . .	41
5.2 BeispieldatenExperiment 1 . . . . .	42
5.3 <i>loss</i> -Graph von Experiment 2, X-Achse: Epochennummer, Y-Achse: <i>loss</i> -Werte . . . . .	43
5.4 <i>mAP</i> -Graph von Experiment 2, X-Achse: Epochennummer, Y-Achse: <i>mAP</i> -Werte . . . . .	43
5.5 BeispieldatenExperiment 2 . . . . .	44

# **Tabellenverzeichnis**

2.1 Räumliche und spektrale Auflösungen von Sentinel-2A[12] . . . . .	6
2.2 Mask R-CNN im Vergleich . . . . .	13

# Literaturverzeichnis

- [1] W. Abdulla. Mask r-cnn for object detection and instance segmentation on keras and tensorflow. [https://github.com/matterport/Mask\\_RCNN](https://github.com/matterport/Mask_RCNN), 2017. [Zuletzt besucht: 06.01.2019].
- [2] W. Abdulla. Splash of color: Instance segmentation with mask r-cnn and tensorflow. <https://engineering.matterport.com/splash-of-color-instance-segmentation-with-mask-r-cnn-and-tensorflow-7c761>, 2018. [Zuletzt besucht: 22.12.2018].
- [3] Anuja Nagpal. L1 and l2 regularization methods. <https://towardsdatascience.com/l1-and-l2-regularization-methods-ce25e7fc831c>, 2017. [Zuletzt besucht: 09.01.2019].
- [4] T. Arlen. Understanding the map evaluation metric for object detection. <https://medium.com/@timothycarlen/understanding-the-map-evaluation-metric-for-object-detection-a07fe6962cf3>, 2018. [Zuletzt besucht: 28.01.2019].
- [5] Bharath Raj. Data augmentation | how to use deep learning when you have limited data - part 2. <https://medium.com/nanonets/how-to-use-deep-learning-when-you-have-limited-data-part-2-data-augmentation-2018-2a2f3>, 2018. [Zuletzt besucht: 09.01.2019].
- [6] H. Butler, M. Daly, A. Doyle, S. Gillies, S. Hagen, and T. Schaub. The GeoJSON Format. RFC 7946, IETF, August 2016.
- [7] C. Consortium. Coco 2018 object detection task. <http://cocodataset.org/#detection-2018>, 2018. [Zuletzt besucht: 06.01.2019].
- [8] C. Consortium. Detection evaluation. <http://cocodataset.org/#detection-eval>, n.d. [Zuletzt besucht: 28.01.2019].

- [9] Copernicus. Copernicus in brief. <https://www.copernicus.eu/en/about-copernicus/copernicus-brief>. [Zuletzt besucht: 15.12.2018].
- [10] N. Corporation. Convolutional neural network (cnn). <https://developer.nvidia.com/discover/convolutional-neural-network>, 2019. [Zuletzt besucht: 04.01.2019].
- [11] ESA. Level-2a algorithm overview. <https://sentinel.esa.int/web/sentinel/technical-guides/sentinel-2-msi/level-2a/algorithm>, 2018. [Zuletzt besucht: 20.12.2018].
- [12] ESA. Radiometric resolutions. <https://earth.esa.int/web/sentinel/user-guides/sentinel-2-msi/resolutions/radiometric>, 2018. [Zuletzt besucht: 20.12.2018].
- [13] ESA. Resolutions. <https://earth.esa.int/web/sentinel/user-guides/sentinel-2-msi/resolutions>, 2018. [Zuletzt besucht: 20.12.2018].
- [14] A. A. Gitelson, Y. Gritz, and M. N. Merzlyak. Relationships between leaf chlorophyll content and spectral reflectance and algorithms for non-destructive chlorophyll assessment in higher plant leaves. *Journal of Plant Physiology*, 160(3):271 – 282, 2003.
- [15] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick. Mask R-CNN. *CoRR*, abs/1703.06870, 2017.
- [16] G. A. F. Hendry, J. D. HOUGHTON, and S. B. BROWN. The degradation of chlorophyll — a biological enigma. *New Phytologist*, 107(2):255–302, 1987.
- [17] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, and K. Murphy. Speed/accuracy trade-offs for modern convolutional object detectors. *CoRR*, abs/1611.10012, 2016.
- [18] J. Hui. Image segmentation with mask r-cnn. [https://medium.com/@jonathan\\_hui/image-segmentation-with-mask-r-cnn-ebe6d793272](https://medium.com/@jonathan_hui/image-segmentation-with-mask-r-cnn-ebe6d793272), 2018. [Zuletzt besucht: 06.01.2019].
- [19] J. Hui. map (mean average precision) for object detection. [https://medium.com/@jonathan\\_hui/](https://medium.com/@jonathan_hui/)

- map-mean-average-precision-for-object-detection-45c121a31173, 2018. [Zuletzt besucht: 25.01.2019].
- [20] Jeremy Jordan. Evaluating image segmentation models. <https://www.jeremyjordan.me/evaluating-image-segmentation-models/>, 2018. [Zuletzt besucht: 27.01.2019].
- [21] C. Mattupalli, C. A. Moffet, K. N. Shah, and C. A. Young. Supervised classification of rgb aerial imagery to evaluate the impact of a root rot disease. *Remote Sensing*, 10(6), 2018.
- [22] NASA. Measuring vegetation (ndvi & evi). [https://earthobservatory.nasa.gov/features/MeasuringVegetation/measuring\\_vegetation\\_2.php](https://earthobservatory.nasa.gov/features/MeasuringVegetation/measuring_vegetation_2.php), 2000. [Zuletzt besucht: 13.12.2018].
- [23] Prashant Gupta. Regularization in machine learning. <https://towardsdatascience.com/regularization-in-machine-learning-76441ddcf99a>, 2017. [Zuletzt besucht: 09.01.2019].
- [24] S. Ren, K. He, R. B. Girshick, and J. Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015.
- [25] T. Shah. About train, validation and test sets in machine learning. <https://towardsdatascience.com/train-validation-and-test-sets-72cb40cba9e7>, 2017. [Zuletzt besucht: 16.01.2019].
- [26] U.S. Department of the Interior. What are the band designations for the landsat satellites? <https://sentinel.esa.int/web/sentinel/technical-guides/sentinel-2-msi/level-2a/algorithms>, n.d. [Zuletzt besucht: 24.12.2018].
- [27] J. Verrelst, J. Muñoz, L. Alonso, J. Delegido, J. P. Rivera, G. Camps-Valls, and J. Moreno. Machine learning regression algorithms for biophysical parameter retrieval: Opportunities for sentinel-2 and -3. *Remote Sensing of Environment*, 118:127 – 139, 2012.

# **Erklärung**

Ich versichere, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe, insbesondere sind wörtliche oder sinngemäße Zitate als solche gekennzeichnet. Mir ist bekannt, dass Zu widerhandlung auch nachträglich zur Aberkennung des Abschlusses führen kann.

Ich versichere, dass das elektronische Exemplar mit den gedruckten Exemplaren übereinstimmt.

Ort:

Datum:

Unterschrift: