Masterthesis

# Deep Learning zur Grassegmentierung im Arbeitsraum eines Rasenmähroboters

Oliver Gruhlke

2 7 8 7 3 6

ogruhlke@uni-bremen.de

09. August 2018

Erstgutachter Prof. Dr. Udo Frese
Zweitgutachter Dr. Felix Putze

Universität Bremen

# Eidesstattliche Erklärung

Hiermit versichere ich, die vorliegende Arbeit selbstständig und nur unter Zuhilfenahme der angegebenen Quellen und Hilfsmittel verfasst zu haben. Die aus fremden Quellen direkt oder indirekt übernommenen Stellen sind als solche kenntlich gemacht.

Ich bestätige außerdem, dass die vorliegende Arbeit noch nicht im Rahmen eines anderen Prüfungsverfahrens eingereicht wurde.

# Zusammenfassung

In dieser Masterarbeit wird ein Deep Learning Ansatz entwickelt, der einem Rasenmähroboter eine selbstständige Erfassung des Arbeitsraums ermöglicht. Hierfür sollen Eingabebilder pixelgenau in mähbar und nicht-mähbar segmentiert werden. Es wird ein Datenset mit Bildern aus der Perspektive eines Rasenmähroboters als Datengrundlage erstellt.
Für die Arbeit wird ein Deep Learning Ansatz aus der aktuellen Forschung ausgewählt, weiterentwickelt und evaluiert, um die aufgeworfene Problemstellung zu lösen.

# Abstract

In this master thesis a deep learning approach for a robotic lawn mower will be developed. With a binary segmentation the mower can capture the workspace own its own. As a Data basis a whole new Dataset will be recorded from a theoretical mower perspective. The deep learning approach will be selected from the current research, further developed and evaluated to solve the raised problem.

# Inhaltsverzeichnis

# Listingverzeichnis

# Abbildungsverzeichnis

# Gleichungsverzeichnis

# Einleitung

Diese Masterarbeit behandelt die Implementierung einer Deep Learning Methode zur Grassegmentierung in unterschiedlichen Gärten. Die Grundlage des Verfahrens bilden Bilder aus der typischen Perspektive eines Rasenmähroboters. Es soll pixelgenau zwischen Flächen, auf denen der Roboter mähen kann und solchen, die er zu meiden hat, unterschieden werden. Zur Lösung des Problems muss zunächst eine geeignete Methode ausgewählt, diese dann implementiert und evaluiert werden. Da es kein Datenset mit Bildern aus der entsprechenden Perspektive gibt, wird im Rahmen dieser Arbeit ein eigenes Datenset aufgebaut und mit einer semi-automatischen Methode segmentiert.

## 1.1 Motivation

Aktuell nutzen handelsübliche Rasenmähroboter keine komplexe Sensorik, um ihren Arbeitsbereich eigenständig zu erfassen. Es gibt lediglich einen Stoßsensor, um bei Hindernissen abzudrehen und eine Magnetspule, um den Begrenzungsdraht zu detektieren. Einige Nachteile treten durch diese stark begrenzte Sensorik auf. Zunächst gibt es das Problem, dass der Begrenzungsdraht im Garten verlegt werden muss. Je nach Beschaffenheit des Gartens ist dies eine langwierige Aufgabe, da der Draht exakt verlegt werden muss. Ebenso muss bei einer Änderung der Gartenarchitektur der Draht, der bereits nach einigen Monaten vergraben ist, neu verlegt werden.
Aktuellere Ansätze setzen auf einen sogenannten Rasensensor, der selbstständig Rasen erkennen soll. Jedoch kann diese Art Sensor nur den Untergrund klassifizieren, auf dem der Rasenmäher im Moment mäht.
Ein Beispiel für einen Rasensensor ist der *ZGS-Sensor* von F. Bernini [Ber09]. Das Konzept, das Bernini in seinem Patent aufzeigt, beschreibt einen Rasenmähroboter mit mehreren Sensoren an der Unterseite. Diese Sensoren sollen Objekte aller Art sowie die Distanz zu diesen, detektieren können, wobei die Sensorart nicht genauer definiert ist.
Aufgrund der Informationen wird anhand von Schwellenwerten versucht, zwischen Hindernissen, Rasenflächen und anderen Flächen zu unterscheiden. Hindernisse müssen eine gewisse Höhe vor dem Mäher haben, um detektiert zu werden. Somit kann das Ende einer Rasenfläche nur durch eine Begrenzung, wie Steinen, erkannt werden. Beete ohne Begrenzung werden überfahren.
Die Analyse der Fläche unterhalb des Rasenmähers wird über die Höhe im Vergleich zur Bodenebene durchgeführt. Ist die Höhe über einem bestimmten Schwellenwert befindet sich der Mäher auf zu mähenden Gras, ist sie unterhalb der Schwelle wird nicht gemäht. Diese Sensorik ermöglicht einem Roboter zwar das gezielte Mähen der Grasfläche, jedoch keine effektive Vorausplanung, da das System nur einem kleinen begrenzten Raum um den Roboter Informationen liefert.

Auch die Hindernisvermeidung ausschließlich durch Stoßsensoren ist problematisch, da flache Gegenstände, wie z.B. liegengebliebene Kleidung, überfahren werden.
Ein weiterer Nachteil der Sensorik ist, dass keine komplexe Wegfindung umgesetzt werden kann. Ein effektives Mähen wird oft nur durch die Häufigkeit des Mähens und zufällige Steuerung erreicht. Schwer erreichbare Ecken können bei unglücklicher Steuerung selten bis gar nicht erreicht werden.
Durch eine selbstständige Detektion mähbarer Flächen anhand einer Kamera können diese Nachteile ausgebessert werden. Es muss kein Draht mehr verlegt werden, Hindernisse können gezielt umfahren und eine komplexe Steuerung, die den gesamten Garten bearbeitet, kann implementiert werden.
Nähme man noch einen Raddrehsensor hinzu, lässt sich durch visuelle Odometrie auch ein Modell des Gartens erstellen und weitaus komplexere Algorithmen können verwendet werden.

## 1.2 Grundidee

Bereits bei der Themenfindung wurde ein Deep Learning Verfahren zur Lösung des Problems angedacht. Im State of the Art Kapitel 3 wird aufgezeigt, welche Erfolge aktuelle Forschungen mit Deep Learning Verfahren erzielen. Ein großer Vorteil, neben der herausragenden Genauigkeit, ist, dass beim Deep Learning komplexe Features durch einfache Konzepte abgebildet werden. Weiterhin müssen diese Features nicht von einem Entwickler definiert, sondern können selbstständig erlernt werden. Gerade dieser Vorteil ist essenziell, da es beispielsweise sehr schwierig ist, eine Menge an Features zu definieren, die Rasenflächen von angrenzenden Büschen unterscheiden.
Das ausgewählte Verfahren soll eine binäre Segmentierung des Eingabebildes liefern. Es soll lediglich zwischen mähbarer und nicht-mähbarer Fläche unterschieden werden. Weitere semantische Klassifizierungen, wie zum Beispiel Hindernisse, sind nicht vorgesehen. Hierfür wird mit einem Datenset aus verschiedenen Gärten mit hoch aufgelösten Bildern einer GoPro-Kamera trainiert. Es werden Bilder der Größe 3000x4000 Pixel verwendet. Eine Segmentierung mit solch großen Bildern ermöglicht einer hypothetischen Robotersteuereinheit eine sehr exakte Bestimmung von Ziel- und Wegpunkten.
Da das Szenario eine Anwendung des Verfahrens auf einem Rasenmähroboter während der Arbeit vorsieht, muss ebenfalls auf eine Echtzeitfähigkeit der Segmentierung geachtet werden. Was genau der Begriff *Echtzeit* für diese Arbeit bedeutet wird im Grundlagenkapitel 2.4 genauer erläutert.
Schwierigkeiten der Segmentierung bilden vor allem das unterschiedliche Aussehen der Gärten und die unterschiedlichen Licht- und Schattenverhältnisse. Aus diesem Grund muss das Datenset in möglichst unterschiedlichen Gärten und Bedingungen aufgenommen werden.

## 1.3 Aufbau der Arbeit

Zunächst werden einige Grundlagen zu Deep Learning Verfahren und *Convolutional Neural Network (CNN)* erläutert. Es wird jedoch grundlegendes Vorwissen vorausgesetzt, da eine vollständige Erklärung den Rahmen der Arbeit überschreiten würde. Außerdem werden Rahmenbedingungen der Arbeit beschrieben.
Danach werden verschiedene Herangehensweisen aus ähnlichen Arbeiten vorgestellt und die Eignung für das gegebene Problem diskutiert. Im Anschluss wird die Erstellung des Datensets sowie seine Inhalte beschrieben.
Im Folgenden wird das ausgewählte Verfahren vorgestellt, die Implementierung, das Trainingsverfahren und die durchgeführten Tests beschrieben. Abschließend folgt dann ein Fazit und ein Ausblick auf mögliche Folgearbeiten.

# Grundlagen

In diesem Kapitel werden Grundlagen zum allgemeinen Verständnis der Arbeit erläutert. Zunächst wird das Prinzip des Deep Learning sowie *Convolutional Neural Network (CNN)* eingeführt. Ebenfalls wird auf verwendete *Conditional Random Field (CRF)* und Inferenz eingegangen. Hierfür soll eine kurze Einführung gegeben, jedoch keine ausschöpfenden theoretischen Hintergründe aufgezeigt werden.
Am Ende des Kapitels werden Rahmenbedingungen bezüglich des Szenarios der Arbeit erläutert.

## 2.1 Deep Learning

Als Deep Learning wird ein Gebiet des *Machine Learnings* bezeichnet, dass sich durch besonders vielschichtige Modelle auszeichnet. Die bekanntesten Modelle sind dabei tiefe neuronale Netze. Die Tiefe der Netze wird anhand der Anzahl der Hidden Layer gemessen und kann sehr stark variieren (von ca. zehn bis zu mehreren hundert).
Der Vorteil solch tiefer Netze ist, dass komplexe Probleme mit hoher Genauigkeit gelöst und sehr komplexe Features verwenden werden können, ohne diese per Hand definieren zu müssen. Es wird jedoch ein sehr großes und variables Datenset sowie eine meist lange Trainingszeit benötigt.
Wichtig ist die Auswahl des Modells. Ist das Modell zu komplex, läuft man in die Gefahr des *Overfitting*, das sich auch als *Auswendiglernen* der Daten beschreiben lässt. Beim *Overfitting* wird ein Modell gelernt, dass zwar auf den Trainingsdaten sehr gute Ergebnisse erzeugt, jedoch nicht generell genug ist, um unbekannte Daten zu verarbeiten, da es sich komplett den Trainingsdaten angepasst hat. *Overfitting* lässt sich durch ein Sinken des Trainingsfehlers mit gleichzeitigem Anstieg des Validierungsfehlers erkennen. Um dem entgegen zu wirken, lässt sich das Modell regulieren, um ein generelles Ergebnis zu erhalten.
Ist das Modell nicht komplex genug, besteht die Möglichkeit des *Underfitting*, wobei das Modell die Komplexität des Problems nicht abbilden kann. Dies lässt sich durch allgemein schlechte Ergebnisse in den Trainingsdaten erkennen.
Aufgrund der meist komplexen Modelle im Themengebiet des Deep Learnings ist *Underfitting* eher nicht zu erwarten. *Overfitting* ist jedoch ein Problem, dass beachtet werden muss.
Ein erster Schritt um *Overfitting* entgegenzuwirken, ist der Einsatz sogenannter Dropout Layer innerhalb des *CNN* Aufbaus (Kapitel 2.1.1). In diesen Layern wird eine bestimmte Anzahl von Informationen entfernt. Ein solch geringer Informationsverlust verschlechtert das Ergebnis nicht, da einzelne Features in einem sehr komplexen Netz keine derart hohe Relevanz haben, dass ihr Fehlen zu einem falschen Endergebnis führt. Die Gefahr vom *Overfitting* wird jedoch verringert.

Die Aufteilung des Datensets in Trainings- und Testdatenset muss ebenfalls gut durchdacht werden. Für ein sinnvolles System müssen die zwei Datensets zwingend getrennt werden. Die Testdaten dürfen nicht während des Trainings verwendet werden, um diese als unbekannte Daten zu halten. Wird diese Regel missachtet, ist das Ergebnis nicht mehr repräsentativ für die Qualität des Modells, da im Anwendungsszenario auch keine Bilder des Livebetriebs zum Training verwendet werden können.
Ebenfalls muss das Trainingsset eine gewisse Diversität aufweisen. Sollten die Trainingsdaten zu einseitig sein, ist es wahrscheinlich, dass kein repräsentatives Modell gelernt werden kann und die Performance auf dem Testset schwächer ist.
Neben diesen beiden Datensets wird noch ein Validierungsset verwendet, das während des Trainings die Qualität des Modells bestimmt.

### 2.1.1 *Convolutional Neural Network (CNN)*

In der Bildverarbeitung wird oft ein *Convolutional Neural Network (CNN)* eingesetzt. Ein *CNN* ist eine spezielle Art eines neuronalen Netzes, bei dem Eigenschaften eines Bildes verwendet werden, um die Verbindungen zwischen den Layern zu reduzieren. So sind Bilder meist so aufgebaut, dass z.B. ein Pixel mit den Pixeln in seiner Nachbarschaft ein Objekt bildet. Bei einem Fully Connected Layer wäre jedes Pixel mit allen anderen des Bildes verbunden, auch wenn weit entfernte Pixel meist keine semantische Auswirkung aufeinander haben.
Im Gegensatz zu klassischen Fully Connected Layern werden zwei Convolutional Layer über einen Kernel verbunden, der jedes Pixel ausschließlich mit seiner direkten Nachbarschaft verbindet. Jede dieser Verbindungen erhält ein Gewicht. Der Kernel wird auf die gesamte Eingabe angewendet, um eine Ausgabe zu erzeugen. Dadurch wird die Anzahl der Gewichte pro Layer stark reduziert. Ein neuronales Netz heißt *convolutional* sobald es mindestens einen Convolutional Layer enthält.
In Gleichung 2.1 ist die Gleichung der Convolution aufgezeigt. Wie auch in der graphischen Erläuterung in Abbildung 2.1 zu sehen, wird durch die Formel ein Bereich des Bildes zunächst mit der Kernelmatrix elementweise multipliziert und dann addiert.

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n) K(m,n) \tag{2.1}$$

Gleichung 2.1: Grundformel der Convolution mit zweidimensionaler Eingabe $I$ und Kernel $K$. $i$ und $j$ sind dabei Indizes der Eingabe und $m$ und $n$ Indizes des Kernels.

In einem Convolutional Layer folgt auf die convolution Operation eine nichtlineare Aktivierungsfunktion. Die einzige in dieser Arbeit verwendete Aktivierungsfunktion ist die *Rectified Linear Unit (ReLU)* (Gleichung 2.2), bei der alle negativen Eingaben auf 0 gesetzt werden.

$$f(x) = max(0, x) \tag{2.2}$$

Gleichung 2.2: Aktivierungsfunktion $ReLU$ auf eine Eingabe $x$. Alle negativen Eingaben werden auf 0 gesetzt.

Im Trainingsprozess neuronaler Netze werden in jeder Iteration die Gewichte aller zu trainierenden Layer angepasst. Dafür wird versucht, den Fehler der Vorhersage während der Iteration zu nutzen, um den Fehler der nächsten Iteration zu verringern. Dieses Ziel wird allgemein am effizientesten mit der Backpropagation erreicht.
Zunächst muss hierfür eine Fehlerfunktion definiert werden. Die Fehlerfunktion wird im Rahmen dieser Arbeit die *softmax cross-entropy* Lossfunktion sein. Beim Softmax wird die gesamte Eingabe auf Werte zwischen 0 und 1 aufgeteilt, die sich auf 1 summieren. Die Verteilung der Softmax Funktion wird beim cross-entropy Loss eingesetzt, um die Distanz zwischen Vorhersage und Ground Truth zu bestimmen.

$$p_i = \frac{e^{a_i}}{\sum_{k=1}^{N} e_k^a} \tag{2.3}$$

$$L(y, p) = -\sum_i y_i log(p_i) \tag{2.4}$$

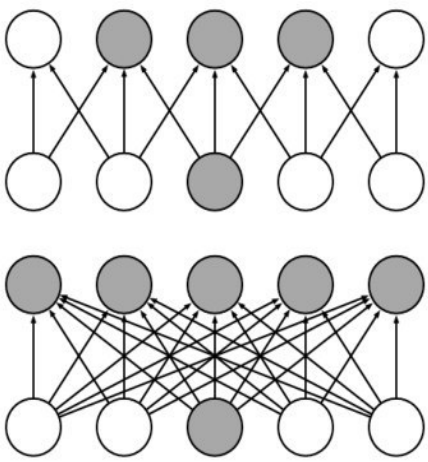
Gleichung 2.3: Formeln der Loss Berechnung. (2.1) zeigt die softmax Funktion, die beim cross-entropy (2.2) verwendet wird.

Diese Lossfunktion wird dann im Backpropagation Algorithmus eingesetzt, um die Gewichte zu optimieren.

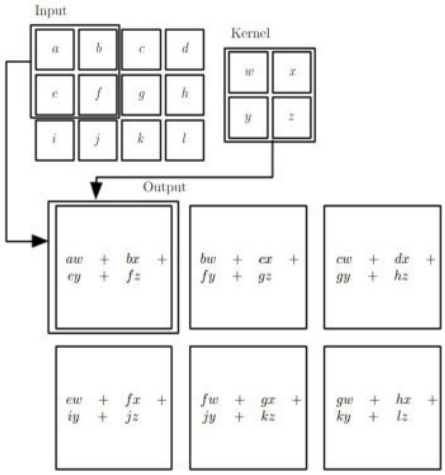
$$w_{t+1} = w_t - \alpha \Delta_{w_k} L \tag{2.5}$$

Gleichung 2.4: Anpassung der Gewichte $w$ während der Backpropagation. $\Delta_{w_k} L$ gibt an, wie die Gewichte eines Neurons geändert werden. Die Lernrate $\alpha$ reguliert diese Auswirkung.

In Abbildung 2.1 werden die Grundprinzipien von *CNNs* skizziert. Es ist deutlich zu sehen, dass die Verzweigung im *CNN* weitaus geringer ist, da jedes Pixel nur über den Kernel mit einigen Nachbarn verbunden ist und nicht mit jedem Pixel des Bildes. Dadurch wird sowohl die Modellgröße als auch die Rechenzeit des Vorwärtsschritts stark verringert.



(a) Einsparen der Verbindungen in den Layern. **Oben:** Ausgabe durch Convolution mit einem Kernel der Größe 3. **Unten:**Ausgabe durch Fully Connected Layer




(b) Prinzip der Kernelanwendung auf ein Eingabebild

Abbildung 2.1: Grundkonzepte von *CNN*. Grafiken aus Goodfellow et. al[GBC16]; Kapitel 9.

Auffällig an dieser Struktur ist die Ähnlichkeit zu herkömmlichen Filtern, die in der Bildverarbeitung eingesetzt werden. So ist ein typischer Filter zur Kantenextraktion (z.B. ein Sobel-Filter) ebenfalls eine 3x3 Matrix, die einem Kernel gleicht. Die Featureextraktion ist also auch durch die Kernelanwendung effizient möglich.
Weiterhin wird oft nach einigen Convolutional Layern ein Pooling Layer eingefügt. Ein Pooling Layer fasst stets Informationen aus einem bestimmten Gebiet zusammen und ersetzt diese. Eine weit verbreitete Pooling Operation ist das max-Pooling (siehe Abbildung 2.2), bei dem aus einem bestimmten Bereich ausschließlich das Maximum in die Ausgabe übernommen wird.
Ein Pooling Layer bietet mehrere Vorteile. Zum einen wird die Feature Map sehr stark verkleinert, was die Berechnungskomplexität verringert. Gleichzeitig wird durch das Pooling eine Invarianz gegenüber Translationen des Eingabebildes erreicht, da die gesamte Region betrachtet wird und ein Feature unabhängig von seiner Position im Bild erkannt wird. Der große Nachteil der Pooling Layer ist der erhebliche Informationsverlust. Beim max-Pooling bleibt ein Großteil der Features unbeachtet und eine geringer aufgelöste Featuremap verbleibt. Dies sorgt oft für eine ungenaue Segmentierung.

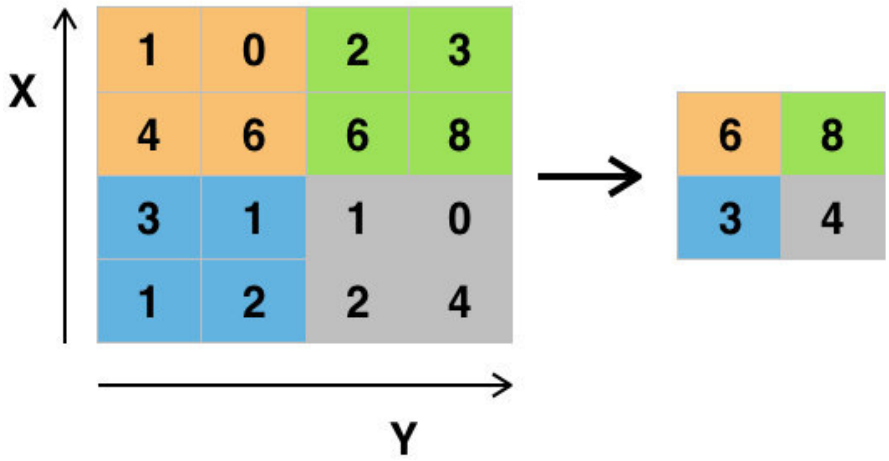



Abbildung 2.2: Ein Beispiel für eine max-Pooling Operation. In der 2x2 großen Ausgabe werden nur die Maxima der vier Bereiche der Eingabe übernommen. Grafik von Wikipedia[1]

Genau wie klassische neuronale Netze können auch *CNN* vielschichtig aufgebaut werden und somit dem Deep Learning angehören. Diese tiefen Netze werden im weiteren auch mit *Deep Convolutional Neural Network (DCNN)* abgekürzt.
Ein weit verbreitetes *DCNN* entwickeln Simonyan et. al [SZ14] 2014 mit dem VGG-16 (Abbildung 2.3) für die Bildklassifizierung. Das VGG-16 zeichnet sich durch 13 Convolutional Layer gemischt mit fünf Pooling Layern aus. Nach den Convolutional Layern folgen drei Fully Connected Layers zur Klassifizierung.
Das Netz dient aufgrund seiner guten Klassifizierungsergebnisse als Basis für einige Segmentierungsansätze und eignet sich deswegen besonders gut als Beispiel.

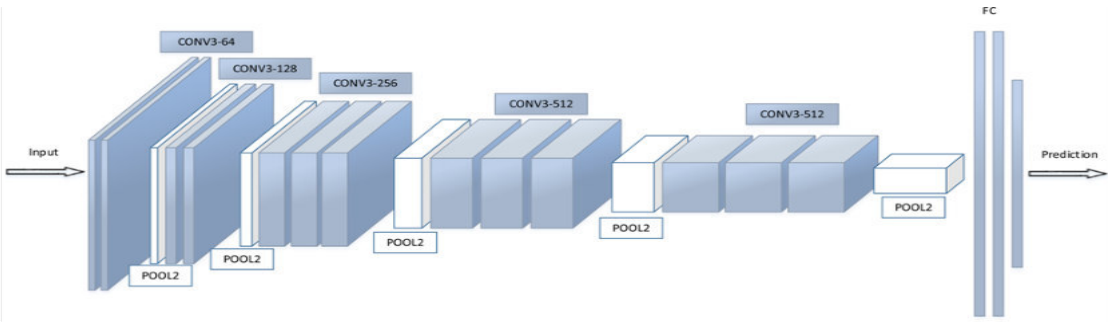



Abbildung 2.3: Aufbau des VGG-16 Netzes von Simonyan et. al [SZ14]. Grafik von El Khiyari et al.[2]

[1] Grafik von Wikipedia `https://en.wikipedia.org/wiki/File:Max_pooling`, Abrufdatum: 03.06.2018
[2] El Khiyari et al.: Face Recognition across Time Lapse Using Convolutional Neural Networks

### 2.1.2 Batchtraining

Bei einem batchbasierten Training werden mehrere Trainingsbeispiele (zu einem Batch) zusammengefasst und gleichzeitig trainiert. Dies hat zum Vorteil, dass ein Update der Gewichte mit einer größeren Anzahl Beispielen durchgeführt und somit genereller wird. Dadurch kann ein batchbasiertes Training zu einer schnelleren Konvergenz des Trainings führen.
Zusätzlich ermöglicht das Batchtraining die Batch Normalisierung [IS15]. Dabei wird nach jeder Aktivierungsfunktion über die gesamte Batch normalisiert.

```
def batchnorm(batch B, parameter (β,γ)):
  m = len(B)
  μ_B = 1/m Σ_{i=1}^{m} B[i]                          //Batch mean
  σ_B^2 = 1/m Σ_{i=1}^{m} (x_i − μ_B)^2               //Batch variance
  x̂_i = (x_i − μ_B) / sqrt(σ_B^2 + ε)                 //normalisieren
  return γx̂ + β                                       //Parameter anwenden
```

Listing 2.1: Pseudocode der Batch Normalisierung aus [IS15], einer Batch $B$ mit den trainierten Parametern $\beta$ und $\gamma$.

Dank der Batch Normalisierung kann das Training stark beschleunigt werden. Gleichzeitig lässt sich die Qualität des Modells verbessern. Die Normalisierung erzeugt regulierende Effekte innerhalb des Netzes, die Droupout Layer können reduziert werden, ohne ins *Overfitting* zu fallen und auch die Lernrate lässt sich erhöhen. All dies führt zu einem schnellen und effizienten Training.
Es muss bei der Batch Normalisierung jedoch beachtet werden, dass der Speicherbedarf beim Training drastisch erhöht wird. Ebenso muss die Größe aller Eingaben innerhalb eines Batches gleich sein. Für die Bildverarbeitung bedeutet dies, dass alle Bilder auf die gleiche Größe skaliert werden müssen. Angesichts der Vorteile, die in der Forschung mit der Batch Normalisierung erreicht werden, ist dieser Nachteil jedoch zu vernachlässigen.

### 2.1.3 Framework

Einen kompletten Deep Learning Ansatz von Grund auf selbst zu implementieren ist nicht im Rahmen der Arbeit realisierbar und wäre ebenfalls nicht zielführend, da bereits sehr gute Bibliotheken für *Machine Learning* existieren. Eine davon ist TensorFlow[3] vom Google Brain Team.

(http://www.scirp.org/JOURNAL/PaperInformation.aspx?PaperID=65406), Abrufdatum: 07.08.2018
[3]TensorFlow Website https://www.tensorflow.org/

TensorFlow bietet eine Implementierung aller wichtigen Layerarten, die benötigten Algorithmen (Aktivierungsfunktionen, Optimierungsfunktionen, Evaluierungsmaße, uvm.), sowie eine effiziente Backpropagation für die Fehlerkorrektur der Netze.
Die Bibliothek bietet eine, zwar komplexe, aber funktionsstarke Python-Schnittstelle und *NVIDIA Cuda®*[4] Implementierungen, die eine Auslagerung der Berechnungen auf die GPU ermöglichen.
Im Hintergrund wird eine C++-Implementierung verwendet, die auf *Tensoren* basiert. Alle Objekte innerhalb der TensorFlow-Programme werden als *Tensoren* dargestellt und untereinander verknüpft.
Die Bibliothek unterstützt das Im- und Exportieren von Modellen. Somit können existierende Modelle, die mit sehr viel größeren Datensets trainiert wurden, geladen und weiter verwendet werden.
Mit Tensorboard stellt TensorFlow eine webbrowserbasierte Visualisierung von Logdaten bereit (Abbildung 2.4). Während des Trainings oder der Evaluierung können eigene Daten in den Logdateien in bestimmten Intervallen gesichert werden. Außerdem wird der verwendete Graph dargestellt.

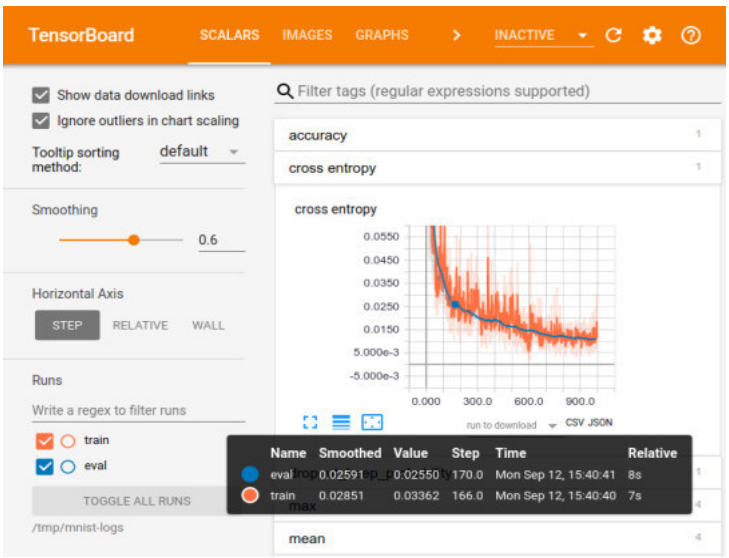



Abbildung 2.4: Beispielbild der Tensorboard Visualisierung von der TensorFlow Website[5]

Aufgrund dieser Komfortfunktionen und der Tatsache, dass eine aktuelle TensorFlow Implementierung für den als Grundlage der Arbeit ausgewählten Ansatz öffentlich verfügbar ist, wird die Bibliothek auch in dieser Arbeit verwendet.

[4] Cuda®Website http://www.nvidia.de/object/cuda-parallel-computing-de.html
[5] TensorFlow Website - Tensorboard: `https://www.tensorflow.org/programmers_guide/summaries_and_tensorboard`, Abrufdatum: 05.06.2018

## 2.2 *Conditional Random Field (CRF)*

Einige Ansätze (Siehe Kapitel 3) nutzen ein *CRF* - entweder während der Segmentierung oder zur Nachbearbeitung der Ergebnisse des Deep Learning Verfahren. Deshalb werden hier kurz die Grundlagen von *CRFs* erläutert.
*CRFs* sind probabilistische Modelle und gehen aus Markov Random Fields (MRF) und Hidden Markov Models (HMM) hervor. Ein *CRF* besteht aus einer Menge an Zufallsvariablen $X = \{X_1, ...X_n\}$ mit ihren möglichen Belegungen $L = \{l_1, ...l_k\}$, deren Abhängigkeiten in Cliquen $c$ dargestellt werden. Daraus folgt eine Darstellung von *CRFs* als Graph $G$.
Die für diese Arbeit relevante Spezialform der *Fully Connected Pairwise CRF* zeichnet sich dadurch aus, dass $G$ ein vollständiger Graph über $X$ ist und die Menge aller Cliquen $C_G$ alle einzelnen Variablen sowie deren paarweise Cliquen beinhaltet.

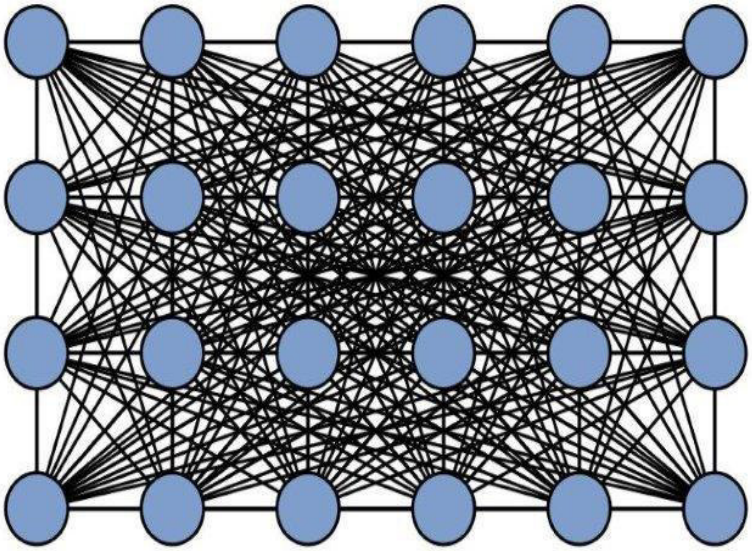


Abbildung 2.5: Beispiel eines Fully Connected *CRF*. Jeder Knoten (Pixel) bildet paarweise Cliquen mit allen anderen. Der hohe Verzweigungsgrad des *CRF* ist offensichtlich zu sehen. Bild von Krähenbühl et al.[6]

Im Themengebiet der Bildsegmentierung gilt es für eine Beobachtung (Farbvektoren des Eingabebildes) $I = \{I_1, ..., I_n\}$ eine optimale Belegung der Label $x \in L^n$ für $X$ zu finden, so dass $x = argmax_{x \in L^n} P(x|I)$ gilt. $P(x|I)$ ist dabei das Ergebnis einer Wahrscheinlichkeitsverteilung, wie zum Beispiel der *Gibbs Verteilung*.

[6] Präsentation von Krähenbühl et al.:`https://www.cs.cmu.edu/~efros/courses/LBMV12/crf_deconstruction.pdf`, Abrufdatum: 14.06.2018

Das Ergebnis der Gibbs Verteilung $E(x)$ für den vollständigen Graphen $G$ ergibt sich durch die Summe der Potentiale aller Cliquen $\phi_c$ - unäre $\phi_u$ und paarweise $\phi_p$ .

$$\begin{aligned} E(x|I) &= \sum_{c \in C_G} \phi_c(x_c|I) \\ &= \sum_i \phi_u(x_i|I) + \sum_{i<j} \phi_p(x_i, x_j|I) \end{aligned}$$

Gleichung 2.5: Gibbs Verteilung für eine Variablenbelegung $x$ gegeben einer Beobachtung $I$. $\phi_u$ beschreibt die unären Potentiale für die Belegung $x_i$ und $\phi_p$ die paarweisen Potentiale der Clique $(x_i, x_j)$.

Die unären Potentiale werden bei der Initialisierung des *CRFs* vom vorherigen Klassifizierer, in dieser Arbeit von einem *DCNN* übernommen. Die paarweisen Potentiale werden über die gewichtete Summe zweier gaußscher Kernel multipliziert mit einer *compatibility* Funktion $\mu$ (Gleichung 2.6), die eine Einschätzung über zwei benachbarte Pixel ausführt, berechnet (vgl. Krähenbühl et al. [KK11] Kapitel 2).

$$\begin{aligned} \phi_p(x_i, x_j|I) &= \mu(x_i, x_j) \cdot k(f_i, f_j) && (2.6) \\ &= \mu(x_i, x_j) \cdot \left( w^1 \cdot e^{-\frac{|p_i - p_j|^2}{2 \cdot \Theta_\alpha^2} - \frac{|I_i - I_j|^2}{2 \cdot \Theta_\beta^2}} + w^2 \cdot e^{-\frac{|p_i - p_j|^2}{2 \cdot \Theta_\gamma^2}} \right) && (2.7) \end{aligned}$$

Gleichung 2.6: Berechnung der Cliquenpotentiale der Belegungen an $i$ und $j$. $w^1$ und $w^2$ gewichten die zwei gaußscher Kernel. $p_i$ und $p_j$ bezeichnen die Positionen, $I_i$ und $I_j$ die Farbvektoren, $f_i$ und $f_j$ die Feature Vektoren der jeweiligen Indizes. $\Theta_\alpha$,$\Theta_\beta$,$\Theta_\gamma$ dienen der Regulierung.

Im weiteren Verlauf der Arbeit bilden die Variablen aus Gleichung 2.6 die Hyperparameter des *CRF*. Wichtig sind hierbei die zwei Gewichtungsparameter $w^1$ und $w^2$, die die Relevanz des farbbasierten Kernels ($w^1$) und des positionsbasierten Kernels ($w^2$) bestimmen.

### 2.2.1 Mean Field Approximation

Der oben beschriebene Berechnung der optimalen Belegung der Zufallsvariablen $X$ liegt die Bestimmung der maximalen Wahrscheinlichkeit der Belegung zu Grunde. Für eine exakte Bestimmung dieses Maximums müssten die Wahrscheinlichkeiten aller möglichen

Belegungen berechnet werden, was jedoch ein NP-Vollständiges Problem und somit nicht praktisch einsetzbar ist. Aus diesem Grund muss das Maximum, zum Beispiel mit der *Mean Field Approximation* Methode, angenähert werden.
Der Mean Field Approximation Algorithmus ist ein iteratives Vorgehen, bei dem in jedem Iterationsschritt eine neue Verteilung $Q(X)$ erzeugt wird, deren Belegung für $X$, möglichst nah an $P(X)$ liegen soll.

```
Initialize Q
while not converged:
  Q̃_i^(m)(x_j) = Σ_{j≠i} k^(m)(f_i, f_j) Q_j(x_j) for all m //Message Passing X_j → X_i
  Q̂_i(x_i) = Σ_{l∈L} μ^(m)(x_i, l) Σ_m w^(m) Q̃_i^(m)(l)      //compatability Funktion
  Q_i(x_i) = exp(−φ_u(x_i) − Q̂_i(x_i))                       //Update der Verteilung
  normalize Q_i(x_i)
```

Listing 2.2: Pseudocode des Mean-Field Approximation Algorithmus (nach Krähenbühl et al. [KK11]) in den vier Schritten einer Iteration. Die Bedeutung der Symbole und Funktionen ist oben (Kapitel 2.2) beschrieben.

## 2.3 Evaluierung

Zur einheitlichen Messung verschiedener Ansätze müssen Evaluierungsmaße definiert werden. Zunächst ist hierbei der Zeitverbrauch sowohl beim Trainieren als auch bei der Inferenz aufzuführen. Die Trainingszeit kann im Rahmen dieser Arbeit vernachlässigt werden, da ein Offline-Learning angestrebt ist.
Die Inferenzzeit kann als verarbeitete Bilder pro Sekunde /*Frames per Second (FPS)* oder auch die benötigte Zeit für ein Bild gemessen werden. Im Szenario der Arbeit ist die Angabe der *FPS* geeigneter, da es sich besser ins Verhältnis zu einer Fahrgeschwindigkeit setzen lässt.
Die nächste zu betrachtende Größe ist der Speicherbedarf des Ansatzes bei Training, Inferenz und des Modells an sich sowie dessen Komplexität. Da das Modell auf einer mobilen Plattform eingesetzt werden soll, wird ein kleineres Modell besser bewertet. Die Komplexität wird durch die Anzahl *Floating Point Operations (FLOPS)* gemessen. Auch hier gilt eine geringere Komplexität als besser zu bewerten.
Als qualitatives Maß hat sich für Segmentierung die *Mean Intersection over Union (MIOU)* durchgesetzt. Hierbei werden die true positives mit den false positives sowie den false negatives über alle Klassen ins Verhältnis gesetzt.

$$MIoU = \frac{1}{k+1} \sum_{i=0}^{k} \frac{p_{ii}}{\sum_{j=0}^{k} p_{ij} + \sum_{j=0}^{k} p_{ji} - pii} \tag{2.8}$$

Gleichung 2.7: Definition *MIOU*. $k+1$: Anzahl der Klassen; $p_{ii}$: True positives; $p_{ij}$ und $p_{ji}$ false positives/negatives

## 2.4 Rahmenbedingungen

In diesem Teil werden spezifische Rahmenbedingungen des Szenarios beschrieben, die im Laufe der Arbeit eine Rolle spielen.
Da für diese Arbeit kein spezifischer Roboter zur Verfügung steht, wird aus Modellen verschiedener Hersteller ein *typischer* Rasenmähroboter definiert. Hierbei sind die relevanten Eigenschaften die Höhe für die Perspektive der Eingabebilder sowie die Fahrgeschwindigkeit, um die Echtzeitfähigkeit der Lösung zu evaluieren. In Abbildung 2.6 sind drei Modelle des Marktführers *Husqvarna* sowie dessen Tochterunternehmen *Gardena* aufgeführt. Modelle anderer Hersteller ähneln sich bei den wichtigen Merkmalen Geschwindigkeit und Höhe. Nur wenige Roboter zeichnen sich durch eine höhere Geschwindigkeit oder kompaktere Maße aus. Aus den betrachteten Modellen ergibt sich eine typische Höhe von 25-30 cm und eine Geschwindigkeit von 1-2 km/h.
Durch Geschwindigkeit und Kameraperspektive lässt sich die Echtzeitfähigkeit für das Szenario definieren. Es muss darauf geachtet werden, dass der Roboter in keinen Bereich fährt, für den noch nicht ausreichend Informationen geliefert wurden. Da Ausreißerdaten mit fehlerhaften Segmentierungen nicht auszuschließen sind, sollten Informationen aus mindestens zwei Bildern vorliegen.
Bei einer angenommenen ausreichend guten Segmentierung bis ca. 10 Meter Entfernung und der geringen Fahrgeschwindigkeit deckt ein Bild ca. 20 Sekunden Fahrzeit ab. Aus diesen Überlegungen ergibt sich eine Minimum *FPS* von 0.1.
Da jedoch auch auf dynamische Hindernisse reagiert werden soll, sollte das Verfahren 1 *FPS* erreichen.

| **Modelle:** |  (a) HUSQVARNA AUTOMOWER® 105 |  (b) HUSQVARNA AUTOMOWER® 450X |  (c) Gardena Mähroboter R40Li |
|---|---|---|---|
| **Ausmaße (LxWxH):** | 55x39x25 cm | 72x56x31 cm | 58x46x26 cm |
| **Geschwindigkeit:** | ca 1 km/h | ca 2,5 km/h | ca 1 km/h |

Abbildung 2.6: Verschiedene Robotermodelle vom Marktführer Husqvarna mit Tochterunternehmen Gardena.

Das Aussehen der Bilder wird durch die Jahreszeit beeinflusst, in denen ein Rasenmähroboter arbeitet sowie typische Gärten des Szenarios.
Grob lässt sich die Arbeitszeit der Roboter auf den späten Frühling bis zum Herbst eingrenzen. Somit muss darauf geachtet werden, Bilder mit blühender Vegetation und unterschiedlich sonnigen Bedingungen aufzunehmen.
Da handelsübliche Rasenmähroboter bereits im höheren Preissegment liegen und eine erweiterte Sensorik zunächst auch höhere Preise zur Folge hat, müssen möglichst große Gärten im Datenset vorhanden sein. Es wird davon ausgegangen, dass die endgültige Anwendung nicht in kleinen Gärten, zum Beispiel in einer Reihenhaussiedlung, liegt, da eine derart große Investition erst bei einem großen Garten sinnvoll erscheint.
Da eine weltweite Datenaufnahme im Rahmen der Arbeit nicht möglich ist, beschränken sich die Bilder im Bezug auf Grasarten auf regionale Gärten in Norddeutschland.

## 2.5 System

Generell gilt bei Deep Learning Verfahren, dass das Training ein sehr kostspieliger Vorgang ist. Es ist sehr rechen- und speicherintensiv und wird deswegen in der Regel auf eine leistungsstarke GPU ausgelagert, die Matrizenoperationen effizienter durchführt als eine CPU. Die Inferenz mit einem fertigen Modell ist im Vergleich zum Training weitaus effizienter und kann auch auf schwächerer Hardware durchgeführt werden. Für diese Arbeit stehen zwei Rechner zur Verfügung. Auf dem Rechner mit stärkerer Hardware wird der Hauptteil der Arbeit durchgeführt, da für das Training viele Ressourcen benötigt werden. Die schwächere Hardware dient der Evaluierung, um einer potentiellen mobilen Plattform näher zu kommen.

### Trainingssystem

- CPU: AMD Ryzen™ 5 1600X
  - 6 Kerne
  - 3,6 GHz
- RAM: 16GB DDR4
- GPU: NVIDIA GeForce GTX 1080
  - 8GB GDDR5X VRAM
  - 2560 Cuda® Kerne
  - 1.657 MHz

### Evaluierungssystem

- CPU: Intel®Core™ i5-7300HQ
  - 4 Kerne
  - 2,5 GHz
- RAM: 8GB DDR4

### Software

Zur Verwendung von TensorFlow (Kapitel 2.1.3) werden einige NVIDIA-Tools benötigt. In dieser Arbeit wird TensorFlow in der Version 1.8.0 verwendet.

- *NVIDIA Cuda®*:
  Programmiermodell von NVIDIA zur parallelen Berechnung von Programmteilen auf der Grafikkarte. Cuda-Code lässt sich in C-/C++-Programme einbinden.
  Cuda wird in der Version 9.0 verwendet.
- *Cuda® Deep Neural Network library (cuDNN)*:
  Library mit stark optimierten Algorithmen für tiefe Neuronale Netze. Bietet für TensorFlow unter anderem grundlegende Funktionen, wie Backpropagation, Convolution, Normalisierungen oder Aktivierungsfunktionen.
  *cuDNN* wir in der Version 7.0.5 verwendet.

Die Vorverarbeitung der Bilder sowie die Evaluierung der Ergebnisse wird in Python 3 durchgeführt. Die wichtigsten Bibliotheken sind dabei:

- *Numpy*:
  *Numpy* ermöglicht die effiziente Verwendung von n-dimensionalen Arrays und numerischen Berechnungen. In dieser Arbeit wird *Numpy* vor allem für die Evaluierung der segmentierten Bilder verwendet sowie zur Berechnung von Tiefeninformationen.
  Genutzt wird die Version 1.14.5

- *OpenCV*:
  *OpenCV* bietet eine große Auswahl an Bildverarbeitungsalgorithmen und wird vor allem für die Vorverarbeitung des Datensets sowie die Kamerakalibrierung eingesetzt. Verwendet wird die Version 4.0.0.
- *Matplotlib*:
  Mit *Matplotlib* können einfach Graphen aus Daten erstellt werden. Die Graphen werden vor allem für die Evaluierung der Experimente benötigt. Verwendet wird die Version 2.2.2

# State of the art

## 3.1 Ansätze zur Bildsegmentierung

Der Einsatz von Deep Learning zum Segmentieren von Bildern wird erst seit einigen Jahren verwendet. Zuvor wurden bereits sehr gute Resultate im Bereich der Bildklassifizierung und Objektdetektion erzielt. Ein logischer Folgeschritt aus diesen Erfolgen ist die Segmentierung. Auf Grund dieser Entwicklung basieren die meisten Ansätze auch auf Klassifizierungs- oder Detektionsmethoden und erweitern diese.

### 3.1.1 Fully Convolutional Networks (FCN)

Das am öftesten verwendete *DCNN* ist das *VGG-16*-Netz von Simonyan et. al [SZ14]. Ihr Ansatz erreicht 2014 eine Top-Platzierung in der Pascal VOC 2012 [Eve+15] Klassifizierungschallenge. Der Aufbau des Netzes dient in dieser Arbeit bereits als Beispiel im Grundlagenkapitel (siehe Abbildung 2.3). Ansätze, die auf dem VGG-16 basieren verzichten auf die drei Fully Connected Layer und erhalten somit eine Heatmap anstatt eines Klassifizierungsergebnisses.
Long et al. [LSD15] entwickeln 2015 ein Verfahren, *Fully Convolutional Networks*, dass auf einem *DCNN* basiert und durch upsampling ein Segmentierungsergebnis bildet. Als Basis experimentieren sie mit VGG-16, AlexNet und GoogLeNet. In Abbildung 3.1 wird der Ansatz mit dem AlexNet dargestellt. *(a)* zeigt das ursprüngliche AlexNet, *(b)* die veränderte Form zum Erzeugen der Heatmap anstatt des Klassifizierungsergebnisses und *(c)* das anschließende Upsampling mit Segmentierungsergebnis.

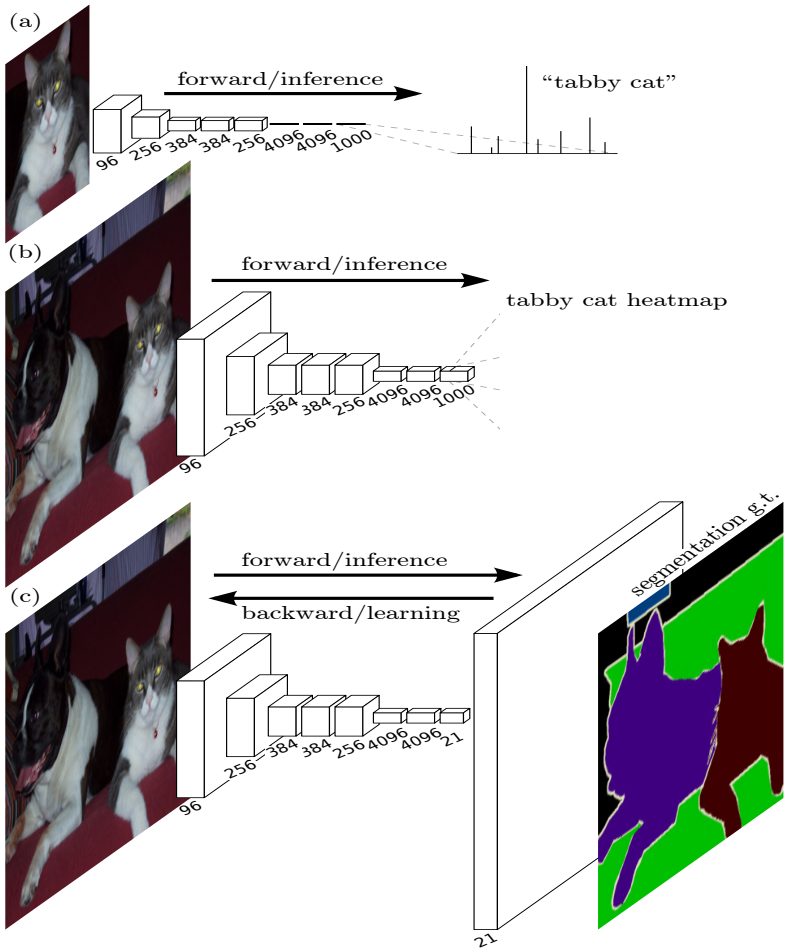
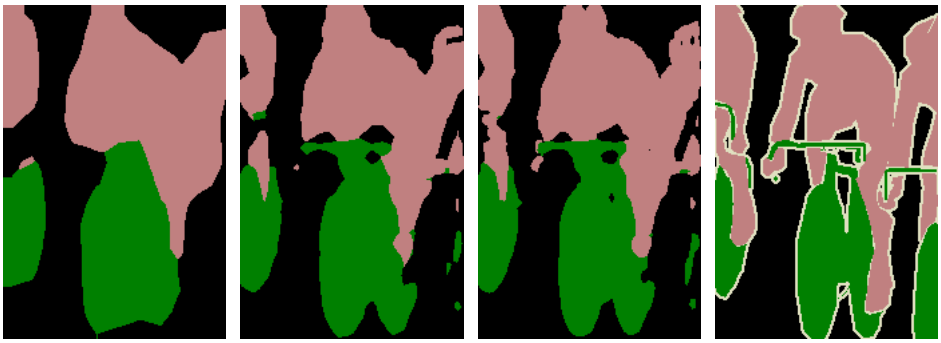



Abbildung 3.1: Long et al. Ansatz zur Segmentierung mit *CNN* und Upsampling. Evolution vom ursprünglichen AlexNet zum Segmentierungsansatz.

Beim Upsampling werden dabei drei verschiedene Methoden angewendet. Die drei Methoden unterscheiden sich darin, wie viele Informationen verwendet werden. Bei der *FCN-32s* Methode wird ausschließlich die Ausgabe des letzten Layers des *CNN* 32-fach hochskaliert. Bei *FCN-16s* und *FCN-8s* werden auch Zwischenergebnisse aus höheren Layern verwendet. So werden lokale Informationen mit größeren Regionen vereinigt und ein besseres Ergebnis erzielt. In Abbildung 3.2 werden die Segmentierungsergebnisse der verschiedenen Upsamplingmethoden gegenübergestellt. Dabei ist deutlich zu erkennen, dass beim *FCN-8s* viel genauere Konturen abgebildet werden als bei den anderen Methoden.
Aus der Arbeit von Long et al. können wir lernen, dass lokale mit globalen Informationen vermischt werden können, um bessere Ergebnisse zu erzielen.



(a) FCN-32s (b) FCN-16s (c) FCN-8s (d) Ground truth

Abbildung 3.2: Gegenüberstellung der verschiedenen Upsamplingmethoden von Long et al.

### 3.1.2 DeconvNet

Noh et al. [NHH15] stellen 2015 ein erweitertes Verfahren mit dem DeconvNet vor. Eine Schwäche am Verfahren von Long et al. haben Noh et al. bei der Segmentierung von großen oder sehr kleinen Objekten im Bild erkannt. Große Objekte werden oft auf viele Objekte aufgeteilt und kleine als Hintergrund erkannt. Den Grund für diese Schwäche sehen sie in der einfachen Upsampling-Methode.
Statt eines einfachen Upsamplings entscheiden sie sich für den Encoder-Decoder Aufbau. Der Decoder nimmt hierbei den Teil des Upsampling in Form von Gegenoperationen zum Encoder ein. Hierfür werden *deconvolutional* und *unpooling* Layer entwickelt, die die Operationen im Encoder spiegeln sollen.
In Abbildung 3.3 ist der Encoder als Convolution und der Decoder als Deconvolution gekennzeichnet. Der Decoder sorgt im Gegensatz zum FCN für ein qualitativ besseres Upsampling, das auch trainiert wird.

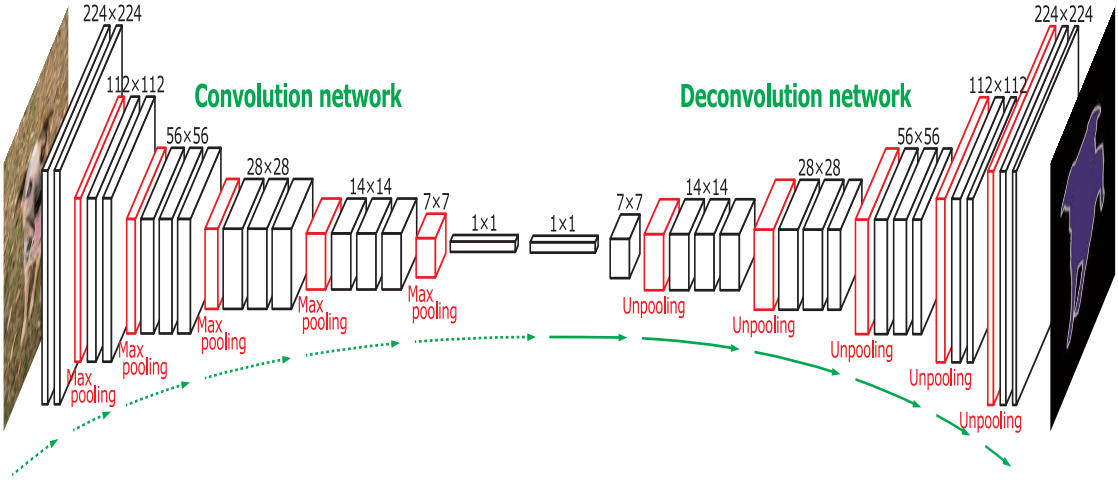
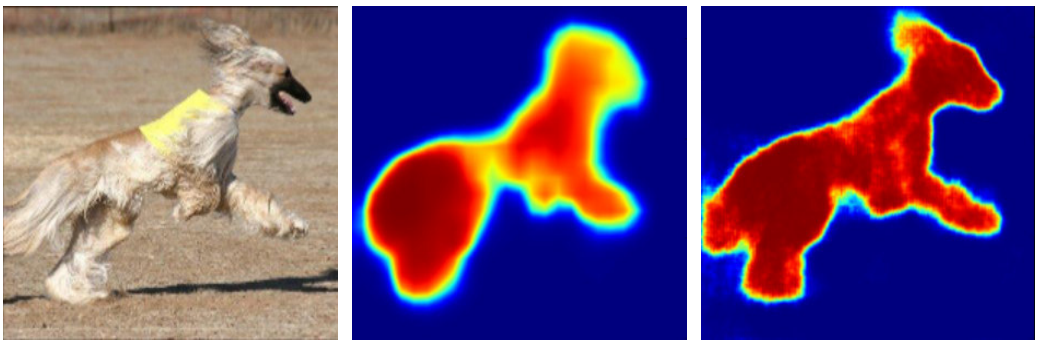



Abbildung 3.3: Aufbau des Ansatzes von Noh et al. [NHH15]

Noh et al. evaluieren ihren Ansatz mit dem Pascal VOC 2012 Datenset [Eve+15] und vergleichen ihre Ergebnisse primär mit dem Verfahren von Long et al.. Sie erzielen dabei im Durchschnitt mit $69,2\%$ ein um $7\%$ besseren $MIOU$. In Abbildung 3.4 ist deutlich zu erkennen, dass Noh et al. weitaus schärfere Konturen um das Objekt darstellen können, während Long et al. ein verschwommenes Segmentierungsergebnis liefern. Es gibt jedoch auch eine Vielzahl an Beispielen, in denen das FCN-8s bessere Ergebnisse liefert als DeconvNet, wodurch die Autoren auf komplementäre Eigenschaften beider Netze schließen.
Mit diesen Überlegungen versuchen Noh et al. ihr Ergebnis weiter zu verbessern. Sie erreichen ein um 3% besseres Ergebnis durch ein Ensemble von DeconvNet und FCN-8s, welches die Eigenschaften beider Netze verbindet.



(a) Eingabebild (b) FCN-8s (c) DeconvNet

Abbildung 3.4: Gegenüberstellung der Ergebnisse von Long et al. (FCN-8s) und Noh et al. (DeconvNet)

### 3.1.3 SegNet

Einen ähnlichen Ansatz fahren Badrinarayanan et al. [BKC15] 2016 mit dem SegNet. Auch sie nutzen einen Encoder-Decoder Aufbau mit einem VGG-16 als Encoder. Anders als Noh et al. verzichten sie jedoch auf die Fully Connected Layers am Ende des Encoders. Dadurch ist die Basis des Decoders eine höher aufgelöste Feature Map als zuvor. Nach dem letzten Layer des Decoders wird eine pixelweise Klassifizierung mit einem Softmax Layer durchgeführt.
Eine Besonderheit am SegNet ist die Reduzierung des Speicherbedarfs. Wie in Abbildung 3.5 durch die oberen vier Pfeile angedeutet, werden die Pooling Indices gespeichert und beim Upsampling wiederverwendet. Im Decoder werden ausschließlich die convolutional Layer gelernt, anstatt wie bei dem Vorgänger auch das Upsampling zu lernen. Durch dieses Verfahren wird der Speicherbedarf während der Inferenz im Vergleich zu den vorherigen Netzen auf nahezu die Hälfte verringert.
Evaluiert wurde SegNet mit dem CamVid Dataset[1], das aus Straßenbildern besteht. Verglichen wurden dabei eine verschiedene Anzahl an maximalen Iterationen. Die besten Ergebnisse lieferten dabei sowohl SegNet als auch DeconvNet mit nur ca. $0,3\%$ weniger. Besonders am SegNet ist dabei, dass bereits mit 40000 Iterationen ein nur wenig schlechteres Ergebnis erzielt wird. Die Performance vom DeconvNet wird bei sinkender Iterationenanzahl schnell schlechter.

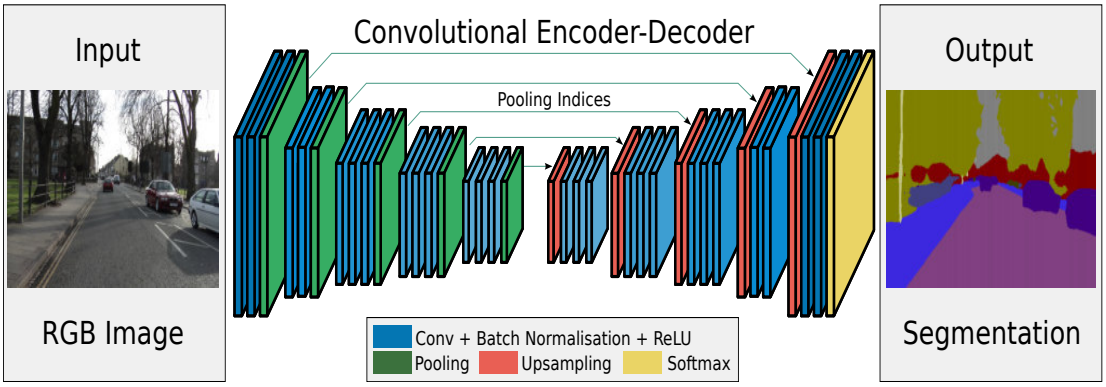



Abbildung 3.5: Aufbau des Ansatzes von Badrinarayanan et al. [BKC15]

### 3.1.4 *Recurrent Convolutional Neural Network (rCNN)*

Pinheiro et al. [PC14] verfolgen 2014 einen anderen Ansatz mit einem *rCNN*. Der Ansatz beruht darauf, ein *CNN* mehrfach hintereinander auszuführen, wobei die Ausgabe eines *CNN* jeweils die Eingabe des nächsten ist. Neben der vorherigen Ausgabe wird das

[1] CamVid Dataset Website: `http://mi.eng.cam.ac.uk/research/projects/VideoRec/`, Abrufdatum: 23.07.2018

Eingabebild auf die Größe der Ausgabe skaliert und ebenfalls in die nächste Iteration gegeben (siehe Abbildung 3.6). Die endgültige Segmentierung erreichen Pinheiro et al. durch einige Kombination aus Translation des Eingabebildes und Mergen der Ergebnisse.

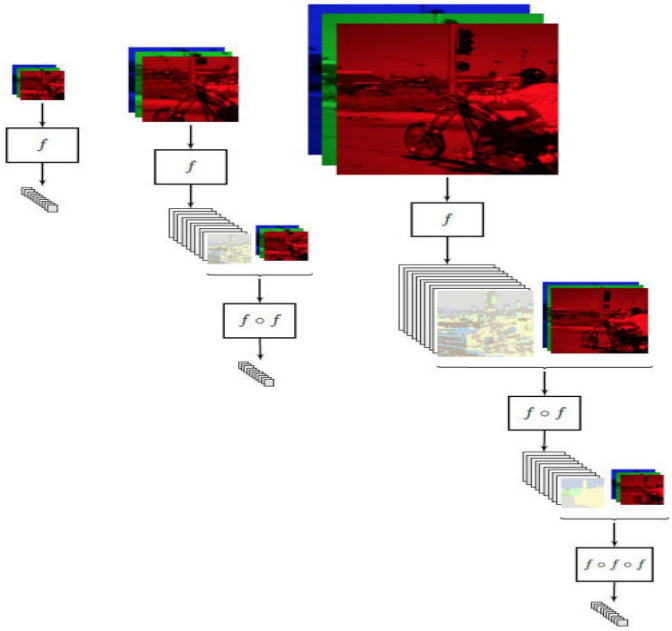



Abbildung 3.6: *rCNN* mit verschiedenen Iterationen nach Pinheiro et al.[PC14]. $f$ kennzeichnet den Einsatz eines *CNNs*

Die Autoren beobachten dabei, dass mit jeder Iteration das Netz sich selbst verbessern konnte und Fehler der vorherigen Iteration verringert wurden. Ebenfalls wird mit diesem Ansatz das Eingabebild in mehreren verschiedenen Skalierungen verwendet, was zu einer stärkeren Generalisierung des Models führt.

### 3.1.5 *Conditional Random Field (CRF)*

In der Literatur (z.B. [LW],[YK15] sowie im oben beschriebenen Ansatz von Noh et al. [NHH15]) wird gezeigt, dass mit einem *CRF* als Postprocessing nach dem neuronalen Netz, die Ergebnisse weiter verbessert werden konnten. Vor allem Konturen von Objekten konnten durch diese Methode genauer segmentiert werden.
In Kapitel 2.2 wird der theoretische Ansatz der Inferenz beschrieben. Krähenbühl et al. [KK11] erkennen den Message Passing Schritt als Flaschenhals der Mean Field Approximation. Dieser Schritt hat eine quadratische Komplexität im Bezug auf die Anzahl der Variablen (hier Pixel). Ihr Ansatz schafft es, diesen Schritt auf lineare Laufzeit zu reduzieren und somit die Komplexität des gesamten Algorithmus stark zu verringern.

In ihrem Paper vergleichen sie ihren Inferenzalgorithmus mit einem *Markov-Chain-Monte-Carlo (MCMC)*-Algorithmus, der in 36 Stunden für das Testbild konvergiert und erhalten innerhalb von $0,2$ Sekunden ein vergleichbares Ergebnis.

### 3.1.6 CRFasRNN

Zheng et al. [Zhe+15] entwickeln einen erwähnenswerten Ansatz, der ein *CRF* in den Lernprozess des neuronalen Netzes einbezieht. Aus dieser Verbindung entsteht ein zweiteiliges Netz aus FCN-8s und *CRF*, wobei das *CRF* sowohl die Ausgabe des FCN-8s, als auch das Ursprungsbild als Eingabe erhält.

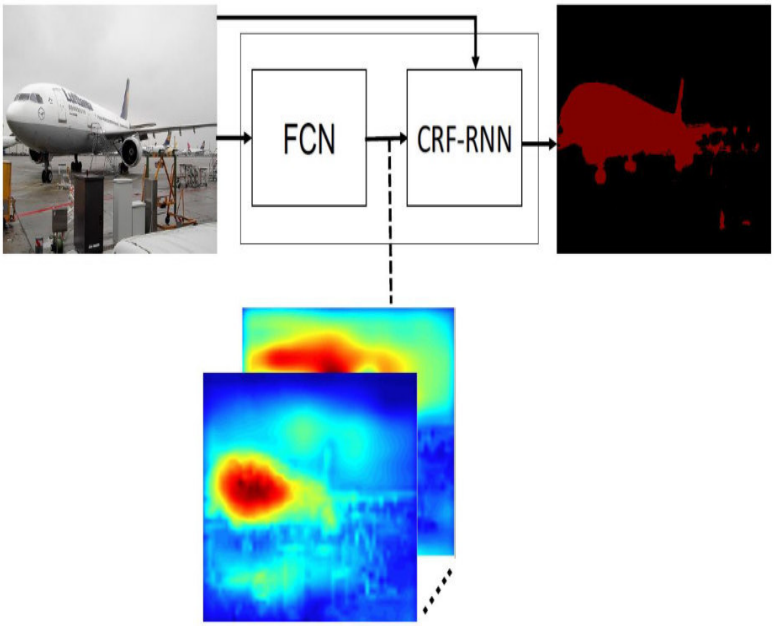



Abbildung 3.7: Aufbau von CRFasRNN von Zheng et al. [Zhe+15].

Für das *CRF as Recurrent Neutal Network (CRFasRNN)* setzen Zheng et al. den Ansatz von Krähenbühl et al. [KK11] in eine Folge von *CNN*-Layern um. Eine Iteration wird dabei durch fünf Layer abgebildet. Die Ausgabe dieser Layer wird in der nächsten Iteration wieder verwendet, wodurch die rekurrente Struktur zustande kommt. Initialisiert wird das *CRF* mit der Ausgabe eines vorgeschalteten FCN-8s. Da das *CRF* hierbei das FCN-8s direkt erweitert und durch die gleiche Struktur der beiden Teilnetze, wird ein end-to-end Training ermöglicht.
Ein direkter Vergleich zwischen diesem Aufbau und einem FCN-8s mit *CRF* als Postprocessing zeigt, dass ein end-to-end Training ein um 6% besseres Ergebnis liefert.

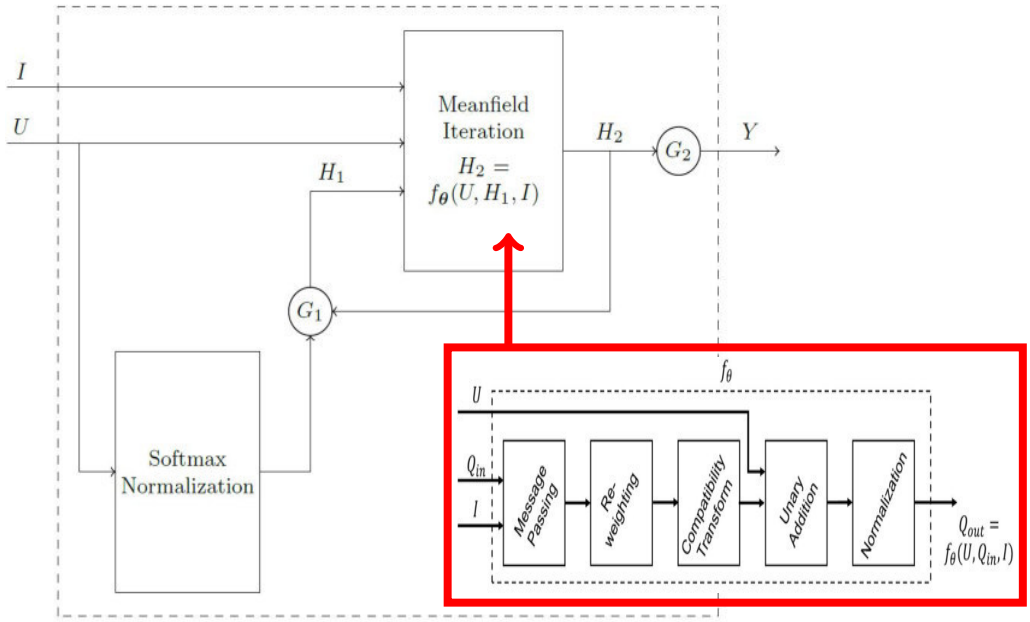



Abbildung 3.8: Umsetzung der Mean-Field-Approximation als *CNN*. **In rot**: Eine Iteration als Layer.

### 3.1.7 DeepLab

Chen et al. setzen mit DeepLab derzeit die obersten Maßstäbe im Pascal VOC 2012 Datenset. Anfang 2018 veröffentlichen sie die Version 3+ [Che+18] ihres Ansatzes, den sie seit drei Jahren konstant weiter verbessern und stehen damit zur Zeit (Stand: 07.08.2018) mit 89,0% *MIOU* an der Spitze des VOC 2012 Leaderboards.
Ihr ursprünglicher Ansatz [Che+14] setzt auf ein VGG-16 mit einem *CRF* als Post-Processing.

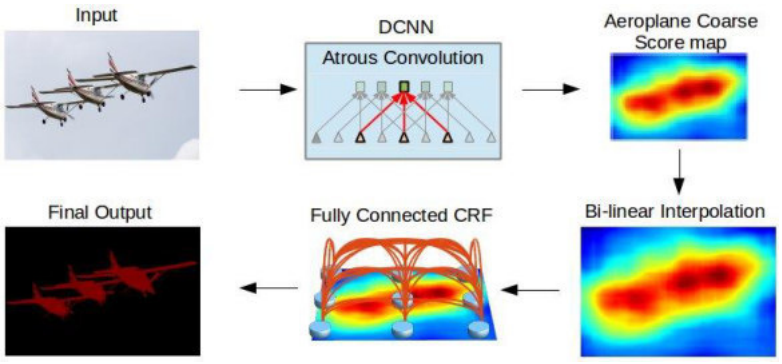



Abbildung 3.9: Aufbau der ersten zwei Versionen von DeepLab von Chen et al. [Che+14]

Ähnlich wie Badrinarayanan et al. [BKC15] sehen sie ein Problem in den niedrig aufgelösten Feature Maps. Um diesem entgegenzusetzen, nutzen sie Atrous Convolutional Layer, die ab einer gewissen Tiefe eine ebenso große Ausgabe wie Eingabe erzeugen. So muss die Ausgabe des VGG-16 nur wenig interpoliert werden, um sie wieder auf die Ursprungröße zurück zu führen. Diese hochskalierte Ausgabe wird dann durch das *CRF* mit dem Ansatz von Krähenbühl et al. [KK11] weiter pixelweise klassifiziert. In der zweiten Version [Che+16] ändern Chen et al. das Grundgerüst vom VGG-16 auf das ResNet-101 [He+15]. Außerdem ersetzen sie die Pooling Layer durch *Atrous Spacial Pyramid Pooling (ASPP)* Layer. Durch diese Layer betrachten sie innerhalb des Netzes parallel unterschiedliche Skalierungen der Umgebung zur Bestimmung eines Pixels. Insgesamt erreichen sie durch die Änderungen ein um ca. 8% besseres Ergebnis als in der ersten Version.
In der dritten Version [Che+17] führen sie einige Änderungen an der Atrous Convolution durch. Innerhalb des Netzes führen sie mehrere parallelisierte Atrous Convolutions mit unterschiedlichen Raten durch, um das Problem der unterschiedlich skalierten Objekte weiter zu verbessern. Ebenfalls passen sie die Atrous Rate in den tieferen Layern weiter an, um die Auflösung der Feature Maps hoch zu halten.
Ein großer Unterschied zu den vorherigen Versionen ist der Verzicht auf das *CRF*. Auch ohne das *CRF* erreichen sie in der dritten Version ein um ca. 6% besseren *MIOU*.
In der neuesten Version (*3+*) setzen auch Chen et al. auf eine Encoder-Decoder Struktur (Abb. 3.10). Der Encoder wird hierbei aus dem Xception-Net [Cho16] gebildet. Die Ausgabe des Encoders wird zunächst mit einem *ASPP* Layer verarbeitet und im Anschluss 4-fach hochskaliert.
Im Decoder wird ein frühes Zwischenergebnis des Xception-Nets verwendet und zunächst die Anzahl der Feature Maps verringert. Aus einer solch frühen Phase des Netzes wird ein low level Feature extrahiert, das im Encoder mit der Ausgabe des Netzes verbunden wird. Durch die zweifache Skalierung um einen geringeren Faktor gehen auch weniger Informationen verloren.
Sie erreichen nun mit 89% ein nochmal um ca. 4% besseres Ergebnis als zuvor.
Es fällt dabei auf, dass Chen et al. viele der Ideen aus den oben aufgeführten Papern vereinen. Sie nutzen Atrous Convolution, um die Auflösung der Feature Maps nicht zu gering werden zu lassen, *ASPP* Layer für die unterschiedlichen Skalierungen der Eingabe, zunächst ein *CRF* als Post-Processing und in der letzten Version auch die oft verwendete Encoder-Decoder Struktur.

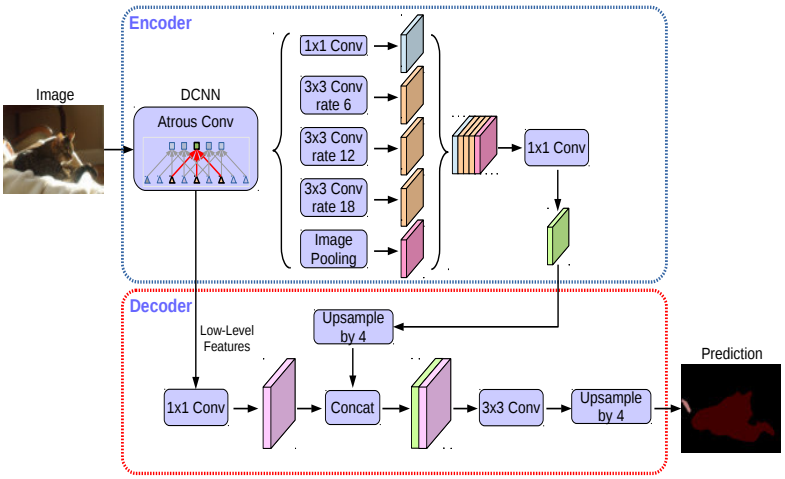
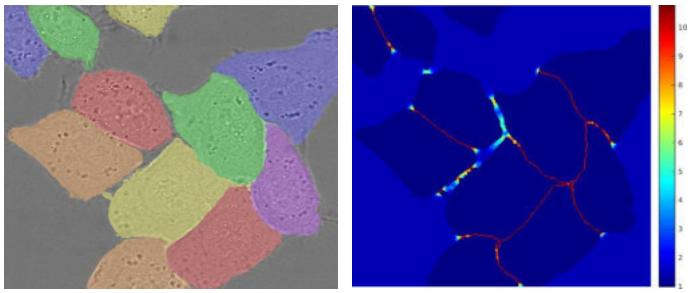



Abbildung 3.10: DeepLab Version 3+ mit Encoder-Decoder Struktur

## 3.2 Gewichteter Loss

Ronneberger et al. [RFB15] stellen in ihrer Arbeit zur Segmentierung einzelner biologischer Zellen eine Erweiterung des Trainingsprozesses vor, der theoretisch auf jeden anderen Ansatz übernommen werden kann, da es sich lediglich um eine Matrizenmultiplikation innerhalb der Lossberechnung handelt.



(a) Ground Truth im Eingabebild visualisiert.

(b) Gewichtung der Pixel

Abbildung 3.11: Gewichtung der Pixel nach Ronneberger et al. [RFB15]. Die Gewichte werden auf Basis des Ground Truths berechnet.

Um die sehr nah aneinanderliegenden Zellen voneinander zu trennen, legen die Autoren den Fokus auf die Zellränder sowie die schmalen Bereiche zwischen den Zellen. Hierfür berechnen sie im Vorfeld eine Gewichtung für jeden Pixel anhand seiner Distanz zur nächstgelegenen Zelle. In Abbildung 3.11 ist eine Gewichtung entsprechend des Ground

Truth abgebildet. Es ist zu beobachten, dass die Zellen selbst eine Gewichtung von 1 haben. Je nachdem wie nah die Zellen aneinanderliegen, wird die Grenze zwischen ihnen stärker gewichtet. Die großen Bereiche außerhalb der Zellen bekomen eine ähnlich niedrige Bewertung wie die Zellen selbst.
Die so erstellte Gewichtsmap wird während der Lossberechnung verwendet, um Fehler in den Randbereichen stärker in den Loss einzubringen. Somit wird im Trainingsprozess das Ausbessern dieser Fehler stärker belohnt.

## 3.3 RGB-D Ansätze

In dieser Arbeit soll auch mit Tiefeninformationen experimentiert werden. Im Folgenden werden einige Arbeiten vorgestellt, die RGB-D Daten verwenden. Die Eingabedaten stammen dabei meist aus Indooraufnahmen mit einem Tiefensensor, um Tiefeninformationen zu jedem Pixel zu erhalten. Hier unterscheidet sich die Prämisse stark zu den Bildern dieser Arbeit, bei denen es sich um Outdooraufnahmen handelt und die Tiefeninformationen nur geschätzt werden können.
Wang et al. [Wan+15] beschreiben drei Möglichkeiten (Abbildung 3.12) RGB-D Daten zu verarbeiten.

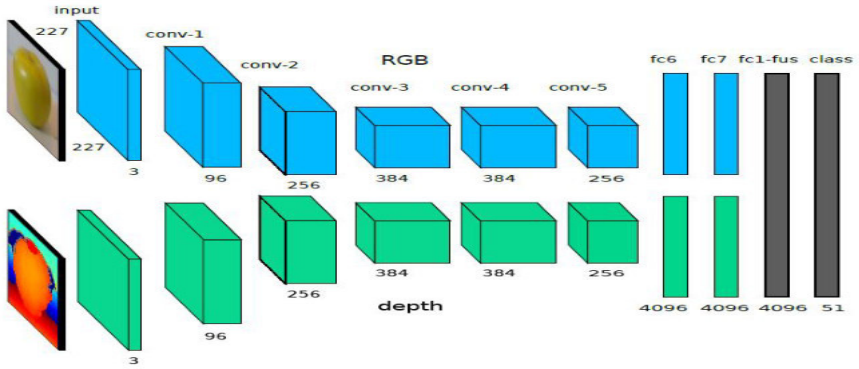
(a) Fusion von RGB mit D vor dem *CNN*.

(b) Zwei separate *CNN* die in einem Layer fusioniert werden.

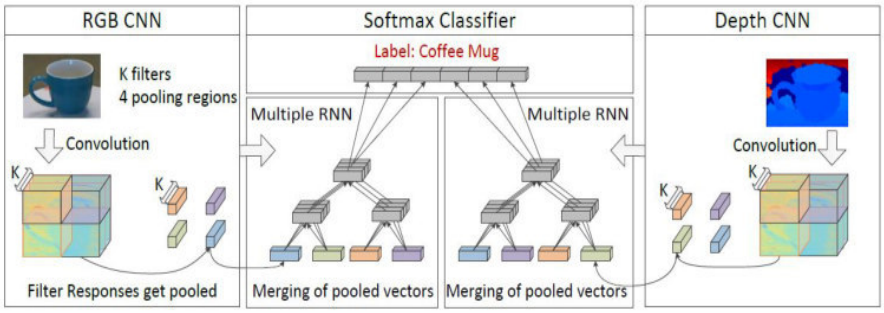
(c) RGB und D komplett separat

Abbildung 3.12: Mögliche Modellarten zur Verarbeitung von RGB-D Daten nach Wang et al. [Wan+15]. Die farblichen Rechtecke kennzeichnen die Layerarten - Grün: Convolutional; blau: Pooling; gelb: Fully Connected; rot: Softmax

Eitel et al. [Eit+15] und Socher et al. [Soc+12] entwickeln jeweils Modelle, die der Variante 3.12 *(b)* zugeordnet werden können. Eitel et al. folgen der Struktur ziemlich genau. Es werden Features aus RGB sowie D trainiert und die Vorzüge beider verbunden. Sie erreichen mit ihrem Ansatz eine accuracy von $84,7\%$ im RGB-D Object Dataset [LBF14].
Socher et al. fokussieren in ihrer Arbeit die RNNs. Sie erreichen mit $86,8\%$ Genauigkeit ein noch besseres Ergebnis als Eitel et al..



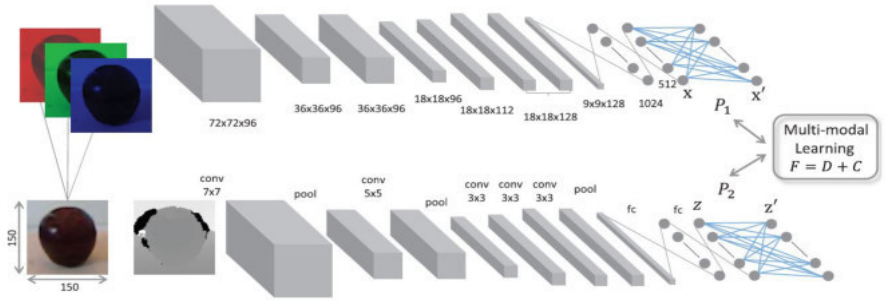

(a) Modelaufbau von Eitel et al. [Eit+15]. Die zwei *CNN* Teile werden in einem Fully Connected Layer fusioniert.



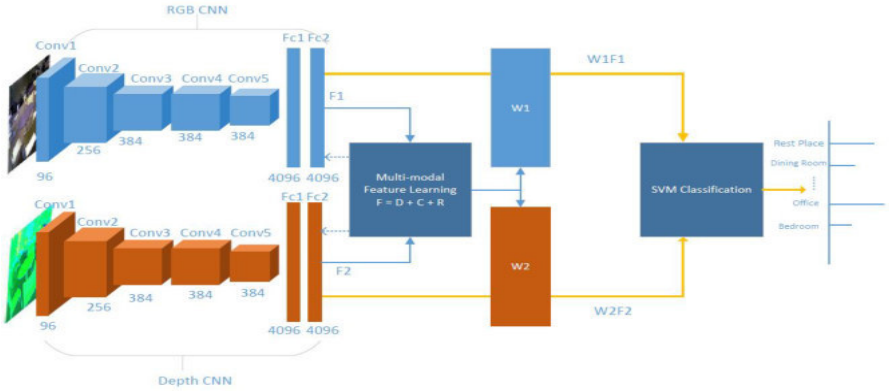

(b) Modelaufbau von Socher et al. [Soc+12]. RGB sowie D werden mit einem *CNN* und mehreren RNNs verarbeitet und konkateniert in einem softmax Layer klassifiziert.

Abbildung 3.13: RGB-D Modelle der Variante 3.12 *(b)*

Die Modelle von Zhu et al. [ZWL16] und Wang et al. [Wan+15] lassen sich in die Variante 3.12 *(c)* einordnen. Wang et al. nutzen zwei *CNNs*, deren Ausgabe in einem *Multi-Modal Layer* verknüpft sind. Im Vergleich zur Fusion über Fully Connected Layer werden Beziehungen der verschiedenen Quellen gelernt. Sie erreichen mit $86,9\%$ Genauigkeit ein noch leicht besseres Ergebnis als Socher et al.
Zhu et al. erweitern genau diesen Aufbau noch um eine SVM zur Klassifizierung. Evaluiert wird der Ansatz auf dem SUN-RGBD Datenset [SLX15] und erreicht mit $41,5\%$ ein um $15\%$ besseres Ergebnis als Wang et al. auf diesem Datenset




(a) Modelaufbau von Wang et al. [Wan+15]. Für RGB und D werden separate Netze verwendet. Die Ausgaben der *CNNs* werden in einem Multi-modal Layer verbunden.




(b) Modelaufbau von Zhu et al. [ZWL16]. Die Ergebnisse der *CNNs* werden wie bei Wang et al. in ein Multi-modal Layer gegeben. Weiterhin werden die Ausgaben mit zwei Projektionsmatrizen weiterverarbeitet und mit einer SVM klassifiziert. Die gestrichelten blauen Linien kennzeichnen die Backpropagation

Abbildung 3.14: RGB-D Modelle der Variante 3.12 *(c)*

Keine der drei Varianten von Wang et al. eignet sich jedoch für diese Arbeit. Um Variante *(a)* umzusetzen, müsste die Eingabe des verwendeten Ansatzes von dreidimensional auf vierdimensional verändert werden. Da der Ansatz mit einem vortrainierten Model initialisiert wird, dessen Eingabe nicht verändert werden kann, ist dies jedoch nicht akzeptabel. Die an mehreren Datensets trainierten Gewichte müssten verworfen und die Layer zufällig initialisiert werden. Dieser Qualitätsverlust wäre nicht mit dem zu erwarteten Gewinn von geschätzten Distanzen zu vereinbaren. Zufällig initialisierte Layer können ebenfalls nicht mit allen Datensets, die beim ausgesuchten Model verwendet wurden, trainiert werden, da nicht alle Sets frei verfügbar sind und auch keine Distanzinformationen für diese vorhanden sind.
Varianten *(b)* und *(c)* scheitern an den ausschließlich geschätzten Distanzen im Datenset (siehe Kapitel 4.4). Diese sind für jedes Bild identisch und lassen somit kein Training rein auf Distanzinformationen zu und die Hälfte des Netzes wäre ohne Informationsgewinn.

In dieser Arbeit wird eine Variante entwickelt, die von den drei vorgestellten Varianten inspiriert ist, jedoch das Problem des geringen Informationsgehalt der Tiefeninformationen berücksichtigt.

# Erstellung des Datensets

In diesem Kapitel wird die Erstellung des Datensets beschrieben. Dies beinhaltet die Beschreibung der Aufnahmeplattform, die Aufnahme der Bilder, die Segmentierung und die Erstellung der Tiefeninformationen.

## 4.1 Plattform

Zur Erstellung des Datensets wird zunächst ein Setup definiert, dass dem Szenario entspricht. Für die Bildaufnahme wird eine GoPro HERO3+ Black Edition Kamera mit 12MP Weitwinkelobjektiv ausgewählt. Die gewählte Kamera bietet die Möglichkeit des 2 Bilder pro Sekunde Modus, mit dem das Datenset erstellt wird.
Durch die hohe Auflösung von 3000x4000 Pixeln bei 72 dpi kann im Lernverfahren ein vielfältiges Prepocessing durchgeführt und aus der Segmentierung eine gute Positionsgenauigkeit berechnet werden.
Als Trägerplattform wird ein Kinderdreirad verwendet, bei dem die Kamera auf dem vorderen Schutzblech angebracht wird (Abbildung 4.1). Das Schutzblech befindet sich in einer Höhe von 25cm über dem Boden. Zusammen mit der Halterung der GoPro ergibt sich eine Höhe von 30cm für die Kamera. Durch das Dreirad wird die gewünschte Perspektive (Kapitel 2.4) erreicht und die Gärten können dank einer Schiebestange gezielt abgefahren werden. Die Kamera wird genau so ausgerichtet, dass der untere Bildbereich knapp über dem Reifen ist und dieser nicht mehr im Bild zu sehen ist. So kann der Bereich möglichst nah an der Trägerplattform aufgenommen werden.


Abbildung 4.1: Aufnahme der Trägerplattform zur Erstellung des Datensets, nach Anforderungen aus Kapitel 2.4.

## 4.2 Bildaufnahme

Insgesamt wurden 8435 Bilder in zwei öffentlichen Grünanlagen und 16 privaten Gärten ab einer Größe von 300m$^2$ aufgenommen.
Die Aufnahmen wurden im Herbst gemacht. Durch eine gezielte Auswahl sonniger und bewölkter Aufnahmetage wurde eine gute Diversität der Daten gewährleistet. In Abbildung 4.2 ist zu sehen, dass die Bilder den geforderten Ansprüchen gerecht werden. Die Schattenverhältnisse sind sehr unterschiedlich und typische Rasenbegrenzungen (Zäune, Steine, Beete usw.) sind vorhanden. Durch die gewählte Jahreszeit ist in einigen Gärten ein erhöhtes Laubaufkommen zu beobachten. Auch so wird für ein realistisches repräsentatives Datenset gesorgt.
Die Aufnahmen werden in sehr langsamen Gehtempo durchgeführt. Aufgrund fehlender Sensorik zur Geschwindigkeitsmessung kann hier jedoch keine genaue Angabe gemacht werden.

(a) Parkanlage (b) Holzzaun und Laub (c) Weg und Beete

(d) Hecke und Schatten (e) Sonne und Schatten (f) Teichanlage

Abbildung 4.2: Beispiele aus dem Datenset. Aufgenommen aus der Perspektive eines Rasenmähroboters mit der GoPro

Die Aufteilung des Datensets erfolgt durch eine zufällige Auswahl in Trainings-, Validierungs- und Testset. Dabei werden je 80% für das Trainingsset verwendet. Aufgeteilt werden zunächst lediglich Bilder aus 16 der 18 Aufnahmereihen. Die Verbleibenden werden als separates Testset für den ausgewählten Ansatz verwendet. Zwar ist kein identisches Bild in mehreren Sets vorhanden, da jedoch für jedes Bild ein zweites Bild nur eine halbe

Sekunde später existiert, kann ein sehr ähnliches Bild in einem anderen Set enthalten sein. Aus diesem Grund gibt es zwei komplett *unbekannte* Gärten als reales Testbeispiel.

## 4.3 Segmentierung

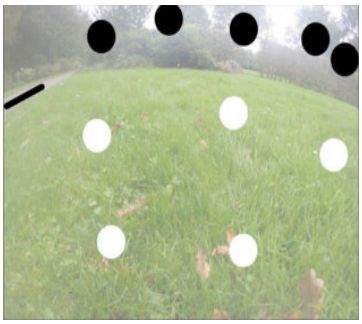
Zur Erstellung der Ground Truth Daten müssen die aufgenommenen Bilder per Hand binär segmentiert werden. Eine Schwierigkeit ergibt sich in der genauen Definition von *mähbaren Flächen*, bei einer Rasenfläche mit vielen erdigen Stellen oder laubübersähten Bereichen, da kleine solcher Bereiche oder einzelne Blätter als mähbar erkannt werden sollen, sehr große Bereiche jedoch nicht - eine feste Grenze gibt es dabei nicht. Bei einer von Laub bedeckten Fläche wird geprüft, ob innerhalb der Fläche Laub oder Rasen überwiegt und nach diesem Kriterium entschieden, selbst wenn sich unter dem Laub Rasen befindet. Bei erdigen Stellen wird die Einschätzung nach menschlichem Ermessen getätigt und individuell bestimmt, ob die Fläche mähbar ist oder nicht. Generell gilt jedoch, dass Flächen, die an Bäumen oder der Rasenkante angrenzen als nicht-mähbar gelten.
Für die Segmentierung des Datensets wurde ein semi-automatisches Segmentierungstool entwickelt. Das Tool basiert auf dem GrabCut Algorithmus [RKB04] sowie dessen Implementierung in OpenCV[1].
Ziel vom GrabCut ist es, auf Basis weniger Informationen des Anwenders eine Segmentierung des gesamten Bildes durchzuführen. In der OpenCV Implementierung wird hierfür zunächst eine rechteckige *Region of Interest (RoI)* ausgewählt. Auf diesen Schritt wird in dem hier beschriebenen Tool verzichtet, da kein Objekt den Vordergrund darstellt.
GrabCut ist ein iteratives Verfahren, dass auf dem GraphCut Algorithmus basiert. Zunächst werden zwei *Gaussian Micture Model (GMM)* (eines für Vordergrund; eines für Hintergrund) anhand der vom Benutzer klassifizierten Pixel initialisiert. Es wird ein Graph aufgebaut, bei dem jedes Pixel mit seinen acht Nachbarn sowie jeweils einem Knoten für Vorder- und Hintergrund verbunden ist.
Die Segmentierung erfolgt durch einen GraphCut in diesem Graphen, der eine Energy Funktion minimiert.

$$min_{\alpha,k}\left(E(\alpha, k, \Theta, z)\right) \tag{4.1}$$

Die Energy Funktion ist dabei abhängig von den *GMMs* $k$, den Pixelwerten $z$, einer Verteilung der Grauwerte $\Theta$ sowie der Belegung jedes Pixel als Vorder- bzw. Hintergrund $\alpha$.
Im entwickelten Segmentierungstool werden die *GMMs* der finalen Segmentierung ex-

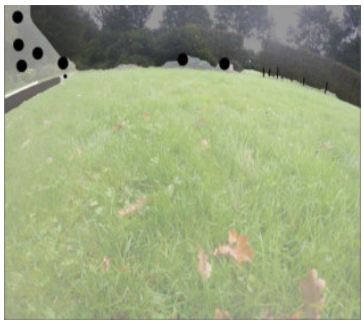
[1] OpenCV GrabCut Tutorial: `https://docs.opencv.org/3.3.0/d8/d83/tutorial_py_grabcut.html`, Abrufdatum: 08.08.2018

trahiert und in das nächste Bild übernommen. So müssen im Algorithmus keine *GMMs* initialisiert werden. Dies spart zum einen Rechenzeit sowie auch eine initiale Benutzerinteraktion.



(a) Initiale Informationen für Vorder- und Hintergrund

(b) Erste Segmentierungsergebnisse



(c) Korrektur der fehlerhaften Bereiche

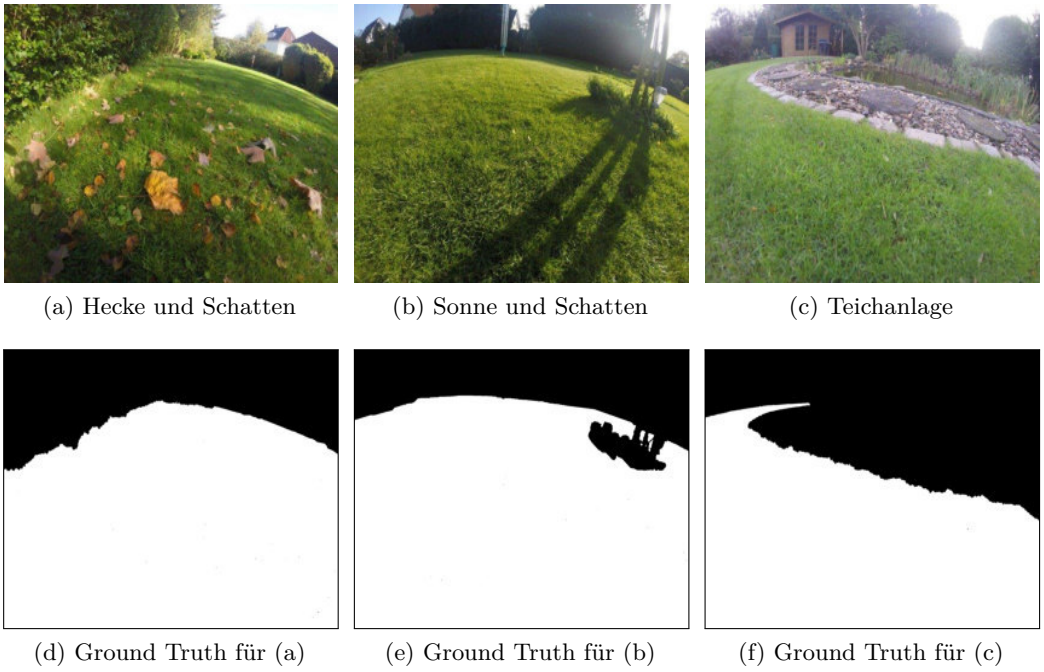
(d) Finale Segmentierung

(e) Initiale Segmentierung vom nächsten Bild mit vorherigem Model

Abbildung 4.3: Einzelschritte der semi-automatischen Segmentierung. *(a)* bis *(d)* behandelt das erste Bild einer Serie. *(e)* zeigt die Segmentierung des Folgebildes mit dem Model aus dem ersten Bild

Segmentiert werden einzelne Aufnahmereihen. Die Einzelschritte sind in Abbildung 4.3 zu sehen. Hierbei wird zu Beginn die mähbare Fläche als Vordergrund (weiße Kreise), sowie die nicht-mähbare als Hintegrund (schwarze Kreise), per Hand grob angeklickt. Durch die gewählten Bereiche wird der GrabCut Algorithmus initialisiert und ausgeführt. Dies lässt sich wiederholen, bis das Ergebnis zufriedenstellend ist. Da das gelernte Model für das nächste Bild übernommen wird und zwischen zwei aufeinanderfolgenden Bildern lediglich eine halbe Sekunde liegt, kann direkt eine gute Segmentierung erstellt werden (Abbildung 4.3 (e)). So wird der zeitaufwendige Schritt der Initialisierung eingespart.
Durch dieses Verfahren lässt sich ein guter Ground Truth in ca. 20 Sekunden pro Bild erstellen. Als einen möglichen Ansatz, der die Problemstellung der Arbeit löst eignet sich das Verfahren jedoch nicht. So sind die trainierten *GMMs* erzielen ausschließlich

für die Aufnahmereihe, in der sie trainiert wurden gute Ergebnisse. Da selbst eine neue Aufnahmereihe im gleichen Garten zu einer schlechten Segmentierung führt, können die Anforderungen, die in der Arbeit gestellt werden, nicht erfüllt werden.



(a) Hecke und Schatten (b) Sonne und Schatten (c) Teichanlage

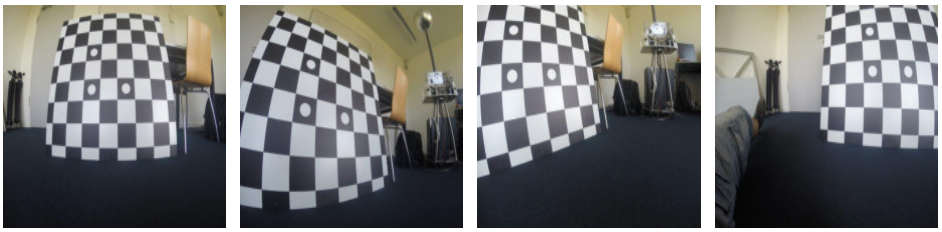
(d) Ground Truth für (a) (e) Ground Truth für (b) (f) Ground Truth für (c)

Abbildung 4.4: Ground Truth zu einigen Beispielen des Datensets.

## 4.4 Kalibrierung

Zur Erweiterung des Datensets sollen Tiefeninformationen erzeugt werden. Hierfür ist eine Kamerakalibrierung zur Bestimmung der intrinsischen und extrinsischen Parameter notwendig, die sich durch eine Schachbrettkalibrierung ermitteln lassen. Diese wurde mithilfe eines Kalibrierungstools eines Kommilitonen durchgeführt. Das Tool basiert auf der OpenCV C++-Implementierung[2] und erweitert dieses um einige Komfortfunktionen wie das automatische Auswählen geeigneter Bilder anhand der *findChessboardCorners* Methode.

[2]OpenCV Kalibrierungstutorial: `https://docs.opencv.org/3.1.0/d4/d94/tutorial_camera_calibration.html`, Abrufdatum: 08.08.2018



(a) (b) (c) (d)

Abbildung 4.5: Beispiele der Kalibrierungsbilder.

Die intrinsische Kameramatrix (Gleichung 4.1) wurde mit der oben beschriebenen Kalibrierung ermittelt. In der Kameramatrix wird die *Focal Length* sowie der *Principal Point* angegeben.
Da die Bilder der GoPro stark verzerrt sind, wie in Abbildung 4.5 zu sehen, gehört zur vollständigen Kalibrierung zusätzlich die Distortion Parameter (Abbildung 4.2). Die angegebenen Parameter entsprechen der Notation des OpenCV Tutorials[3].

$$M = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1750.908347588039 & 0 & 2000 \\ 0 & 1748.251212566175 & 1500 \\ 0 & 0 & 1 \end{pmatrix}$$

Gleichung 4.1: Intrinsische Kameramatrix

$$\begin{aligned} k_1 &= 3.7011 & k_2 &= 1.817 \\ p_1 &= -1.2214 \cdot 10^{-4} & p_2 &= -5.9821 \cdot 10^{-5} \\ k_3 &= 0.0894 & k_4 &= 3.986 \\ k_5 &= 2.7668 & k_6 &= 0.3672 \\ s_1 &= 0.0033 & s_2 &= -7.0122 \cdot 10^{-4} \\ s_3 &= -9.6709 \cdot 10^{-4} & s_4 &= 3.2171 \cdot 10^{-4} \\ \tau_x &= -1.9316 \cdot 10^{-3} & \tau_y &= 9.6892 \cdot 10^{-3} \end{aligned}$$

Gleichung 4.2: Distortion Parameter

Mithilfe der Parameter lässt sich die perspektivische Verzerrung der Bilder aufheben. In Abbildung 4.6 ist die Korrektur der Verzerrung in zwei Schritten aufgezeigt. Zunächst werden die intrinsischen und distortion Parameter verwendet. In Abbildung 4.6 *b)* ist die Verzerrung bereits herausgerechnet und die im Ursprungsbild noch gebogene Hecke ist gerade. Da jedoch nicht der gesamte korrigierte Bereich aufgenommen wurde, entstehen

[3]OpenCV Kameraparameter: `https://docs.opencv.org/3.3.1/d9/d0c/group__calib3d.html`, Abrufdatum: 08.08.2018

die gebogenen Ränder und die schwarzen Füllflächen. Für eine sinnvolle Weiterverwendung des Bildes wird im zweiten Schritt ein maximal großes Rechteck gesucht, dass keine schwarzen Bereiche einfasst.
Das so erstellte Bild zeigt den zentralen Bereich des Bildes ohne Verzerrung. War es zunächst noch eine Überlegung das komplette Datenset zu entzerren, um auf perspektivisch korrekten Bildern zu arbeiten, wird dieses jedoch anhand der Ergebnisse verworfen. Durch die verhältnismäßig sehr großen schwarzen Ränder muss das Rechteck kleiner gehalten werden und die Ränder des Bildes gehen verloren. So wird der betrachtete Gesamtbereich durch diesen Schritt zu klein um eine Einschätzung des Arbeitsraums vorzunehmen.
Ein weiterer Punkt, der gegen die Korrektur der Perspektive spricht, ist die fehlende Notwendigkeit. Da keine klassischen Objekte, sondern eine Rasenfläche segmentiert wird, kann keine Verzerrung das Objekt verändern. Ebenfalls weisen sowohl Trainings- als auch Testdaten die identische Verzerrung auf.
Stattdessen wird die Kalibrierung ausschließlich für die Berechnung der Distanz zu jedem Pixel und der Erstellung einer Tiefenkarte verwendet.

(a) Ursprungsbild mit Verzerrung

(b) Entzerrung des Bildes anhand der Parameter

(c) Größtmögliches Rechteck ohne schwarze Flächen

Abbildung 4.6: Korrektur der Verzerrung in zwei Schritten.

Für die Erstellung einer Tiefenkarte ist eine Berechnung der Realdistanz für jedes Pixel nötig. Hierfür muss die Projektion eines Punktes in der Welt auf die Bildfläche (Gleichung 4.3) invertiert werden.
Für die Invertierung ist die schrittweise Projektion in Gleichung 4.3 *unten* leichter zu betrachten. Zunächst wird durch die Gleichung (4.2) die Weltkoordinate in das Referenzsystem der Kamera transformiert. Im zweiten Schritt (4.3) wird der dreidimensionale Punkt auf die zweidimensionale Bildebene projiziert, indem durch den $z$-Wert (Achse in Kamerablickrichtung) geteilt wird. Im letzten Schritt wird dann für den Punkt auf der Bildebene anhand der intrinsischen Parameter die Pixelkoordinaten berechnet.

$$M \cdot [R|t] \cdot \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} u \\ v \\ 1 \end{pmatrix}$$

$$R \cdot \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} + t = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \tag{4.2}$$

$$\frac{1}{z} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} \tag{4.3}$$

$$M \cdot \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \tag{4.4}$$

Gleichung 4.3: Projektion von Weltkoordinaten $(X\ Y\ Z)^T$ in Pixel $(u\ v\ 1)^T$ mithilfe der intrinsischen $M$ und extrinsischen $[R|t]$ Kameraparameter. **Unten:** Projektion in mehreren Schritten.

Diese schrittweise Berechnung lässt sich nun leicht invertieren. Zunächst muss einmalig die intrinsische Kameramatrix invertiert werden, um die Pixelkoordinaten auf die Bildebene zu projizieren.

$$M^{-1} \cdot \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} \tag{4.5}$$

Für den nächsten Schritt muss nun die Annahme der flachen Ebene getroffen werden. Der in (4.5) berechnete Punkt stellt einen Punkt auf einer Geraden (4.6) ausgehend vom Kameraursprung dar. Es kann jedoch nicht berechnet werden, wo der Punkt auf dieser Geraden wirklich liegt, da die $z$ Information fehlt. Aus diesem Grund wird von einer komplett flachen Grundfläche ausgegangen, um den Schnittpunkt der Geraden mit der Bodenebene (4.7) zu berechnen.

(4.6) beschreibt die Gerade anhand des Mittelpunktes der Kamera in Weltkoordinaten $t$, der Rotation der Kamera $R$ und dem Punkt auf der Bildebene $(x'\ y'\ 1)^T$. (4.7) zeigt die Ebenengleichung der Bodenebene mit Annahme der ebenen Bodenfläche. Durch Gleichsetzen der Gleichungen wird der Schnittpunkt und somit die Weltkoordinaten des Punktes berechnet.

$$t + R \cdot z \cdot \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \tag{4.6}$$

$$\left( \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} - \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \right) \cdot \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} = 0 \tag{4.7}$$

Gleichung 4.4: Hilfsgleichungen zur Berechnung der Koordinaten des Kamerasystems $(x'\ y'\ 1)^T$ in Weltkoordinaten $(X\ Y\ Z)^T$.

Mit diesen theoretischen Überlegungen wird nun die extrinsische Kalibrierung durchgeführt. Eine standardisierte Kamerakalibrierung führt üblicherweise sowohl eine intrinsische als auch eine extrinsische Kalibrierung durch. Dies setzt jedoch voraus, dass die Kalibrierungsaufnahmen aus der realen Perspektive der Kamera aufgenommen werden. Weitere Anforderungen an eine gute Kalibrierung sind, dass das Schachbrett zu einem Großteil im Bild und über mehrere Bilder in allen Teilen des Bildes zu sehen sein muss. Aufgrund der sehr niedrigen Perspektive über dem Boden im Szenario dieser Arbeit sind all diese Kriterien nicht zu erfüllen und die Kalibrierung wird nicht aus der realen Perspektive aufgenommen.
Stattdessen wird die extrinsische Matrix per Hand erstellt. Zunächst wird die Position der Kamera (4.8) durch messen der Höhe über dem Boden bestimmt. Da der Ursprung des Weltkoordinatensystems auf den Punkt am Boden direkt unter der Kamera gesetzt wird bleiben $x$ und $z$ des Kameraursprungs 0.

$$t = \begin{pmatrix} 0 \\ .30 \\ 0 \end{pmatrix} \tag{4.8}$$

Für die Rotationsmatrix $R$ der Kamera stehen bereits zwei Winkel fest. Die Kamera ist nicht nach rechts oder links geneigt und zeigt gerade nach vorne. Somit gibt es keine Rotation um $Z$ oder $X$. Der Neigungswinkel nach vorne wird experimentell bestimmt. Hierfür wird ein Schachbrett in einer bestimmten Entfernung vor die Trägerplattform gelegt und anhand der Berechnungen aus Gleichung 4.3 ein Punkt in genau dieser Entfernung in das Bild gezeichnet. In Abbildung 4.7 wird ein Referenzbild mit zwei bekannten Distanzen gezeigt. Zum einen ist die Unterkante des Schachbretts $0.38m$ und der Bürostuhl $4m$ vom Weltkoordinatenursprung entfernt. Die bunten Punkte zeigen

dabei die Projektion der Punkte $(0.38\ 0\ 0)^T$ und $(4\ 0\ 0)^T$ auf Pixelkoordinaten unter der Annahme des neben dem Punkt stehenden Neigungswinkel (in Radiant). Es ist zu erkennen, dass bei einem Neigungswinkel von 0.515 die beiden Distanzen im Bild richtig berechnet werden. Es ergibt sich somit die Rotationsmatrix (4.9)

$$\begin{pmatrix} cos(0.515) & -sin(0.515) & 0 \\ sin(0.515) & cos(0.515) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{4.9}$$

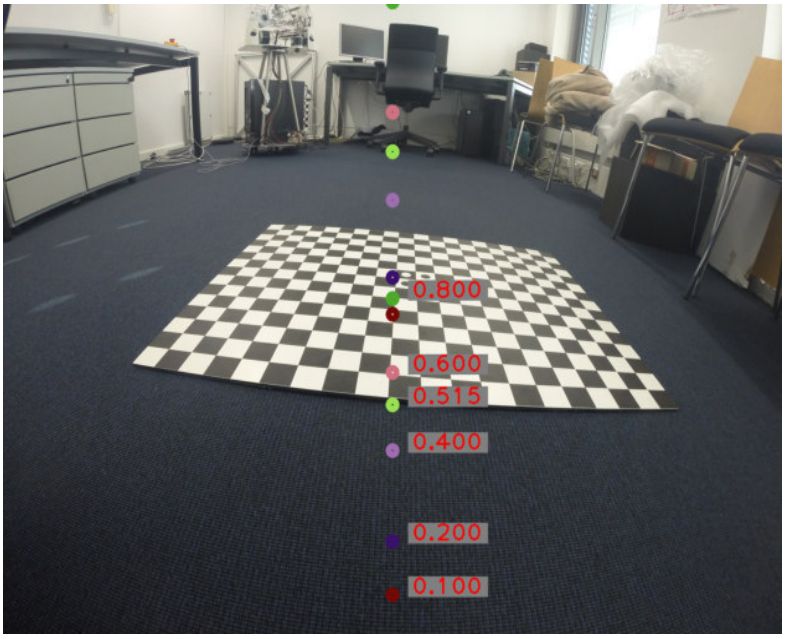



Abbildung 4.7: Experimentelle Bestimmung des Neigungswinkels anhand eines Referenzbildes. Die Unterkante des Schachbretts ist $0.38m$ und der Stuhl $4m$ von der Kamera entfernt.

Für die Erstellung des Tiefenbildes wird für jeden Pixel die Distanz zum Kameraursprung berechnet. Hierbei ist zu beachten, dass ab einer bestimmten Pixelhöhe im Bild der Schnittpunkt zur Bodenebene hinter der Kamera liegt. Die Entfernung zu diesen Pixeln wird auf *unendlich* gesetzt und eine Horizontlinie definiert. In Abbildung 4.8 sind die Tiefeninfomationen als $\frac{1}{Distanz}$ sowie die Horizontlinie in blau dargestellt.
Da die Distanz zu den Pixeln nah am Horizont Werte über 2000 erreicht, wird der Wert als Nenner verwendet. Ansonsten wären geringe Werte im weitaus relevanteren Nahbereich verhältnismäßig zu gering.

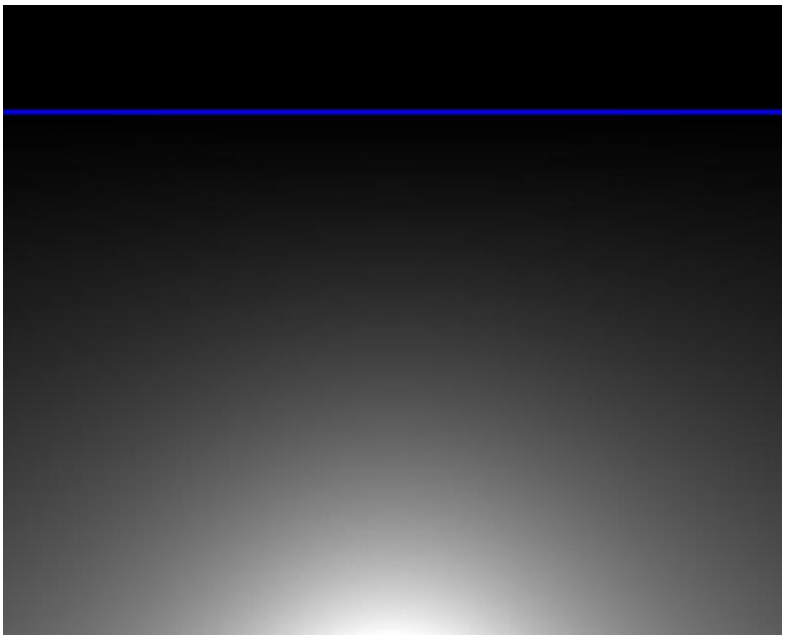


Abbildung 4.8: Darstellung der Tiefeninformationen und der Horizontlinie (blau)

# Lösungsansatz

Zur Lösung des in dieser Arbeit gestellten Problems, wird ein *DCNN* eingesetzt. Die Basis des Ansatzes wird der derzeit (Stand 07.08.2018) führende Ansatz der Pascal VOC 2012 Challenge [Eve+15] *DeepLab* in seiner dritten Version sein. In diesem Kapitel wird zunächst genauer auf DeepLab und die Besonderheiten in den Layern eingegangen, im Anschluss werden die durchgeführten Experimente aufgeführt.

## 5.1 DeepLab 3+ - Basics

Der allgemeine Aufbau von DeepLab 3+ ist bereits kurz im Kapitel 3 und Abbildung 3.10 beschrieben. Des Weiteren werden zwei Konzepte verwendet, die generelle *CNNs* erweitern.
Die Atrous Convolution erweitern Convolutional Layer um eine Rate, nach der einzelne Pixel übersprungen werden. In Abbildung 5.1 *(a)* ist ein eindimensionales Beispiel gegeben, bei dem jedes zweite Feature ausgelassen wird.
Durch dieses Auslassen der Pixel kann die Convolution trotz hoher Auflösung und Anzahl Feature Maps performant durchgeführt werden.
Ebenso werden durch die Rate Informationen aus den umliegenden Regionen, anstatt ausschließlich benachbarter Pixel verwendet. Durch Verändern der Rate lassen sich unterschiedlich große Regionen definieren und eine Robustheit gegenüber Translation oder Skalierung erzeugen, ähnlich wie durch Pooling Layer. Durch die Atrous Convolution kann in DeepLab komplett auf Pooling Layer verzichtet und so der Informationsverlust minimiert werden.

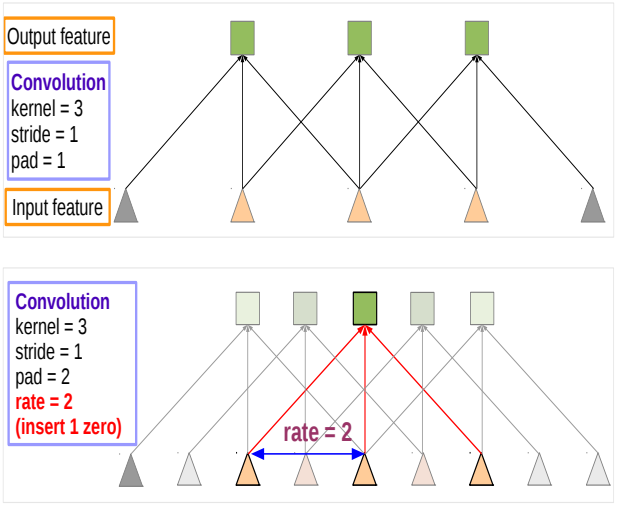



Abbildung 5.1: **Oben:** Normale Convolution mit einer Kernelgröße von 3. **Unten:** Atrous Convolution mit Rate von 2 bei gleicher Kernelgröße.

Beim *ASPP* werden mehrere Atrous Convolutional Layer mit unterschiedlicher Rate parallel angewendet und die Ausgaben konkateniert. Indem Informationen unterschiedlich großer Regionen rund um jedes Pixel beachtet werden, wird ein Pooling Verhalten nachgebildet.
In DeepLab 3+ wird ein *ASPP* Layer nach dem *DCNN* verwendet. Da jedoch bei einer sehr großen Atrous Rate im Verhältnis zur Größe der Feature Map Probleme aufgrund der Bildgrenzen zu beobachten sind, wird zu den Atrous Convolutional Layern noch ein klassischer Average Pooling Layer konkateniert.

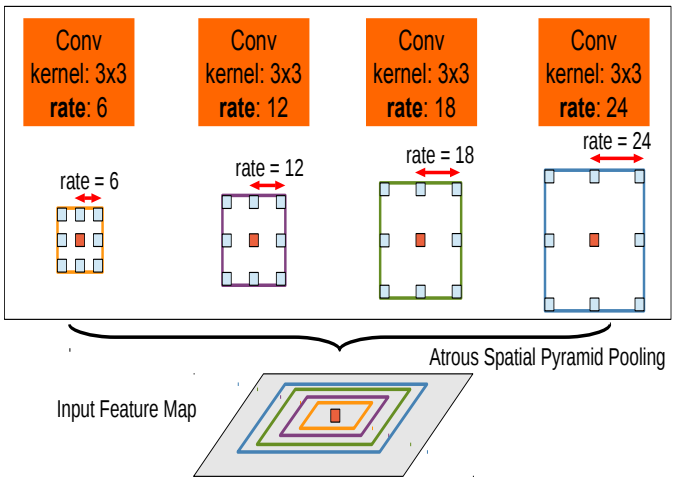
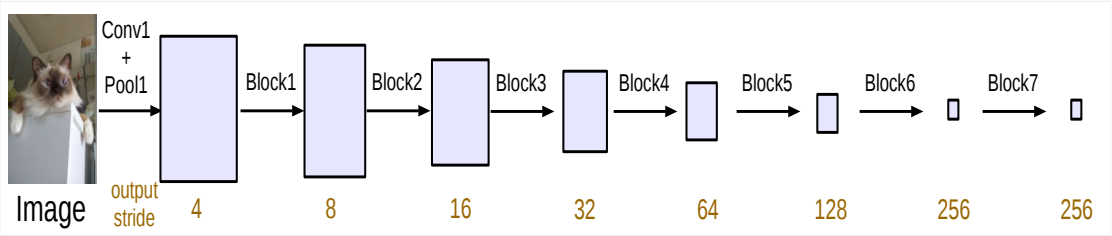
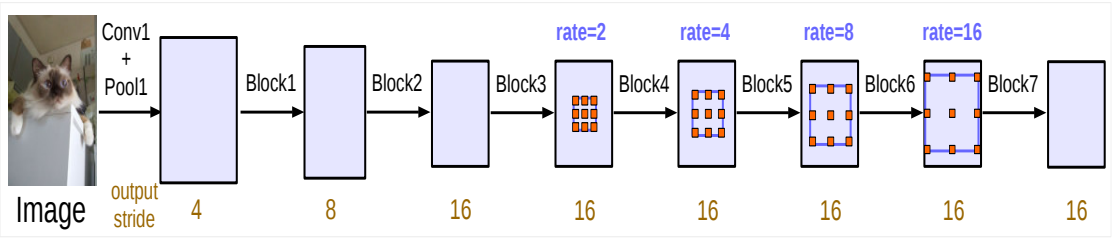



Abbildung 5.2: Beim *ASPP* werden mehrere Atrous Convolutional Layer mit unterschiedlicher Rate parallel angewendet.

Seit DeepLab V2 lässt sich die minimale Größe der Feature Maps relativ zur Eingabegröße mit der *Output stride* begrenzen. Zunächst wird die Eingabegröße mit der *Output stride* dividiert. Mit jedem neuen Block wird die aktuelle Größe mit der Zielauflösung verglichen. Ist die minimale Größe erreicht, wird der Stride Parameter der Convolutional Layer auf stets 1 gesetzt und Atrous Convolution mit ansteigender Rate verwendet.
So wird das Problem einer zu geringen Auflösung von Feature Maps in tiefen Layern umgangen.
Mit Einführung der Encoder-Decoder Struktur in DeepLab 3+ wird der Faktor *Decoder Output stride* verwendet, der die Größe der Feature Map innerhalb des Decoders angibt.




(a) Weitere Tiefe mit normaler Convolution




(b) Weitere Tiefe mit Atrous Convolution

Abbildung 5.3: Aufbau eines tiefen Netzes mit Atrous Convolution statt normaler Convolution. Ab Erreichen der *Output stride* von 16 wird die Rate anstatt der stride erhöht und die Auflösung bleibt konstant.

Durch die insgesamt vier vorhandenen Paper zu DeepLab lassen sich bereits im Vorfeld Aussagen zur Parametrisierung treffen. Generell ist die Belegung der Hyperparameter ein Trade-off zwischen Segmentierungsqualität und Speicher- bzw. Rechenaufwand.

**Eingabegröße**: Für eine gute Qualität sollte die Eingabegröße so groß wie möglich gehalten werden, wobei die Rechenzeit beachtet werden muss. Mit sinkender Eingabegröße sinkt zwar die Rechenzeit, Chen et al. erhalten jedoch durch eine Änderung auf 321x321 statt der verwendeten 513x513 Pixel ein um 9% schlechteres Ergebnis.

**Batchsize**: Auch die Batchsize sollte so hoch wie möglich gewählt werden. Eine starke Verbesserung der Ergebnisse wird durch Batch-Normalisierung gewonnen, die bei einer

größeren Batchsize besser funktioniert. Eine Erhöhung der Batchsize von 4 auf 16 bringt einen Qualitätsgewinn von 13%. Gleichzeitig verringert eine hohe Batchsize die Anzahl der zur Konvergenz nötigen Trainingsiterationen.

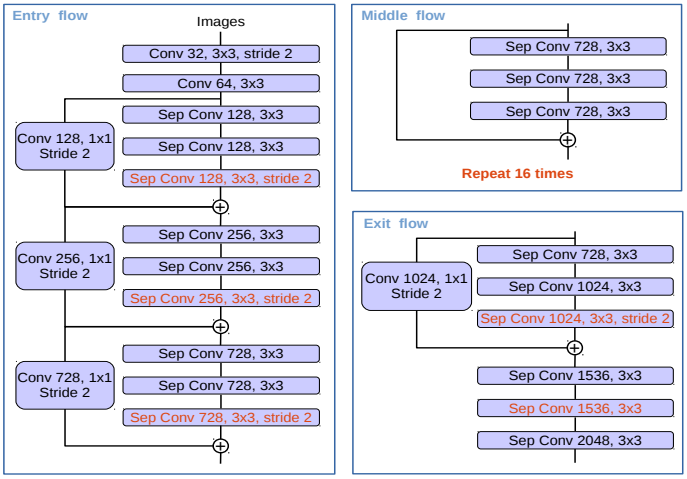
***Output stride***: Die *Output stride* beeinflusst direkt die größtmögliche Batchsize und wurde deswegen in dieser Abhängigkeit evaluiert. Chen et al. erreichen mit einer Kombination aus Trainings-*Output stride* und Batchsize von 16 die besten Ergebnisse. Niedrigere *Output stride* lassen keine ausreichend große Batchsize zu, während größere Werte einen zu großen Informationsverlust erleiden, der auch durch eine große Batchsize nicht auszugleichen ist. Bei gleicher Batchsize ist jedoch eine niedrigere *Output stride* zu bevorzugen. So beobachten sie, dass eine Evaluierungs-*Output stride* von 8 einer von 16 überlegen ist.

### 5.1.1 Limitierungen

Im Vorfeld der Experimente müssen einige Limitierungen der Parametrisierung im Rahmen dieser Arbeit erwähnt werden. Da die zur Verfügung stehende Hardware sehr limitiert ist (siehe Kapitel 2.5), reicht der *Video Random Access Memory (VRAM)* für bestimmte Parametrisierungen nicht aus. Das genaue Trainingssetup der DeepLab Autoren ist nicht einzusehen, jedoch wird ein asynchrones Training auf 50 GPUs oder eine NVIDIA®Tesla®K80 mit 24 GB *VRAM* erwähnt.
Mit der vorhandenen Hardware kann die Eingabegröße nicht erhöht werden, da das Model selbst bei einer Batchsize von 1 zu groß wird. Ebenfalls kann die Batchsize nicht sinnvoll erhöht werden. Bei der Standard Eingabegröße von 385x513 Pixel kann eine maximale Batchsize von zwei angewendet werden. Aufgrund dieser Tatsache und der, das eine Batch Normalisierung erst bei größeren Batches sinnvoll wird, wird die Batchsize im Rahmen aller Experimente auf eins belassen. So zeigen Experimente aus Deeplab V3 [Che+17], dass eine starke Qualitätssteigerung erst ab einer Batchsize von acht auftritt. Vielmehr beginnen Chen et al. diesen Vergleich erst mit einer Batchsize von 4 und empfehlen 16. Auch bei einer geringeren Eingabegröße können diese Parameter nicht erreicht werden.
Eine weitere Limitierung gilt der Anzahl der Trainingsiterationen. Für einen fairen Vergleich der Ansätze muss die Iterationsanzahl in allen Versuchen gleich sein. In diversen Papern (wie z.B. Badrinarayanan et al. [BKC15]) wird gezeigt, dass eine Erhöhung der Iterationen zu einer Verbesserung der Ergebnisse führt. Die Experimente mit Tiefeninformationen benötigen eine große Anzahl an Iterationen, da viele neu initialisierte Gewichte trainiert werden müssen. Gleichzeitig sind die Modelle dieses Ansatzes größer und benötigen mehr Zeit pro Iteration. Die Anzahl der Iterationen wird auf 100000 gesetzt.
Bei den Ansätzen mit Ausschnitten konvergiert der Loss bei dieser Anzahl zwar nicht, da das Training mit Tiefeninformationen jedoch bereits ca. elf Stunden dauert, kann die Anzahl der Iterationen nicht erhöht werden.

## 5.2 Verwendete Grundnetze

Für das *DCNN* im Encoder nutzen Chen et al. Xception-Net [Cho16] und MobileNet [San+18], die in diesem Abschnitt genauer beschrieben werden. Beide Netze nutzen eine Blockstruktur, bei der mehrere Layer in Blöcken zusammengefasst werden. Die Blöcke werden als *linear residual blocks* definiert. Dabei wird nach den Layern innerhalb des Blocks eine *shortcut connection* zur Eingabe erzeugt - die Eingabe wird mit der Ausgabe addiert. Diese Verbindungen haben den Vorteil, dass der Gradient besser auf alle Layer in sehr tiefen Netzen angewendet werden kann. Ohne Shortcuts wird der Gradient in wachsender Tiefe verschwindend gering.
Das Xception-Net (Abbildung 5.4) besteht aus drei verschiedenen Abschnitten. Im *entry flow* werden die ersten Features erzeugt. Im Gesamtkonstrukt von DeepLab 3+ dient die Ausgabe des zweiten Blocks im *entry flow* als eine der Eingaben im Decoder. Der *middle flow* besteht aus 16 identischen Blöcken, in denen die Größe der Feature Map konstant bleibt. Im *exit flow* wird die Anzahl der Kanäle noch einmal stark erhöht.

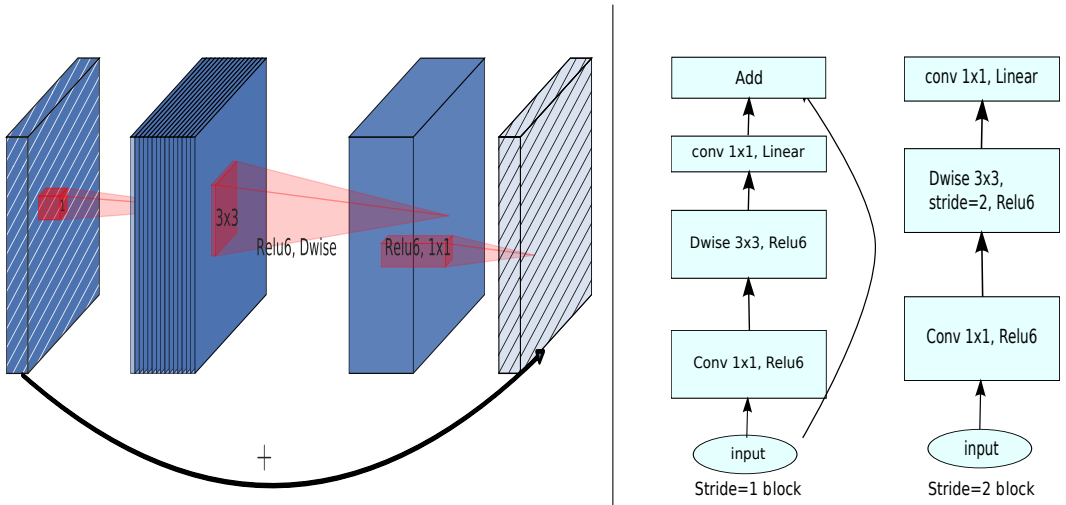



(a) Eingangsblöcke des Netzes (b) Mittlere und letzte Blöcke des Netzes

Abbildung 5.4: Genaue Spezifikation des im DeepLab verwendeten, leicht veränderten Xception-Net.

MobileNet ist ein speziell für mobile Geräte entworfenes Netz. Dabei wurde auf den Speicherbedarf und die Rechenzeit geachtet. Zum Erreichen des Ziels entwerfen Sandler et al. einen neuartigen Layer Block, den *inverted residual with linear bottleneck* (Abbildung 5.6 *(a)*). Die Eingabe dieses Blockes ist eine Feature Map mit einer geringen Anzahl Dimensionen, die in einem ersten Schritt mit einer 1x1 Convolution erweitert wird. Darauf folgt ein Convolutional Layer und zuletzt wird die Anzahl der Kanäle wieder reduziert.

Abbildung 5.6*(b)* zeigt die Unterschiede der Blöcke je nach ihrer stride im mittleren Layer des Blocks.




(a) Inverted residual block

(b) Bottleneck blocks nach Stride

Abbildung 5.5: Genaue Spezifikation der Blockstruktur von MobileNet. In (a) wird die Erweiterung und Verringerung der Dimensionen visualisiert. (b) zeigt den unterschiedlichen Blockaufbau je nach verwendeter stride im Convolution Layer.

Der komplette Aufbau des MobileNets, wie es im DeepLab verwendet wird, ist in Abbildung 5.6 skizziert. Im Vergleich zur Originalfassung aus dem Paper von Sandler et al. [San+18] wird der letzte Convolutional Layer in DeepLab entfernt, um die Anzahl der Dimensionen der Ausgabe geringer zu halten.

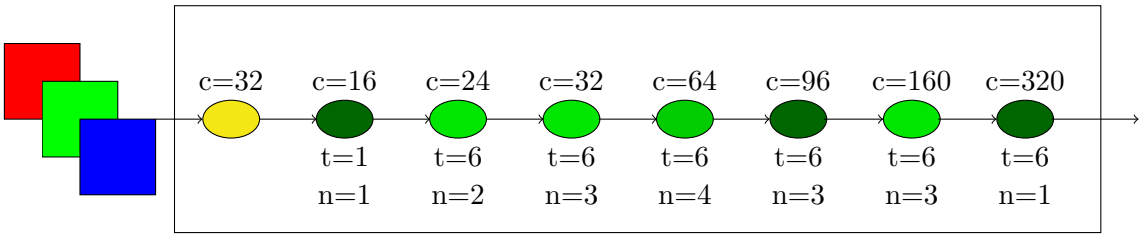


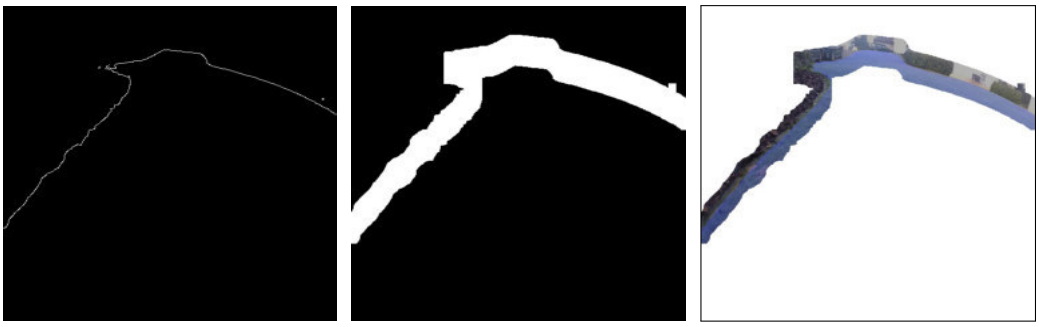

Abbildung 5.6: Aufbau von MobileNet von Sandler et al. [San+18]. Die grünen Knoten stehen für Bottleneck Blocks aus Abbildung 5.5 (b), wobei hellgrüne eine stride von 2 und dunkelgrüne eine stride von 1. $c$ gibt die Ausgabedimension des Blocks, $t$ den Erweiterungsfaktor und $n$ die Anzahl der Wiederholungen je Block an. Der gelbe Knote steht für einen Convolutional Layer mit einer stride von 2

Im Vergleich zum Xception-Net ist die Segmentierung innerhalb von DeepLab zwar weitaus schneller, dafür aber auch qualitativ schlechter. Einerseits liegt das am generell weniger komplexen Modell, andererseits jedoch auch an der Tatsache, dass Chen et al. beim MobileNet sowohl Decoder, als auch *ASPP* aus dem Gesamtaufbau ausklammern. Zwar erhöht beides die Segmentierungsqualität, Chen et al. legen jedoch mehr Gewicht auf die Geschwindigkeit des Netzes. Auch in dieser Arbeit wird MobileNet sowohl mit als auch ohne Decoder und *ASPP* evaluiert. Im Folgenden wird die Version *MobileNet ASPP* genannt.

## 5.3 Bewertungskriterium: *RoI-MIOU*

Bereits früh im Verlauf der Experimente wird ersichtlich, dass der *MIOU* als Bewertungskriterium nicht optimal für das Szenario geeignet ist. Durch die großen, eher leicht zu segmentierenden Flächen ist der *MIOU* oft sehr hoch. Für ein möglichst exaktes Navigieren ist der Randbereich der Rasenflächen der Relevanteste. Problematisch ist, dass dieser aufgrund seiner verhältnismäßig geringen Fläche, bei der Berechnung des *MIOU* kaum ins Gewicht fällt. In Abbildung 5.14 sind Beispiele für fehlerhafte Grenzbereiche bei sehr gut bewerteten Bildern zu sehen.
Für eben solche Fälle wird eine *Region of Interest (RoI)* entlang der Rasenkante und daraus der *RoI-MIOU* definiert. Die *RoI* wird aus dem Ground Truth extrahiert und auf die Prediction angewandt (siehe Abbildung 5.7). Im ersten Schritt wird mit einem *Canny Edge Detector* die Rasenkante im Binärbild bestimmt. Diese Kante wird stets zuverlässig erkannt, da ein Binärbild verarbeitet wird.
Die Kante wird im nächsten Schritt mit der OpenCV Funktion *dilate*[1] erweitert. Dabei wird die Breite der *RoI* bei 385x513 Bildern auf 50 Pixel und bei 3000x4000 auf 150 Pixel gesetzt.
Im letzten Schritt werden alle Pixel außerhalb der *RoI* auf einen default Wert von 255 gesetzt. Der *RoI-MIOU* ignoriert diesen default Wert und bewertet ausschließlich die Prediction an der Rasenkante. In Abbildung 5.7 (c) ist zu sehen, dass ein Großteil des Bildes, nämlich genau die großen Flächen, ignoriert werden und der Bereich um die Rasenkante verbleibt.

[1] OpenCV Dokumentation: `https://docs.opencv.org/3.4.0/db/df6/tutorial_erosion_dilatation.html`



(a) Rasenkante aus Ground Truth extrahiert. (b) Kante mit Dilate verbreitert. (c) *RoI* angewandt auf die Prediction.

Abbildung 5.7: Erzeugen der *RoI* beginnend von der Rasenkante aus dem Ground Truth. In (c) ist ausschließlich die Prediction entlang der Rasenkante relevant.

## 5.4 *Conditional Random Field (CRF)*

Auch in dieser Arbeit wird zur Verbesserung der Ergebnisse ein *CRF* genutzt. Es wird die Implementierung von Krähenbühl et al. [KK11] verwendet, deren Vorzüge im State of the Art Kapitel 3 beschrieben werden. Es handelt sich dabei um eine C++-Implementierung, die die Autoren auf ihrer Website[2] veröffentlichen.
Da die unären Potentiale zur Initialisierung der *CRF* vom DeepLab bereitgestellt werden, wird die Implementierung noch erweitert. Die Ergebnisse der DeepLab Inferenz werden als *npy* Dateien gespeichert und in der C++-Implementierung mithilfe der *cnpy* Bibliothek[3] geladen.
Zunächst wird hierfür die *argmax*-Funktion, für die pixelweise Klassifizierung entfernt. Somit ist die Ausgabe von DeepLab nun dreidimensional. Für jedes Pixel erhält man ein Potential für jede Klasse. Diese Potentiale dienen direkt der Initialisierung des *CRF*.
Betrachtet man diese Potentiale fällt auf, dass der Unterschied der zwei Klassen im Grenzbereich geringer wird (siehe Abbildung 5.8). Dieser geringe Unterschied lässt sich als eine *Unsicherheit* der Segmentierung interpretieren. Der Argmax-Klassifizierer beachtet dies jedoch nicht. Ein gut parametrisiertes *CRF* kann aus diesem geringem Unterschied eine richtige Segmentierung erzeugen.

[2] Website Krähenbühl et al: `http://graphics.stanford.edu/projects/drf/`, Abrufdatum: 05.06.2018
[3] CNPY Bibliothek: `https://github.com/rogersce/cnpy`, Abrufdatum: 05.06.2018


(a) Eingabebild mit schmalen Pfeiler

(b) Absoluter, normierte Differenz zwischen den Beiden Klassen

Abbildung 5.8: Differenz der Klassenpotentiale. Im Grenzbereich wird der Wert sehr schnell gering.

### 5.4.1 Parametertuning

Die Qualität der *CRF*-Ergebnisse hängt unter anderem von den Parametern ab, die die paarweise Potentialberechnung beeinflussen. Dieses Parameterset besteht aus den farbunabhängigen (smoothness kernel) $x_{devG}$ und $y_{devG}$ und den farbabhängigen Parametern (appearance kernel) $x_{dev}$, $y_{dev}$, $r_{dev}$, $g_{dev}$ und $b_{dev}$ sowie den Gewichtungen der beiden Kernel $p$ und $pp$ (Vergleiche Gleichung 2.6).
Wie Krähenbühl et al. [KK11] beschreiben, werden die Parameter am Besten mit einem Grid Search Verfahren optimiert. Hierfür wird zunächst eine Auswahl aus dem Datenset zum Training ausgewählt. Da das *CRF* im Rahmen dieser Arbeit vor allem die Randbereiche optimieren sowie harte Kanten an Objekten wie Bäumen erzeugen soll, werden zehn Bilder nach diesen Kriterien ausgewählt. Das Datenset zum Training muss entsprechend klein gehalten werden, um die Grid Search überhaupt durchführen zu können. Außerdem müssen die Parameter gruppiert werden, da eine vollständige Grid Search ebenfalls zu viel Zeit benötigt.
Für eine Grid Search Iteration wird für jeden Parameter eine Anzahl möglicher Belegungen definiert und für jede mögliche Kombination dieser Belegungen das *CRF* Ergebnis evaluiert werden. Anhand des Maximums der Evaluierungen kann dann die beste Belegung der Iteration bestimmt werden. Für die nächste Iteration werden dann die Belegungen der Parameter im Bezug zum vorherigen Maximum neu belegt.
Bei einer vollen Grid Search mit neun Parametern und drei Belegungen müssen jedoch $3^9 = 19683$ Evaluierungen des gesamten Datensets durchgeführt werden. Für ein Datenset aus zehn Bildern und einer hypothetischen Evaluierungsdauer von zwei Belegungen pro Sekunde ergibt sich eine Dauer von ca. 30 Stunden pro Iteration.
Eine Gruppierung der Parameter in drei Untermengen lässt sich aus der Definition des

Ansatzes von Krähenbühl et al. erstellen. In den zwei Kerneln werden die Parameter bereits in drei Gruppen eingeteilt, die einzeln optimiert werden können:
**G1:** $x_{devG}, y_{devG}, p$; **G2:** $x_{dev}, y_{dev}, pp$; **G3:** $r_{dev}, g_{dev}, b_{dev}$
Durch diese Gruppierung sowie einer Parallelisierung der Gruppen, lässt sich die benötigte Zeit pro Iteration auf ca 10 Minuten reduzieren. Über mehrere Iterationen werden dann für die einzelnen Gruppen Belegungen definiert. Nach jeder Iteration werden die besten Ergebnisse einer Gruppe als konstant in den anderen Gruppen gesetzt. Ändert sich die beste Belegung einer Gruppe nach einer Iteration nicht mehr, werden die möglichen Belegungen für die nächste angepasst. Der beste Parameter liegt dabei im Zentrum der nächsten Belegungen.
In Abbildung 5.9 ist die iterative Verbesserung gemäß des *RoI-MIOU* aufgezeigt. In den ersten Iterationen sind die *CRF* Ergebnisse oft noch schlechter als die ursprüngliche Segmentierung. Zum Ende vom Grid Search wird die Qualität stets besser und die Schwankungen werden geringer, da die getesteten Parameter weniger auseinanderliegen. Die so optimierten Parameter verbessern die Ergebnisse für die zehn Trainingsbilder um ca. einen halbes Prozent. Das Optimieren der Parameter wird abhängig vom Datenset durchgeführt, da in einigen Experimenten Datensets aus Ausschnitten der Bilder verwendet werden. Bei diesen ist das Erscheinungsbild von Objekten gänzlich anders. Vor allem entfernte Objekte werden beim Skalieren des Gesamtbildes stark verkleinert. In Ausschnitten bleiben diese im Vergleich weitaus größer.

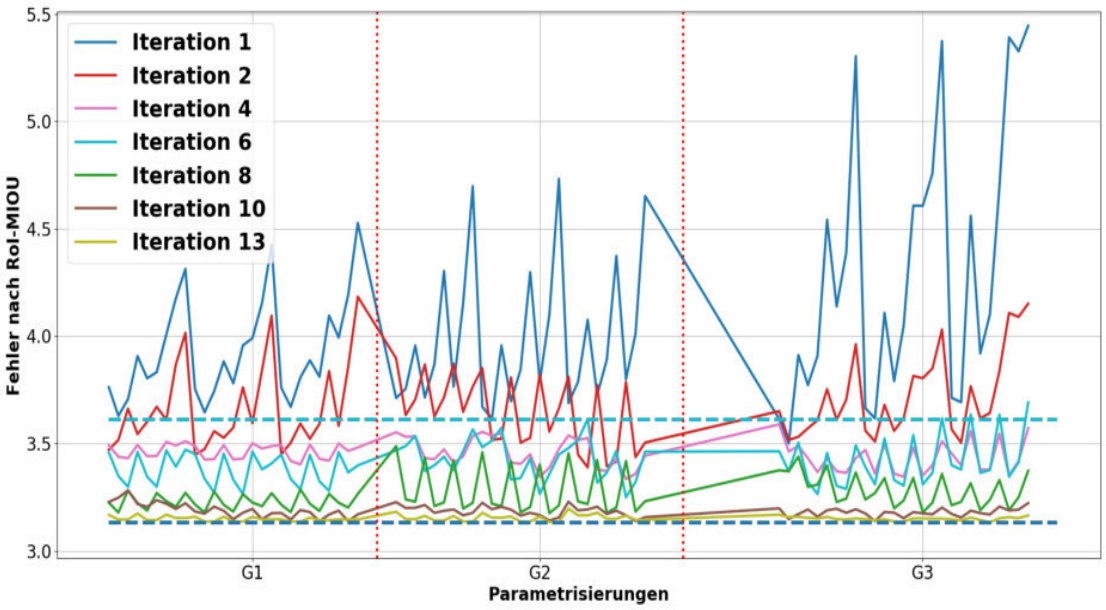



Abbildung 5.9: Iterative Verbesserung des *RoI-MIOU* über 13 Iterationen. Die Kurven beschreiben die Fehlerverläufe aller Parameter einer Iteration. Die roten vertikalen Linien kennzeichnen die Grenzen der drei Parametergruppen. Die untere horizontale gestrichelte Linie gibt den Fehler des *DCNN* an, die obere die beste *CRF* Optimierung.

## 5.5 Experimente

Im Folgenden werden die durchgeführten Experimente sowie die Ergebnisse vorgestellt. Evaluiert wird vor allem die Genauigkeit und Geschwindigkeit der verschiedenen Ansätze. Da die Eingabegröße der Bilder variabel ist und innerhalb der Experimente verändert werden kann, wird für eine Vergleichbarkeit der Ansätze die Verarbeitung eines 3000x4000 Pixel großen Bildes betrachtet. Die Qualität wird dabei mit dem Fehler ($100 - mio$ in Prozent) bzgl. des als Standardmaß geltenden *MIOU* (Kapitel 2.3) sowie dem hier eingeführten *RoI-MIOU* gemessen.
Es werden zunächst *Baseline*-Experimente aus den DeepLab Papern durchgeführt, um eine Vergleichsbasis für weitere Experimente zu schaffen. Aus der *Baseline* werden die besten Konzepte für beide Grundnetze ausgewählt, die in weiteren Experimenten verwendet werden.

### 5.5.1 Experiment 1: *Baseline* (V1)

Im ersten Versuch wird das erstellte Datenset auf die unveränderte Version von DeepLab 3+ angewendet. Im Vorfeld wurden die Bilder sowie der Ground Truth, dafür auf 385x513 Pixel skaliert. Dies entspricht in etwa der Eingabegröße der DeepLab Paper, erhält aber die Seitenverhältnisse des Datensets.
Das Trainingssetup besteht dabei zunächst aus den ursprünglichen Parametern der veröffentlichen Trainingsskripte (Abbildung 5.10). Trainiert wird für 100000 Iterationen bei einer Lernrate von 0.001.

| | Lernrate | Iterationen | Atrous Raten | Decoder Skalierung | Output Skalierung |
|---|---|---|---|---|---|
| (a) Standardparameter | 0.001 | 100000 | $[6, 12, 18]$ | 4 | 16 |
| (b) *ASPP* halbiert | 0.001 | 100000 | $[3, 6, 9]$ | 4 | 16 |
| (c) Gewichteter Loss | 0.0001 | 100000 | $[3, 6, 9]$ | 4 | 16 |

Abbildung 5.10: Trainingssetup des ersten Versuchs für beide Netze

Im Trainingsverlauf fällt auf, dass das Trainingsloss bereits sehr schnell (nach wenigen tausend Iterationen) konvergiert, der Validierungsfehler jedoch bis zum Schluss weiter sinkt. Beide Netze zeigen ein sehr ähnliches Verhalten beim Training.

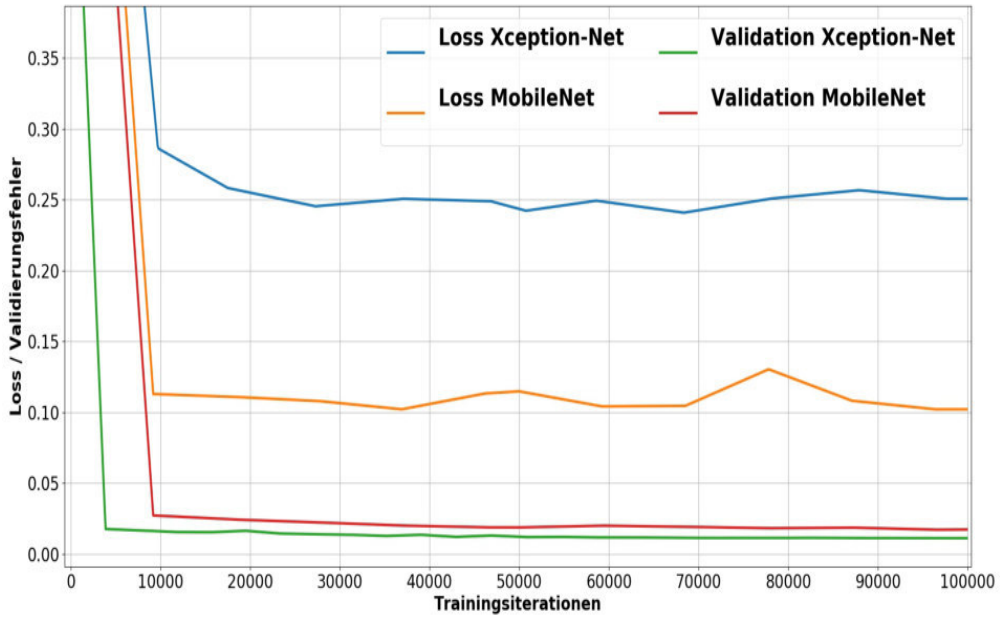



Abbildung 5.11: Trainingsverläufe des ersten Experiments. Der Trainingsloss konvergiert bereits nach wenigen tausend Iterationen, der Validierungsfehler sinkt jedoch beständig. Die Losskurve beschreibt den softmax cross-entropy Loss (Kapitel 2.1.1), der Validierungsfehler gemäß *MIOU* (Kapitel 2.3)

Der Fehler im *MIOU* über die Testdaten beträgt für 385x513 beim Xception-Net 1.27% und 1.36% für 3000x4000. Beim MobileNet ergibt sich ein *MIOU* von 1.75% bei 385x513 und 1.80% bei 3000x4000.
Zur Ermittlung der Qualität wurde die Ausgabe von DeepLab vor der Klassifizierung auf die gewünschte Größe hochskaliert. Es fällt direkt auf, dass beide Werte sich kaum unterscheiden. Die Qualität scheint also durch die Skalierung nur wenig zu sinken. Ein Beispiel für die sehr guten Ergebnisse ist in Abbildung 5.12 aufgeführt. Anhand dieses Beispiels ist zu sehen, dass der hohe *MIOU* relativiert werden muss. Ein Großteil des Bildes nimmt eine einfach zu segmentierende Rasenfläche sowie der Horizont ein. Die im Szenario wirklich entscheidenden Bereiche definieren sich aber durch genau die Ränder zwischen Rasen und Begrenzung, die jedoch durch die Größe der anderen Flächen kaum ins Gewicht fallen.

(a) Xception-Net

(b) MobileNet

(c) *MobileNet ASPP*

Abbildung 5.12: Typisches Beispiel des Datenset. Eine offene Rasenfläche ohne Hindernisse, begrenzt von Büschen. Ein Unterschied zwischen den drei Modellen ist kaum erkennbar.

In Abbildung 5.14 ist das Problem exemplarisch dargestellt. Alle drei Ausschnitte haben einen hohen *MIOU*, in den Grenzbereichen ist die Segmentierung jedoch nicht optimal. Vor allem in *(a)* wäre ein Fahren gegen die großen Steine als kritisch zu beurteilen.
In *(b)* und *(c)* wird neben dem schlecht segmentiertem Grenzbereich das Problem von schmalen Rasenbereichen deutlich. Diese sind oft in den niedrig skalierten Bildern zu schmal. Durch die Atrous Convolution und das *ASPP* werden viele Kontextinformationen aus der Umgebung eingebunden, so dass der Grasbereich nicht mehr abgegrenzt werden kann.
Um diese Beobachtung in die Qualitätsbewertung der Experimente einfließen zu lassen, wird in dieser Arbeit der *MIOU* auf genau den Grenzbereich angewendet (Siehe Kapitel 5.3)
In Abbildung 5.13 ist der Vergleich vom *MIOU* zum *RoI-MIOU* verdeutlicht. Es fällt auf, dass gerade bei der Skalierung auf 3000x4000 die Qualität in der *RoI* stark sinkt. Dies ist darauf zurück zu führen, dass kleine Fehler bei der Skalierung auf die 8-fache Größe mit skaliert werden und somit deutlich stärker ins Gewicht fallen.

| | 385x513 | 3000x4000 | 385x513 *RoI* | 3000x4000 *RoI* |
|---|---|---|---|---|
| Xception (a) | 1.27% | 1.36% | 3.03% | 6.96% |
| Xception (b) | 1.15% | 1.24% | 2.68% | 6.16% |
| MobileNet | 1.75% | 1.80% | 2.87% | 8.14% |
| *MobileNet ASPP* | 1.68% | 1.75% | 2.74% | 8.01% |

Abbildung 5.13: Fehler bzgl. des *MIOU* und *RoI-MIOU* des ersten Versuchs für beide Netze, jeweils in Eingabegröße 385x513 sowie skaliert auf 3000x4000

(a) Große Steine an der Grenze als Rasen segmentiert

(b) Schmaler Rasenabschnitt nicht richtig segmentiert; Terrasse *abgeschnitten*

(c) Steinabgrenzung als Rasen. Schmaler Rasenabschnitt nicht segmentiert.

Abbildung 5.14: Fokus auf den Grenzbereich, in dem falsch segmentiert wurde.

Weitere Probleme durch niedrig skalierte Eingabe sowie dem *Verschwinden* von Objekten durch Kontextinformationen werden in Abbildung 5.15 deutlich. In beiden Ausschnitten ist zu sehen, dass die Grenze stets einen geschwungenen Verlauf nimmt und fast nie harte Winkel vorhanden sind. Auch werden oftmals Bäume scheinbar übergangen, jedoch durch eine Senke im Grenzverlauf angedeutet. Der gesamte Stamm wird, genau wie der schmale Grasbereich, durch Kontextinformationen überdeckt.
Beide Beobachtungen sind auch auf das in zwei Schritten durchgeführte 16-fache Upsampling im Decoder zurückzuführen. Spitze Winkel sowie ein Wechsel der Klasse innerhalb dieser Skalierung sind nicht möglich. Ein Hinweis darauf wird auch in den Beobachtungen in Abbildung 5.8 gegeben. Hier ist der Stamm in der Differenz der Klassenpotentiale deutlich zu erkennen, die mähbar-Klasse ist jedoch noch leicht überlegen.

(a) Geschwungene Kante, statt spitzer Winkel.

(b) Bäume werden nur durch leichte Senkung im Grenzverlauf angedeutet.

Abbildung 5.15: Typische Probleme der Segmentierung. Enge Konturen an Objekten werden oft nicht richtig abgebildet. Sind die Objekte zu schmal werden sie oft übergangen.

Eine weitere mögliche Erklärung dieser Beobachtung lässt sich in den Atrous Raten finden. In der skalierten Eingabe von Abbildung 5.15 *(b)* ist der Baum ausschließlich 11 Pixel breit und komplett von Rasenfläche umgeben. Da im *ASPP* mit hohen Atrous Raten Umgebungsinformationen eingefasst werden, kann der sehr stark überlegene Anteil an Rasen die wenigen Baumpixel überstimmen. Eine theoretische Überlegung zu dieser Erklärung ist in Abbildung 5.16 skizziert. Im rechten Bild ist der Anteil an nicht mähbaren Flächen ausgeglichen und die charakteristische Senke wird erzeugt. Im linken hingegen ist neben dem zentralen Pixel kaum ein Pixel kein Rasen.

Abbildung 5.16: Theoretische Betrachtung des *ASPP* am Eingabebild. Die verschiedenen Atrous Raten (farbige Quadrate) fassen die Regionen um ein blaues Pixel zusammen. Im linken Bild sind beide Regionen beinahe vollständig mit Rasen gefüllt, rechts ist die Verteilung ausgeglichener.

Aus dieser Überlegung werden die Atrous Parameter für einen neuen Testlauf halbiert (siehe Abbildung 5.10(b)). Diese Halbierung sorgt wie erwartet für eine starke Verbesserung der Randbereiche und bestärkt die theoretische Überlegung aus Abbildung 5.16. Im Vergleich der Ergebnisse (Abbildung 5.13) fällt auf, dass die ersten drei Spalten nahezu identisch bleiben. Nach der Skalierung auf 3000x4000 Pixel beträgt der Unterschied in der *RoI* jedoch 1% und stellt somit eine starke Verbesserung dar. In Abbildung 5.17 werden die Vorzüge der angepassten Parameter dargestellt. Typische Fehler des ersten Versuchs werden hier stark verbessert. Kanten von Objekten werden klarer abgebildet und auch schmale Grasbereiche besser segmentiert.
Aus diesen Erkenntnissen wird im Folgenden die halbierte Atrous Rate verwendet.

(a) Der Fehlerfall aus Abbildung 5.14 wird mit klarer Kante segmentiert.

(b) Spitze Winkel an der Bank werden besser abgebildet.

(c) Klare Kanten an Bäumen werden gut segmentiert.

Abbildung 5.17: Eine Halbierung der Atrous Raten verbessert die Segmentierung im Randbereich stark.

Bei der Evaluierung von MobileNet fällt auf, dass die Qualität im weit entfernten Bereich oft fehlerhaft ist. In Abbildung 5.18 sind Beispiele dieser Fehler aufgezeigt. Es sind drei Ausschnitte aus dem oberen Bildbereich ausgewählt. In den Bildern ist zu beobachten, dass im oberen Bildbereich Fehler in der Segmentierung auftauchen, die im Nahbereich nicht auftreten. Ebenfalls scheint die Erweiterung um *ASPP* und Decoder kaum eine Verbesserung zu erbringen. Dies könnte darauf zurückzuführen sein, dass die entsprechenden Layer nicht im vortrainierten Model vorhanden sind und somit weitaus weniger optimiert wurden, um eine spürbare Verbesserung zu erwirken. Da MobileNet ohne *ASPP* und Decoder zudem noch ca. ein Fünftel *FLOPS* spart, wird es weiterhin ausschließlich ohne beides weiterverwendet.
In *(c)/(d)* wird der Randbereich zur Hecke oben nicht richtig abgebildet. Ähnliche Kanten im Nahbereich sind jedoch keine Probleme. In den Bildern *(e)/(f)* ist unten links ein kleiner Teil einer guten Segmentierung entlang der Terrasse zu sehen. In den oberen Bildbereichen werden diese Kanten nicht mehr in der gleichen Qualität abgebildet. Der schmale Grasstreifen über der Terrasse zeigt ebenfalls einen typischer Fehlerfall der MobileNet Segmentierung.
Aus diesen Beobachtungen ist zu schließen, dass eine höhere Auflösung und klarere Kanten für eine sehr gute Segmentierung im Fernbereich benötigt werden.
Des Weiteren ist in den Bildern *(a)/(b)* zu beobachten, dass MobileNet fehleranfälliger für intensive Lichteinstrahlung als das Xcepion-Net ist.


Abbildung 5.18: Drei Ausschnitte der Probleme der MobileNet Segmentierungen im oberen Bildbereich. In **(a)/(b)** sorgt die Sonneneinstrahlung für Probleme. Stark fehlerhafte Randbereiche wie in **(c)/(d)** sind ebenfalls oft zu beobachten. Schmale Graszungen wir in **(e)/(f)** werden nur in den seltensten Fällen erkannt.

### Baseline: Gewichteter Loss

Aus der *Baseline* wird das Xception-Net mit halbierter Atrous Rate und MobileNet ohne Decoder und *ASPP* ausgewählt.
Als erste Strategie wird versucht, das Modell mit Fokus auf den Randbereich zu trainieren. Hierfür wird nach Vorbild von Ronneberger et al. [RFB15] für jedes Bild des Trainingssets eine Gewichtung der Pixel aus dem Ground Truth berechnet. Diese Gewichtung reicht dabei von 11 genau an der Rasenkante bis 1 an den entferntesten Punkten.
Für die Gewichtung $w$ eines Pixels $x$ wird mit der Gleichung (5.1) eine Gaußverteilung der Gewichte angenommen.

$$w(x) = 1 + w_0 \cdot e^{\left(-\frac{d(x)}{2 \cdot \sigma^2}\right)} \tag{5.1}$$

Dabei bezeichnet $d(x)$ die Distanz eines Pixels zum nächstgelegenen Übergang zwischen den zwei Klassen. Die Parameter $w_0 = 10$ und $\sigma = 5$ werden dabei genau wie von Ronneberger et al. gewählt. Die so errechneten Gewichte (Abbildung 5.19) erzielen einen deutlichen Fokus auf die Randbereiche. Da jedoch die Gewichte weit entfernt von der Kante sehr nah gegen null gehen, laufen die ersten Tests in Probleme der Floating Point Precision. Innerhalb der Lossberechnung wird ein Logarithmus verwendet, wobei dies bei den sehr geringen Gewichten aufgrund der Präzision auf ein $log(0)$ hinaus läuft. Um diesem Problem entgegenzuwirken, wird die Berechnung von Ronneberger et al. um eine Addition mit 1 erweitert.

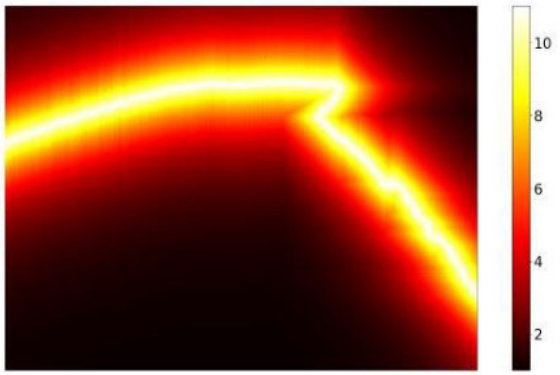



Abbildung 5.19: Gewichtung der Pixel nach Vorbild von Ronneberger et al. [RFB15]. Die Gewichtung nimmt in beide Richtungen von der Rasenkante ab. Die äußeren Bildränder werden mit 1 gewichtet.

Das Trainingssetup vom Experiment *V1* mit gewichtetem Loss ist in Tabelle 5.10 beschrieben. Aufgrund des gewichtetem Loss muss die Lernrate hierbei verringert werden. Trotz der geringeren Lernrate sind große Schwankungen in der Losskurve zu beobachten (siehe Abbildung 5.20), die innerhalb der 100000 Iterationen nicht konvergieren. Dieses Verhalten ist auf die unterschiedliche Komplexität der Randbereiche zurückzuführen. So ist durch einen komplexen Randbereich die Segmentierung schlechter und durch die hohe Gewichtung von diesem der Loss entsprechend höher, während ein leichter zu segmentierender Randbereich in einen geringeren Loss resultiert.
Trotz des ungleichmäßigen Loss sinkt der Validierungsfehler beständig. Ein Unterschied der *DCNNs* ist beim Training nicht zu beobachten.

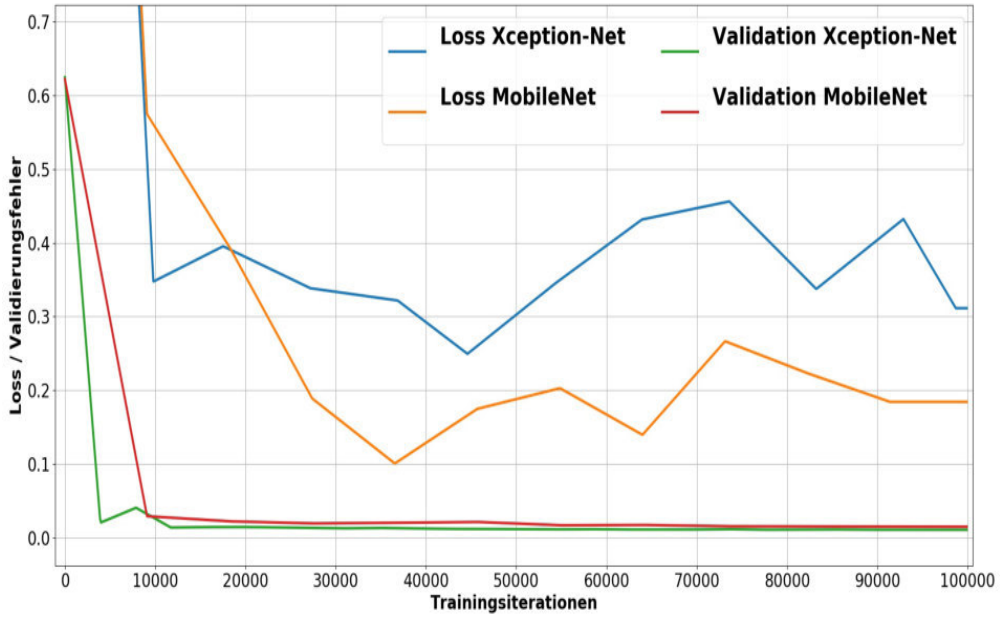



Abbildung 5.20: Trainingsverläufe des ersten Experiments mit gewichteten Loss. Die Losskurven schwanken zunächst weitaus stärker und konvergieren sehr langsam. Der Validationfehler sinkt trotz dieser Schwankungen beständig, da die Gewichtung ausschließlich bei der Lossberechung verwendet wird. Die Losskurve beschreibt den softmax cross-entropy Loss (Kapitel 2.1.1), der Validierungsfehler gemäß *MIOU* (Kapitel 2.3)

In Abbildung 5.21 sind die Ergebnisse mit gewichtetem Loss aufgezeigt. Es fällt auf, dass die Gewichtung des Loss für das Xception-Net kaum eine Veränderung erzeugt. Beim MobileNet ist ebenfalls nur eine geringe Verbesserung um ca. 1% zu erkennen. Die Tests zeigen, dass keine Nachteile in der Segmentierung der großen Flächen durch die niedrigere Gewichtung auftreten. Da die für die Gewichtung rechenintensive Berechnung im Vorfeld durchgeführt werden kann und während des Trainings keine Steigerung der Rechenzeit oder Komplexität resultiert, kann die Gewichtung ohne Nachteile in beiden Netzen verwendet werden.

| | 385x513 | 3000x4000 | 385x513 *RoI* | 3000x4000 *RoI* |
|---|---|---|---|---|
| Xception (b) | 1.15% | 1.24% | 2.68% | 6.16% |
| MobileNet (a) | 1.75% | 1.80% | 2.87% | 8.14% |
| Xception (c) | 1.14% | 1.22% | 2.63% | 6.05% |
| MobileNet (c) | 1.53% | 1.58% | 2.37% | 7.21% |

Abbildung 5.21: Erweiterung der Abbildung 5.13 um die Ergebnisse mit gewichtetem Loss

In Abbildung 5.22 ist zu sehen, dass die hohe Gewichtung der Kante in schwierig zu segmentierenden Fällen eine Verbesserung bringt. Ohne gewichteten Loss sind oft leichte Ausbeulungen auf die Steinbegrenzungen oder aber ein größerer Abstand der mähbaren Fläche zu beobachten. Mit gewichtetem Loss hingegen werden Fehler in beide Richtungen ausgebessert und die segmentierte Kante läuft weitaus besser an der realen Kante entlang. Vor allem im extremen Nahbereich, wie der oberen Zeile, in dem die Auflösung so hoch ist, dass zwischen einzelnen Grashalmen die Steinbegrenzung zu sehen ist, ist die genauere Kante wichtig, da so eine zentimetergenaue Navigation ermöglicht wird.

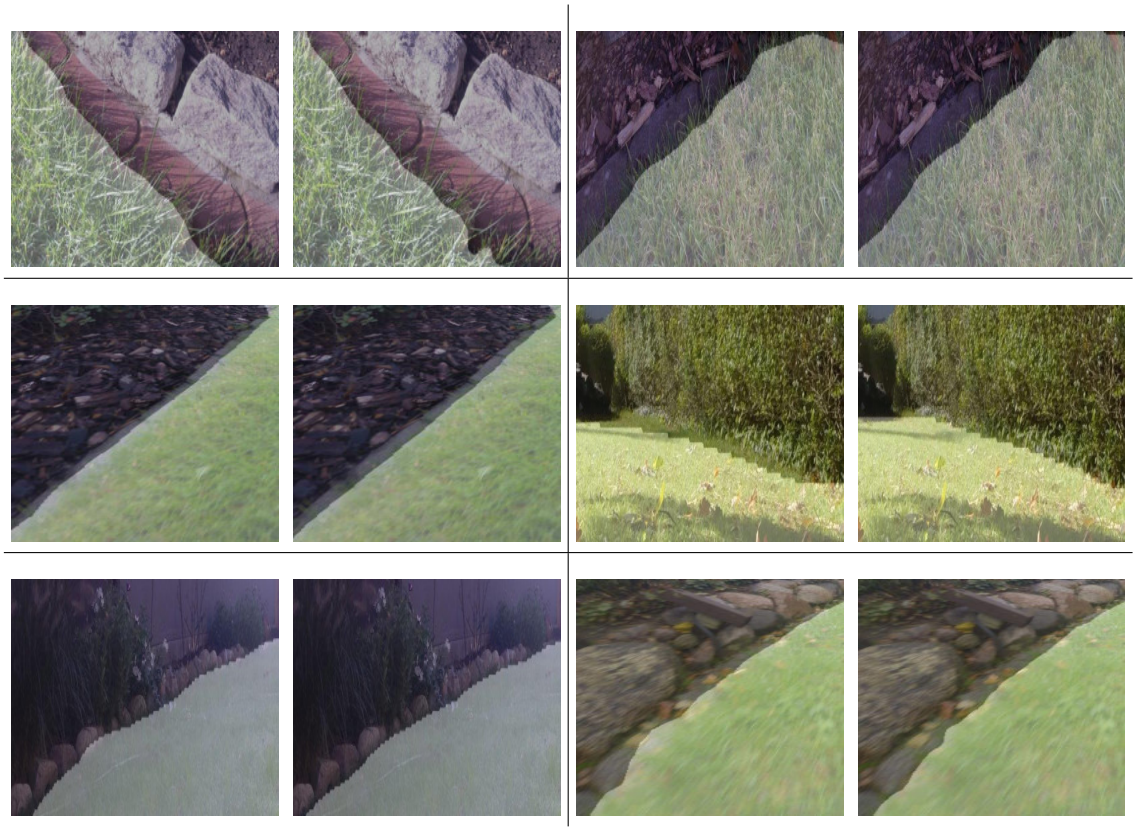


Abbildung 5.22: Genauere Segmentierung der Rasenkante mit MobileNet und gewichteten Loss. Das linke Bild der jeweiligen Paare entstammt der Segmentierung ohne, das rechte mit gewichtetem Loss. Bis auf die Hecke (zweite Zeile; rechts) befinden sich alle Kanten im Nahbereich. Im extremen Nahbereich (obere Zeile) ist der Unterschied nur marginal, jedoch ergeben in dieser Entfernung auch kleine Verbesserungen einen entscheidenden Informationsgewinn.

### Baseline: *CRF*

Im nächsten Schritt wird im Post-Processing ein *CRF* verwendet. In den Kapiteln 2.2 und 5.4 werden die theoretischen Hintergründe des *CRFs* erläutert. Nun wird versucht, die Ergebnisse des ersten Experiments zu verbessern. Vor allem die Probleme im Bezug zur geringen Auslösung können mit dem *CRF* gelöst werden, da dieses auf der Eingabeauflösung arbeitet und die durch Upsampling erzeugten Bereiche weiter präzisieren kann. Zunächst werden die optimalen Hyperparameter mithilfe des Grid Search Verfahrens bestimmt (siehe Kapitel 5.4.1).
In Abbildung 5.23 sind die Ergebnisse des *CRF* Post-Processing im Vergleich zu den *DCNN* Ergebnissen aufgezeigt. Beim Xception-Net ist kaum ein Unterschied durch das Post-Processing auszumachen. Diese Beobachtung unterstützt die Entscheidung der DeepLab Autoren das *CRF* in der Version 3+ zu entfernen. Die Verbesserungen sind so gering, dass sie in den Fehlern der Eingabegröße keine Auswirkungen haben. Lediglich bei der ca. 8-fachen Hochskalierung wird eine geringe Verbesserung erkennbar.
Bei MobileNet ist der Einfluss des *CRF* weitaus stärker sichtbar. In der *RoI* bei 3000x4000 Pixeln beträgt die Verbesserung ca. ein Prozent.

| | 385x513 | 3000x4000 | 385x513 *RoI* | 3000x4000 *RoI* |
|---|---|---|---|---|
| Xception (c) | 1.14% | 1.22% | 2.63% | 6.05% |
| MobileNet (c) | 1.53% | 1.58% | 2.37% | 7.21% |
| Xception (c) + *CRF* | 1.14% | 1.20% | 2.64% | 5.89% |
| MobileNet (c) + *CRF* | 1.34% | 1.39% | 2.94% | 6.34% |

Abbildung 5.23: Erweiterung der Abbildung 5.21 um die Ergebnisse mit *CRF*

Abbildung 5.24 zeigt exemplarisch die Auswirkungen des *CRFs* auf die Segmentierung und zeigt auch, warum die Qualität ausschließlich beim MobileNet gesteigert wird. MobileNet erreicht oft nur ungenaue Segmentierungen im Randbereich und vor allem kleine Ecken wie in den oberen zwei Zeilen werden oft nicht abgebildet (vgl. auch Abbildung 5.18). Xception-Net ist an den Eigenschaften, die das *CRF* am meisten verbessert bereits sehr gut. Die Segmentierungen mit *CRF* erreichen mit beiden Netzen ähnlich gute Ergebnisse.

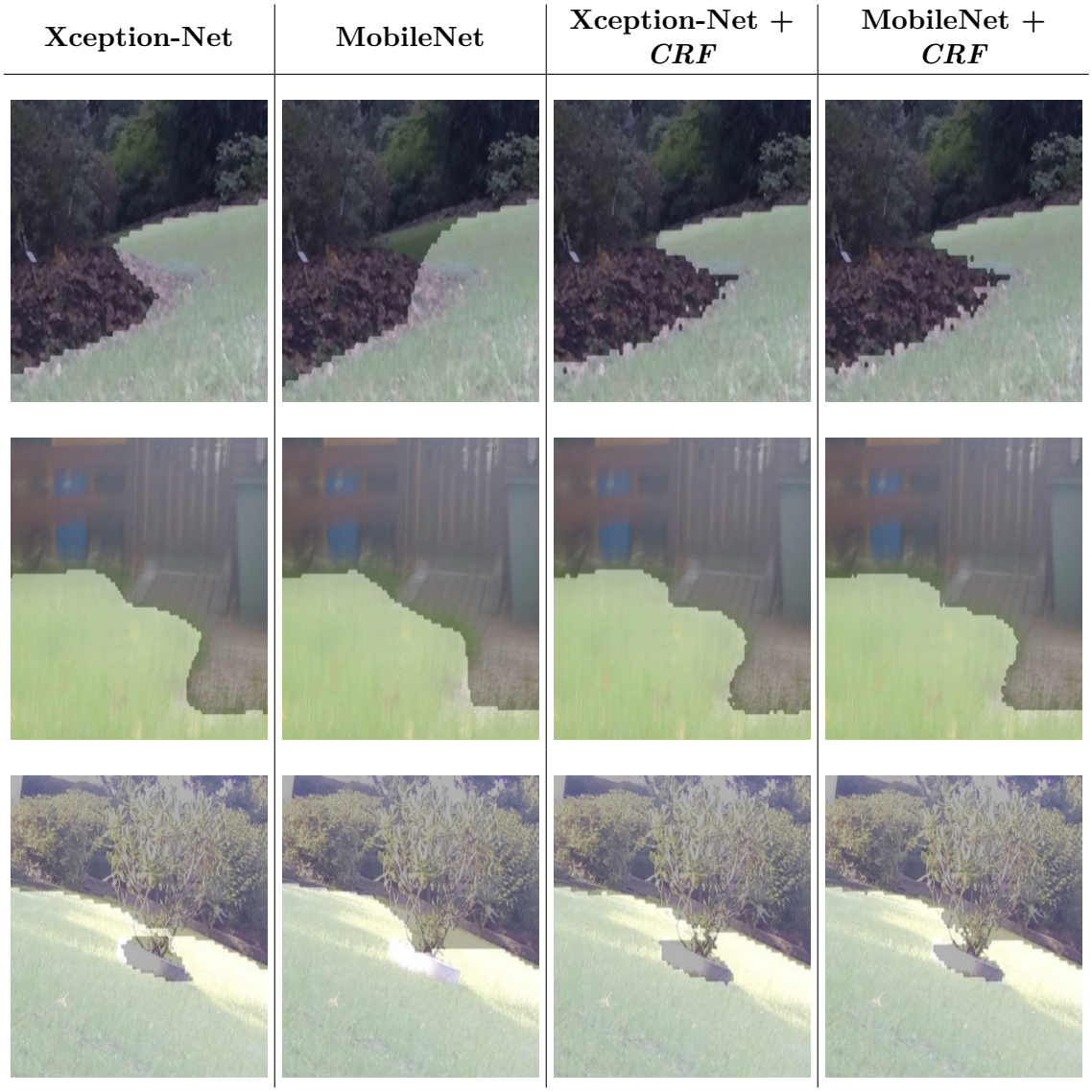



Abbildung 5.24: Auswirkung der *CRFs* auf die Segmentierung. Gerade beim MobileNet sind die Rasenkanten weitaus besser segmentiert. Auffällig sind auch die sehr kleinen nicht-mähbaren Flächen im Randbereich des oberen Bildes.

In Abbildung 5.23 ist beim MobileNet die Beobachtung zu machen, dass der *MIOU* bei einer Auflösung von 385x513 im Gesamtbild besser, in der *RoI* jedoch schlechter ist. In Abbildung 5.25 ist zu sehen, dass Artefakte der Segmentierung vom *CRF* verkleinert werden. Dies führt zu einer Verbesserung im Gesamtbild. Gleichzeitig wird die Rasenkante weitaus ungleichmäßiger segmentiert. Gerade bei weichen Kanten, wie in Abbildung 5.25 und 5.24 jeweils *oben*, ist ein *zackiger* Verlauf und ein *fleckiges* Muster zu sehen. Dieses Muster muss zwar nicht falsch sein, bei der Ground Truth Erstellung wird jedoch eine

eher gleichmäßige Kante erzeugt. So wird die Segmentierung in der *RoI* bei Bildern, in denen keine Ecken oder Objekte vorhanden sind, schlechter.

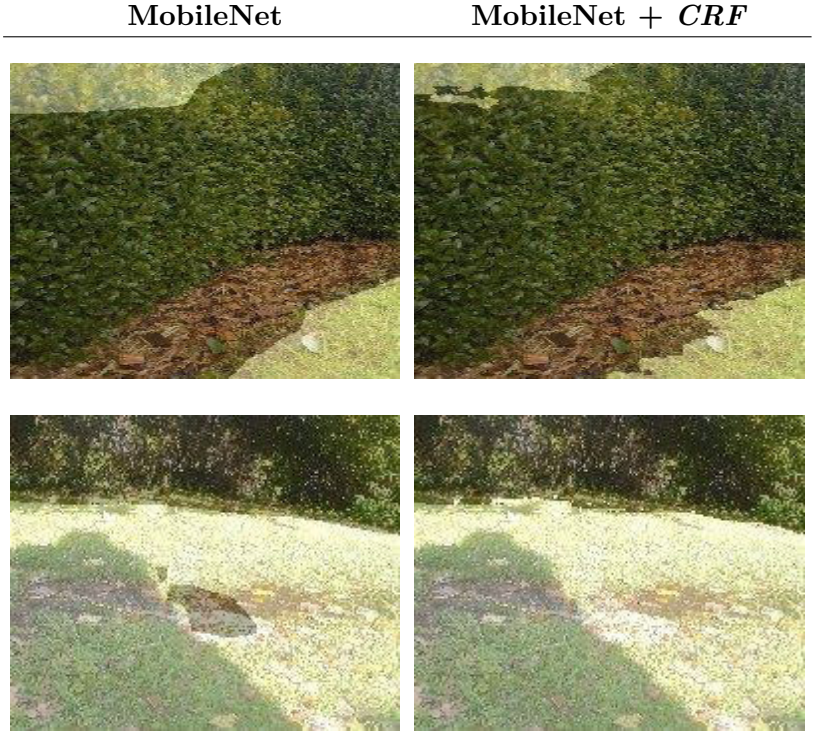



Abbildung 5.25: Das *CRF* verkleinert Bereiche, die vom MobileNet grob falsch segmentiert werden. Gleichzeitig ist die Rasenkante ungleichmäßiger.

In den *Baseline*-Experimenten wurde gezeigt, dass das unveränderte DeepLab 3+ bereits sehr gute Ergebnisse erzielen kann. Gerade im Nahbereich ist die Qualität nahezu optimal. Im Fernbereich sind jedoch noch Fehler auszumachen, die auf eine geringe Auflösung und große Hochskalierungen in diesem Bereich zurückzuführen sind.
Ein gewichteter Loss, der die Rasenkante stärker gewichtet als die großen Flächen, verbessert die Segmentierung im Randbereich, ohne die Flächen zu verschlechtern.
Gerade für MobileNet sorgt auch ein *CRF* als Post-Processing für große Verbesserungen, mit der eine ähnliche Qualität wie beim Xception-Net erreicht wird.

### 5.5.2 Experiment 2: Ausschnitte (V2)

Im zweiten Experiment wird versucht Schwächen des vorherigen auszubessern, die hauptsächlich mit einer zu geringen Auflösung im Fernbereich erklärbar sind. Hierfür wird die Skalierung der Eingabe verändert. Aufgrund der bekannten Perspektive ist es möglich eine Skalierung in Abhängigkeit der Distanz zur Kamera durchzuführen. Die Berechnung einer Tiefenkarte, die zu jedem Pixel unter der Annahme einer flachen Bodenebene berechnet, ist im Kapitel 4.4 beschrieben.
Im ersten Schritt gilt es, die Distanzen der Perspektive genauer zu analysieren. In Abbildung 5.26 sind exemplarisch die ersten Meter in gerader Linie vor der Kamera visualisiert. Bei dieser Betrachtung fällt auf, dass der erste Meter über die Hälfte des Bildes einnimmt und weitere Meter kaum mehr eindeutig auszumachen sind. Hier lässt sich bereits ein Problem der Eingabedaten aus dem ersten Experiment erkennen. Aus einem 3000x4000 Bild wird gleichmäßig auf 385x513 skaliert, obwohl der Informationsgehalt ungleichmäßig verteilt ist. Es werden also im Nahbereich (ersten 2 Meter) gleich viele Informationen verworfen, wie im Fernbereich. Da der große Schwachpunkt des ersten Experiments die geringe Auflösung gerade im Fernbereich ist, lässt sich durch eine intelligente Aufteilung, die mehr Informationen des Nahbereichs verwirft und dafür den Fernbereich höher aufgelöst behält, an genau dieser Stelle entgegenwirken.
Es wird nun versucht eine Aufteilung zu finden, die den Informationsverlust im Fernbereich verringert, ohne im Nahbereich zu viele Informationen zu verwerfen oder die Komplexität zu stark zu erhöhen.

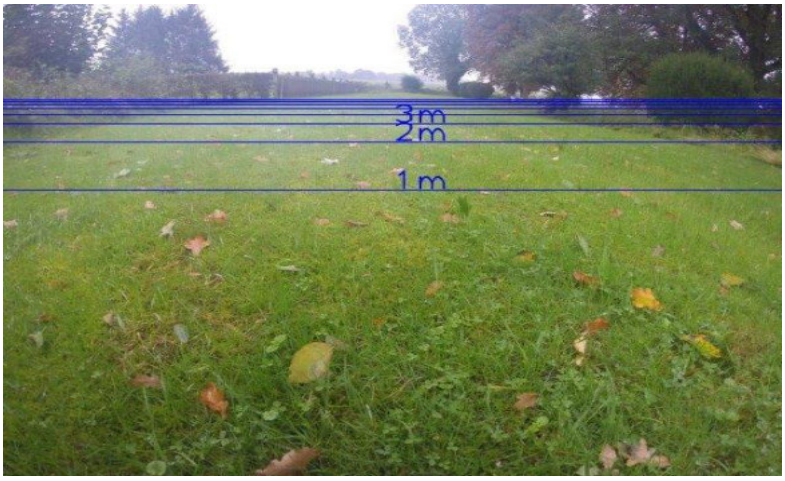



Abbildung 5.26: Visualisierung der Metergrenzen im Bild. Jede blaue Linie kennzeichnet je einen weiteren Meter (frontal vor der Kamera). Der erste Meter nimmt über die Hälfte des Bildes ein, ab fünf Meter sind die einzelnen Linien kaum mehr erkennbar.

Ein erster Versuch der Aufteilung stützt sich auf Quadrameterzahlen. In Abbildung 5.27 wird die Aufteilung eines Bildes nach einem Quadratmeter aufgezeigt. Da sich die Größe in Pixeln pro Quadratmeter im Fernbereich stark verringert, sorgt diese Aufteilung für sehr viele, kleine Teile. Auch eine Erhöhung der Quadratmeter auf 2 oder 4 bewirkt keine nennenswerte Verringerung der Ausschnitte.
Daraus folgen zwei direkte Probleme, die den Ansatz der gleichmäßigen Quadratmeterzahlen untragbar machen. Zum einen müssen alle Eingabebilder auf die gleiche Größe skaliert werden (siehe Kapitel 2.1.2). Dies sorgt für starke Verzerrungen der sehr kleinen Bereiche.
Zum anderen sorgt die Anzahl an Teilbildern dafür, dass die Rechenzeit zur Segmentierung eines kompletten Bildes zu hoch wird, da jeder Teil einzeln von DeepLab verarbeitet werden muss.

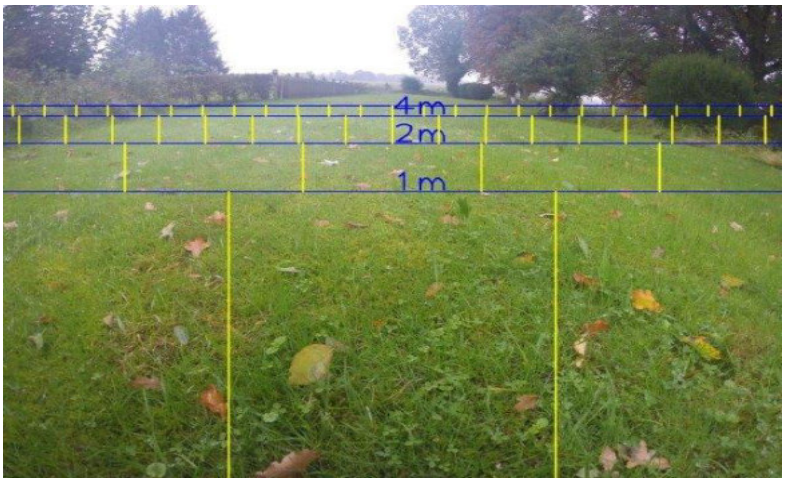



Abbildung 5.27: Aufteilung des Bildes nach Quadratmetern. Die vertikalen Linien sind so gewählt, dass die Distanz an der oberen horizontalen Linie $1m^2$ ergibt.

Eine zweite Aufteilung fokussiert die Unterteilung in Nah- und Fernbereich. Dabei wird keine gleichmäßige Quadratmeteranzahl angestrebt, sondern gleichförmige Seitenverhältnisse. In Abbildung 5.28 ist die Aufteilung eines Eingabebildes in drei Teilbilder zu sehen. Die Skalierung des unteren Teils ist deutlich verlustbehafteter als die der oberen Teile. Ebenfalls ist die Anzahl der Teile mit drei gering genug, um effizient verarbeitet werden zu können. Ein weiterer Nebeneffekt der Skalierung ist die leichte Korrektur der perspektivischen Verzerrung der oberen Bildbereiche. Der Stauchung durch die Weitwinkelaufnahme wird durch die Skalierung der y-Achse entgegengewirkt.


Abbildung 5.28: Aufteilung des Bildes in drei gleich skalierte Teilbilder. **Oben:** Skizzierung der Trennlinien, so dass die Seitenverhältnisse aller Teile ähnlich bleiben , um Verzerrung durch Skalierung zu verringern. **Unten:** Aufgeteilte und skalierte Eingabe.

Ein generelles Problem an der Segmentierung von Bildern mit *CNNs* ist die Betrachtung der Bildränder, da Convolution und Pooling stets Kontextinformationen benötigen, die an den Rändern nur ungenügend vorhanden sind. Um neben den ursprünglichen Bildrändern keine künstlichen Ränder an den Abschnittsgrenzen zu erzeugen, überlappen sich die Ausschnitte wenn möglich über 70 Pixel. Diese überlappenden Bereiche werden aus der fertigen Segmentierung verworfen, um nur Bereiche zu verwenden, die mit Kontextinformationen verarbeitet wurden.
Die genauen Bereiche und Skalierungen werden gesichert, um nach der Einzelsegmentierung ein Ursprungsbild in 3000x4000 Pixel zu erhalten. Dabei ist zu beachten, dass

die Aufteilung des Bildes am Horizont endet. Da die angenommene flache Ebene jedoch nie real gegeben ist, sind auch Rasenflächen über der Horizontlinie vorhanden. Bei der Bewertung des Experiments muss auf den Fakt geachtet werden, dass Bereiche über dem Horizont nicht beachtet werden.
Jeder Ausschnitt wird auf 222x296 skaliert. Die Skalierung wird so gewählt, dass die Seitenverhältnisse des Ursprungsbild erhalten werden und die Gesamtzahl an Pixeln, die zur Grundlage der Segmentierung eines Bildes dient, vergleichbar zum ersten Versuch bleibt. In *V1* wird mit der Auflösung eine Gesamtzahl von $385 \cdot 513 = 197505$ Pixeln verwendet. Bei der Zielauflösung wird eine Anzahl von $3 \cdot (222 \cdot 296) = 197136$ Pixel genutzt. So wird eine Vergleichbarkeit der beiden Versuche garantiert.
Das so erstellte Datenset wird als Eingabe für DeepLab verwendet. Da die Eingabegröße im Vergleich zum ersten Versuch stark verringert wird, müssen auch die Atrous Raten angepasst werden. Die ursprünglichen Atrous Raten wurden bereits im ersten Versuch aufgrund der niedrigen Auflösung halbiert, um schmale Objekte zu segmentieren. Ziel der Aufteilung ist es, die Auflösung dieser Objekte zu vergrößern. Aus diesem Grund werden sowohl die ursprünglichen Raten, als auch die neuen Raten des ersten Versuchs halbiert.

| | Lernrate | Iterationen | Atrous Raten | Decoder Skalierung | Output Skalierung |
|---|---|---|---|---|---|
| (a) | 0.0001 | 100000 | $[3, 6, 9]$ | 4 | 16 |
| (b) | 0.0001 | 100000 | $[2, 3, 5]$ | 4 | 16 |

Abbildung 5.29: Trainingssetup des zweiten Versuchs für beide Netze

Das Training der Ausschnitte mit gewichtetem Loss bietet ein zunächst irritierendes Bild. In Abbildung 5.30 sind sehr große Schwankungen im Loss zu beobachten. Dieses Verhalten ist erneut auf den gewichteten Loss zurückzuführen. Das bereits erwartete Schwanken wird durch die Ausschnitte jedoch noch weiter verstärkt, da in sehr vielen der unteren Ausschnitte keine Rasenkante vorhanden ist und somit die Gewichtung über alle Pixel des Ausschnitts 1 beträgt. So ist der Loss in diesen Ausschnitten geringer, während die Rasenkante in allen anderen mehr Bildfläche einnimmt und somit der hoch gewichtete Bereich größer wird.
Es fällt ebenfalls auf, dass die Losskurven nicht konvergieren, die Validierungskurven jedoch schon.

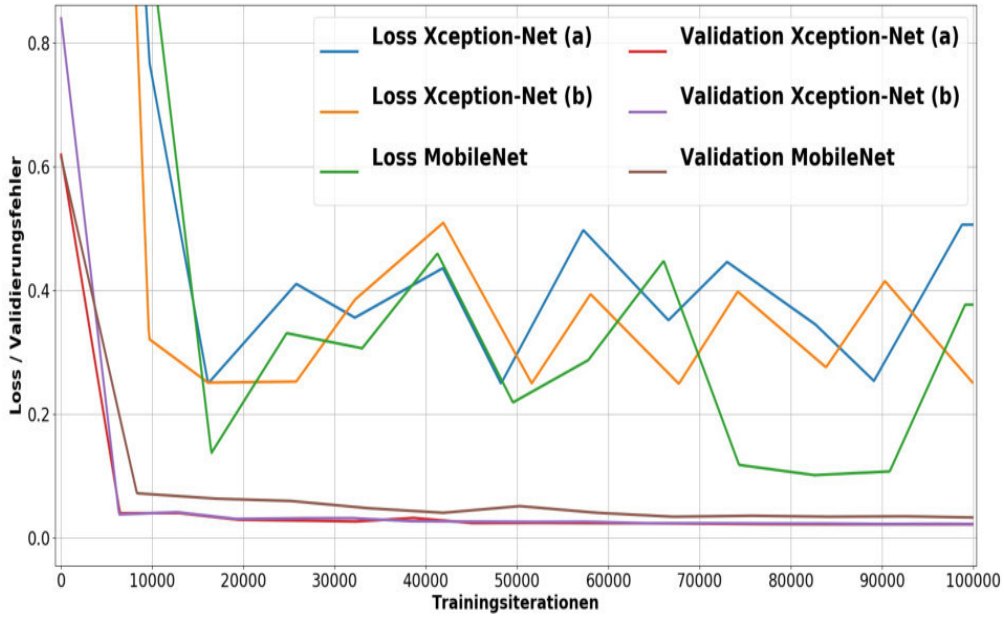
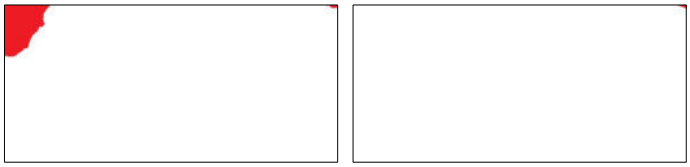



Abbildung 5.30: Trainingsverläufe des zweiten Experiments mit den Setups *(a)* und *(b)* aus Abbildung 5.29. Die Losskurven schwanken sehr stark und scheinen nicht zu konvergieren. Die Validierungskurven konvergieren hingegen am Ende des Trainings. Die Losskurve beschreibt den softmax cross-entropy Loss (Kapitel 2.1.1), der Validierungsfehler gemäß *MIOU* (Kapitel 2.3)

Auch für dieses Experiment wird ein *CRF* zum Post-Processing eingesetzt. Genau wie im ersten Versuch werden zunächst die Hyperparameter mit dem Grid Search Verfahren optimiert (siehe Kapitel 5.4.1).
Während der Testläufe fällt auf, dass im Großteil der Bilder der gesamte untere Ausschnitt aus einer Rasenfläche besteht. In diesem Bereich ist demnach lediglich eine der zwei Klassen (Rasen) vorhanden. Jedes dieser Bilder benötigt keine Verarbeitung des *CRFs*. Diese mögliche Vorselektion der Ausschnitte reduziert die Gesamtrechenzeit für die Verarbeitung eines Eingabebildes.

Die Ergebnisse des Experiments *V2* sind in Abbildung 5.32 dargestellt. Wie in der Einleitung zu diesem Experiment beschrieben, wird jedes Bild zur Bewertung an der Horizontlinie abgeschnitten. Im Vergleich zu *V1* fällt auf, dass der Fehler in den einzelnen Ausschnitten größer ist, die *RoI* jedoch um $0.5 - 1\%$ besser segmentiert wird. Die schlechten Werte in den einzelnen Ausschnitten ist auf die Berechnungsformel des *MIOU* zurückzuführen. Es reicht bereits eine kleine Fläche in einer der zwei Klassen, die nicht richtig segmentiert wird, um den *MIOU* extrem zu verringern. In Abbildung 5.31 ist ein unterer Abschnitt abgebildet, in dem der große Bereich der nicht-mähbar-Klasse falsch segmentiert wurde. Durch die geringe richtige Segmentierung in dieser Klasse ergibt sich ein *MIOU* von ca. 50%. In einem Ausschnitt kann es öfter vorkommen, dass ausschließlich

eine kleine Ecke eine andere Klasse hat als der Rest des Bildes, der nicht richtig erkannt wird und genau das beschriebene Verhalten erzeugt. In ganzen Bildern tritt dies nicht auf, da die Klassenverteilung innerhalb des ganzen Bildes ausgewogen ist.



(a) Ground Truth (b) *DCNN* Segmentierung

Abbildung 5.31: Ein Großteil des Ausschnitts ist richtig segmentiert. Da jedoch die nicht-mähbar-Klasse (rot) ausschließlich im kleinen Segment oben rechts (das hier kaum zu erkennen ist) im Bild erkannt wurde, ergibt der *MIOU* einen niedrigen Wert.

Bei den zwei Parametrisierungen von Xception-Net ist die Qualität von (b) leicht besser. Da es zwischen beiden keinen Unterschied in Komplexität oder Rechenzeit gibt, werden im Folgenden die geringeren Atrous Raten verwendet Bei der Betrachtung der Ergebnisse mit *CRF* fällt auf, eine Annäherung der Qualität sowohl der beiden Parametrisierungen von Xception-Net, als auch der von MobileNet, auftritt. So beträgt die Differenz zwischen Xception-Net und MobileNet nach dem Post-Processing, statt über einem Prozent vorher, nur noch ca 0.6%

| | 222x296 | 3000x4000 (Horizont) | 3000x4000 *RoI* (Horizont) |
|---|---|---|---|
| Xception (a) | 6.4% | 2.75% | 5.47% |
| Xception (b) | 6.38% | 2.67% | 5.37% |
| MobileNet | 7.71% | 3.78% | 6.58% |
| Xception (a) + *CRF* | 6.18% | 2.62% | 5.23% |
| Xception (b) + *CRF* | 6.09% | 2.56% | 5.23% |
| MobileNet + *CRF* | 6.87% | 3.40% | 5.89% |

Abbildung 5.32: Fehler bzgl. des *MIOU* und *RoI-MIOU* des zweiten Versuchs für beide Netze, jeweils in Eingabegröße 222x296 sowie skaliert auf 3000x4000

Der größte Vorteil, der in *V2* auszumachen ist, ist die genauere Segmentierung an allen Rasenkanten im oberen Bildabschnitt. Aufgrund der geringeren Skalierung in der Vorverarbeitung muss das Ergebnis geringer zurück skaliert werden. So ist das typische zu beobachtende Treppenmuster der Kanten viel feiner (siehe Abbildung 5.33). In *V1* ist zu sehen, dass stets ein Pixel exakt an der Kante richtig segmentiert wird, während mehrere Nachbarpixel durch die Skalierung falsch sind. Bei der Kante in *V2* ist die Anzahl dieser falschen Pixel stark verringert. Durch diese Besonderheit wird die Gesamtqualität aller Kanten erhöht.

(a) Zoom auf eine Kante aus *V1*

(b) Zoom auf die selbe Kante aus *V2*

Abbildung 5.33: Durch die höhere Auflösung der Ausschnitte und die dadurch resultierend geringere Skalierung wird die Kante feiner segmentiert. In beiden Bilder ist die Zoomstufe identisch.

Durch die genaueren Kanten werden auch Konturen von Objekten genauer abgebildet. In Abbildung 5.35 ist zu sehen, dass die typische Senke ohne spitze Winkel aus *V1* durch spitzere Winkel und engere Konturen ersetzt werden.
Noch stärker fällt die Verbesserung beim MobileNet auf. Wie in Abbildung 5.34 zu sehen, werden in *V2* auch spitz zulaufende Ecken (oberste Zeile) und grobe Fehler im Fernbereich (zweite Zeile) stark verbessert. Die unteren zwei Zeilen zeigen, dass die Segmentierung von Objektkanten stark verbessert wird. Die Auswirkung dieser Verbesserung fällt in MobileNet größer aus als beim Xception-Net, da die Segmentierung in *V1* von MobileNet an diesen Stellen deutlich schlechter ist.


Abbildung 5.34: Vergleich zwischen V1 und V2 an Objektgrenzen. Die Konturen werden in V2 deutlich besser abgebildet und spitzere Winkel an den Ecken erzeugt.

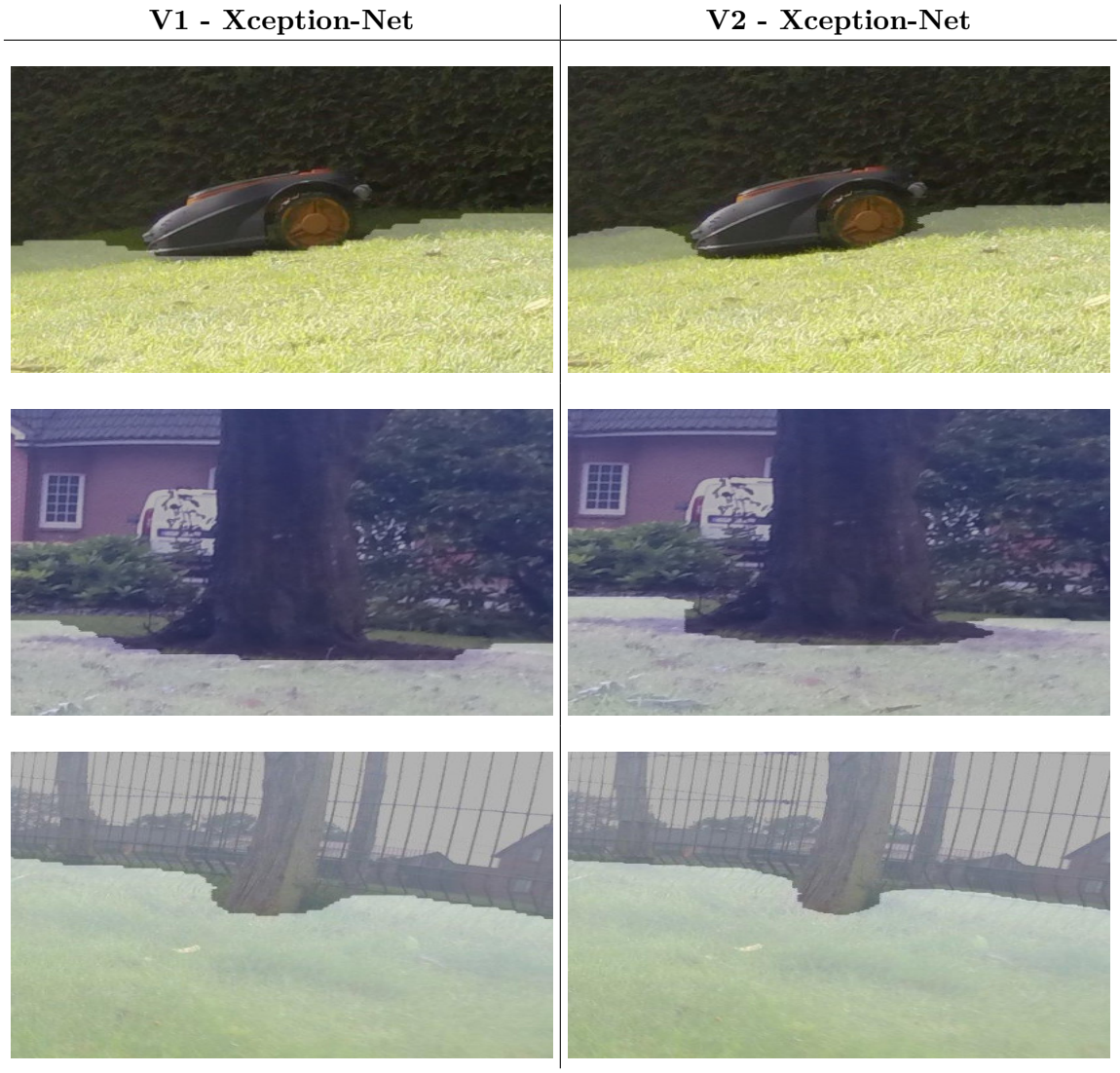



Abbildung 5.35: Vergleich zwischen V1 und V2 mit Xception-Net an Objektgrenzen. Aufgrund der höheren Auflösung im Fernbereich werden die Konturen in V2 deutlich besser abgebildet und spitzere Winkel an den Ecken erzeugt.

Ein bemerkenswertes Ergebnis ergibt sich beim Vergleich von *V2*-MobileNet mit *V1*-Xception-Net (Abbildung 5.36). In vielen Fällen werden große Fehler, wie die Grenzen der Terrasse oder der spitz zulaufende Bereich zwischen Beet und Weg, aus *V1*-MobileNet behoben. So erreicht MobileNet in vielen Fällen die Qualität der Xception-*Baseline*, bei weiterhin geringerer Komplexität. Lediglich die höhere Fehleranfälligkeit von MobileNet bleibt vorhanden.

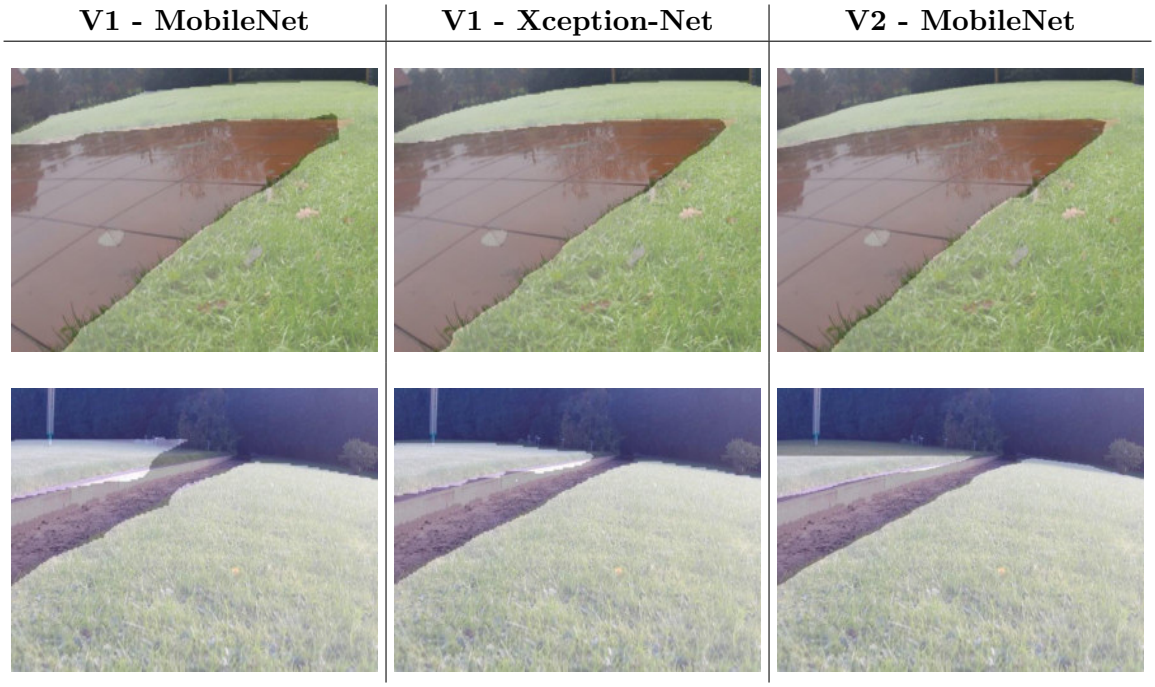



Abbildung 5.36: Beispielsegmentierung, die zeigt, dass grobe Fehler der MobileNet Segmentierung aus *V1* in *V2* verbessert werden. In vielen Fällen wird eine Qualität ähnlich zum Xception-Net aus *V1* erreicht.

Neben den aufgeführten Verbesserungen an Objektgrenzen führt die geringere Skalierung dazu, dass Bereiche, die in *V1* zu schmal sind und durch Kontextinformationen übergangen werden, nun weitaus größer sind. In Abbildung 5.37 ist die Auswirkung dieser Vergrößerung zu sehen. Die Graszunge der oberen Bilder wird in *V2* zu einem Großteil erkannt. Auch der Weg in den unteren Bildern ist ausschließlich in *V2* segmentiert. Der kleine Bereich, den *V1*-Xception-Net erkennt, kann auch als kleine Fehlsegmentierung interpretiert werden.
Zwar sind in allen Fällen die Segmentierungen nicht optimal, jedoch ist die grundlegende Detektion der Bereiche ein großer Gewinn für eine mögliche Robotersteuerung.
Mit dem *CRF* als Post-Processing kann die Qualität hier stark verbessert werden. So werden die groben Segmentierungen nun als deutliche Graszungen oder Wege abgebildet. In beiden Netzen ist eine solche Verbesserung zu beobachten. Beim Xception-Net sind die Ergebnisse nach dem Post-Processing weiterhin besser, was auf die bessere *DCNN* Segmentierung zurückzuführen ist.


Abbildung 5.37: Vergleich zwischen V1 und V2 bei der Segmentierung schmaler Bereiche im oberen Bildabschnitt. In *V1* deutet Xception-Net diese Bereiche maximal leicht an, in *V2* werden sie grob segmentiert und mit *CRF* feiner präzisiert. Auch MobileNet bildet mit *CRF* diese Bereiche ab.

Ein Grund für die hohe Qualitätssteigerung der MobileNet Segmentierung mit *CRF* Post-Processing ist die genauere Abbildung von Bäumen. In Abbildung 5.38 sind exemplarisch einige Fälle aufgeführt, in denen das *CRF* eine deutliche Verbesserung an den Kanten von Bäumen erzeugt. Gerade nach der Hochskalierung auf 3000x4000 Pixel hebt diese Anpassung die Gesamtqualität.


Abbildung 5.38: Das *CRF* Post-Processing bildet die Objektkanten der Bäume sehr exakt ab und sorgt somit für eine bessere Qualität in der *RoI*.

Als eine potentielle Fehlerquelle für *V2* wurden bereits im Vorfeld die Abschnittsgrenzen ausgemacht. In seltenen Fällen treten diese erwarteten Fehler auf (siehe Abbildung 5.39). Meistens treten diese Fehler in ähnlichen Umgebungen wie im Beispiel auf, wenn die Rasenfläche nicht durchgängig, sondern von braunen Erdflecken versetzt ist. Im Beispiel sind diese Flecken im unteren Abschnitt groß und auch in den überlappenden Pixeln nach oben vorhanden. Es scheint eine große Erdfläche oder eine Rasenkante vorhanden

zu sein.
Im oberen Abschnitt sind die Erdflecken kleiner und die großen Flächen aus den überlappenden Bereichen beeinflussen dies nicht so stark. Somit wird der gesamte obere Ausschnitt in diesem Bereich als mähbar segmentiert und die gerade Kante zwischen den Abschnitten entsteht.
Auch ein *CRF* Post-Processing kann diese Fehlerfälle nicht ausgleichen.

Abbildung 5.39: Beispiel für schlechte Segmentierungen an den Grenzen der Abschnitte.

Diese Probleme können jedoch in der Bewertung des Ansatzes weitestgehend ignoriert werden. Zum einen treten sie nur extrem selten auf, zum anderen können sie nur an den Abschnittsgrenzen auftreten. Das bedeutet entweder horizontal in ca. einem Meter Entfernung oder vertikal im Zentrum ca. einem Meter (vgl. Abbildung 5.28). So kann kein Hindernis in der direkten Nähe von diesen Fehlern betroffen sein, gleichzeitig ist sichergestellt, dass potentielle Fehlerbereiche während einer Fahrt auch in anderen Bereichen des Bildes erscheinen und somit ohne diese Fehlergefahr segmentiert werden können.
In *V2* wurden Informationen über die Perspektive der Bildaufnahme genutzt, um durch geschickte Bildaufteilung die Schwächen der *Baseline* auszugleichen. So konnte die Auflösung im Fernbereich erhöht und die Qualität in der *RoI* verbessert werden, ohne den Nahbereich zu verschlechtern. Die Informationsgrundlage in Form von der Anzahl der Pixel bleibt dabei annähernd gleich. Ein *CRF* als Post-Processing erhöht die Qualität in einem ähnlichen Maß wie auch bei der *Baseline*. Fehleranfällige Abschnittsübergänge verringern zwar die Qualität des Ansatzes, müssen aber ansonsten nicht weiter beachtet werden.

### 5.5.3 Experiment 3: Tiefeninformationen (V3)

Im dritten Experiment werden die Tiefeninformationen (Kapitel 4.4) zur Verbesserung des Ansatzes verwendet. Zunächst muss ein neues Modell entworfen werden, dass Tiefeninformationen als Eingabe verarbeiten kann. Im Kapitel 3.3 werden verschiedene Ansätze hierfür sowie ihre Problematik im Bezug zu dieser Arbeit vorgestellt.
Der hier vorgestellte Ansatz ist von den zuvor vorgestellten Ansätzen inspiriert und ähnelt diesen. Der Grundgedanke des neuen Ansatzes ist es, dass *DCNN* zu einem großen Teil zu spiegeln und in diesem gespiegelten Teil RGB-D Daten zu verarbeiten. Diese zwei Teile werden im folgenden *Tiefennetz* und *Grundnetz* genannt.
Nach jedem Block des Grundnetzes werden die jeweiligen Feature Maps in das Tiefennetz gegeben. So werden vor jedem Block des Tiefennetzes die Tiefen- und RGB-Informationen konkateniert. Eine Weiterleitung der Tiefeninformationen in das Grundnetz ist nicht vorgesehen. In Abbildung 5.40 ist der Ansatz grundlegend skizziert.

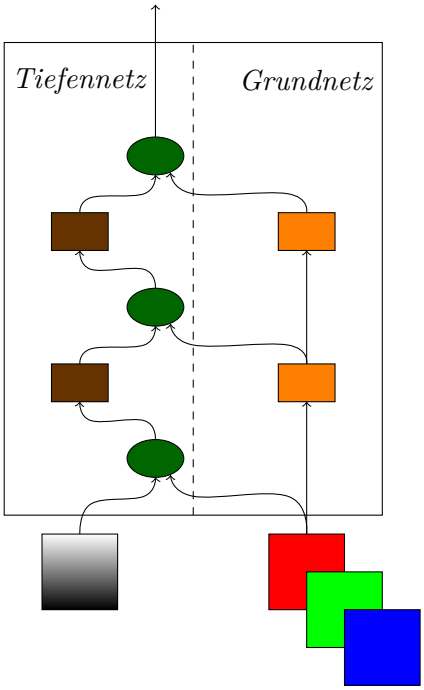
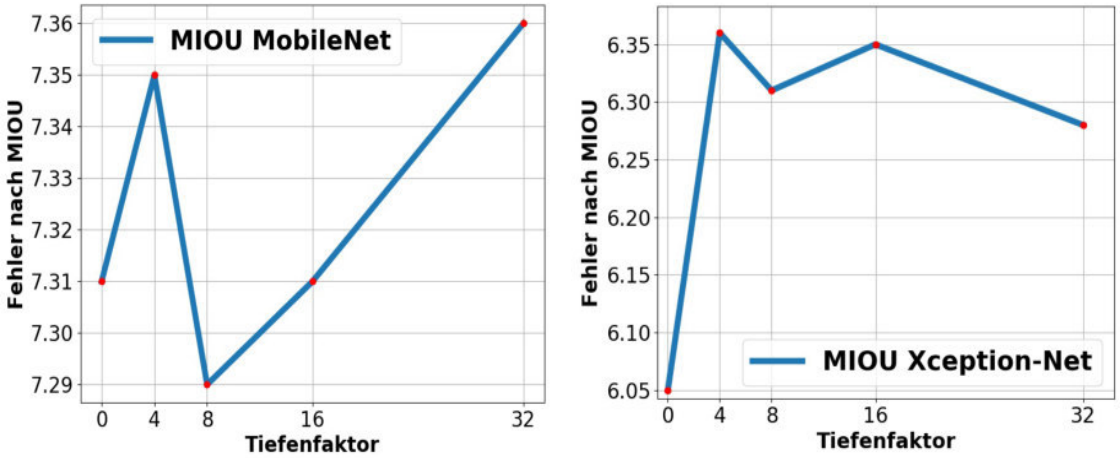



Abbildung 5.40: Skizze des Ansatzes mit Tiefeninformationen. Die elliptischen Knoten kennzeichnen Konkatenationen. Die rechteckigen Knoten stehen für Blöcke des *DCNN*. Die braunen haben dabei eine um einen Tiefenfaktor geringere Kanaltiefe als die orangenen.

Dieser Aufbau hat den Vorteil, dass die vortrainierten Gewichte des Grundnetzes weiterverwendet werden können, da dessen Struktur nicht verändert, sondern nur Zwischenausgaben extrahiert werden. Theoretisch wäre es zwar möglich die Struktur zu ändern und zusätzliche Gewichte neu zu initialisieren, praktisch bietet Tensorflow diese Möglichkeit aber nicht.
Das Tiefennetz wird nicht die identische Komplexität des Grundnetzes erhalten, da das Modell so zu groß wäre. Vielmehr wird ein Tiefenfaktor bestimmt, der die Anzahl der Kanäle in Abhängigkeit zum Grundnetz angibt. Je niedriger dieser Faktor gewählt wird, desto mehr Kanäle erhält das Tiefennetz. Dies resultiert in einem größeren Modell sowie einen höheren Einfluss der Tiefeninformationen auf die Gesamtsegmentierung. Zunächst muss ein möglichst guter Tiefenfaktor gewählt werden. Dieser wird experimentell bestimmt, indem die Qualität mit unterschiedlichen Faktoren verglichen wird. Die Trainingsparameter sind dabei die gleichen wie in Experiment *V1* (Abbildung 5.10 (c)). Als mögliche Tiefenfaktoren werden $4, 8, 16$ und 32 gewählt. Niedrigere Faktoren sind aufgrund des beschränkten *VRAMs* nicht möglich und höhere resultieren in einer Kanaltiefe von 1 in vielen Blöcken und somit auf eine verschwindend geringe Auswirkung. In Abbildung 5.41 ist die Qualität je nach Faktor aufgeführt. Der Wert für 0 beschreibt dabei das Ergebnis aus *V1* als Referenzwert. Aus der Grafik geht der Tiefenfaktor 8 als bester Wert für MobileNet und 32 für Xception-Net hervor. Zur Evaluierung der Tiefenfaktoren wird das gesamte Datenset, wie in *V1* genutzt.




(a) Tiefenfaktoren für MobileNet

(b) Tiefenfaktoren für Xception-Net

Abbildung 5.41: Vergleich der Tiefenfaktoren anhand des Fehlers im *RoI-MIOU* bei einer Auflösung von 3000x4000.

Im Trainingsverlauf fällt zuerst auf, dass das allgemeine Niveau der Losskurven höher ist als in den ersten Experimenten. Ebenfalls konvergiert das Training innerhalb der 100000 Iterationen nicht. Beide Beobachtungen sind auf die zufällig initialisierten Gewichte im Tiefennetz zurückzuführen, die eine längere Trainingszeit benötigen würden.
Der Validierungsfehler sinkt in beiden Netzen jedoch weiterhin beständig und lässt ein erfolgreiches Training erkennen.

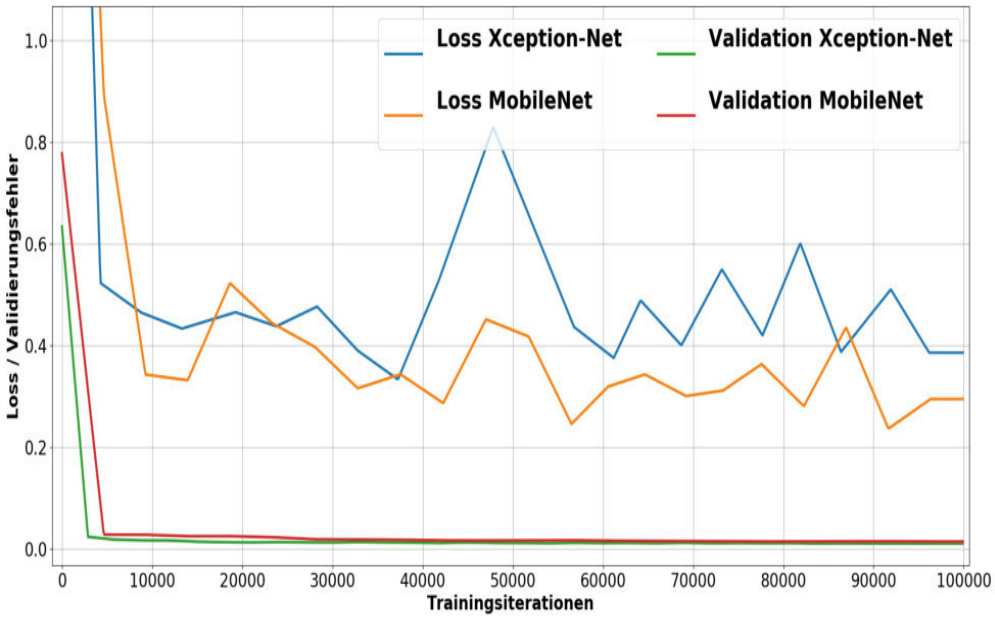



Abbildung 5.42: Der Loss verhält sich ähnlich wie in den vorherigen Experimenten. Allgemein ist der Loss aufgrund der neuen zufällig initialisierten Gewichte höher als in *V1*. Die Losskurve beschreibt den softmax cross-entropy Loss (Kapitel 2.1.1), der Validierungsfehler gemäß *MIOU* (Kapitel 2.3)

Bei der Evaluierung von *V3* fällt auf, dass eine leichte Verschlechterung erreicht wird. Besonders in der *RoI* wird der Unterschied deutlicher. In ganzen Bildern tritt beim MobileNet noch eine leichte Verbesserung auf, während beim Xception-Net die Verschlechterung in allen Bereichen auftritt. Auch in Abbildung 5.41 fällt bereits auf, dass keiner der Tiefenfaktoren die Qualität der ursprünglichen Segmentierung erreichen kann und die besten Ergebnisse auch beim Tiefenfaktor mit dem wenigsten Einfluss der Tiefeninformationen erzielt werden.
Ein Post-Processing mit einem *CRF* gleicht die Segmentierungen des Versuchs an die Ergebnisse mit *CRF* aus *V1* an. Da die Eingabebilder identisch zu *V1* sind, können die Hyperparameter übernommen werden.

| | 385x513 | 3000x4000 | 385x513 *RoI* | 3000x4000 *RoI* |
|---|---|---|---|---|
| Xception *V1* | 1.14% | 1.22% | 2.63% | 6.05% |
| MobileNet *V1* | 1.53% | 1.58% | 2.37% | 7.21% |
| Xception | 1.19% | 1.25% | 2.77% | 6.32% |
| MobileNet | 1.50% | 1.55% | 3.31% | 7.29% |
| Xception + *CRF* | 1.14% | 1.18% | 2.65% | 5.89% |
| MobileNet + *CRF* | 1.30% | 1.35% | 2.91% | 6.32% |

Abbildung 5.43: Fehler bzgl. des *MIOU* und *RoI-MIOU* des dritten Versuchs für beide Netze, jeweils in Eingabegröße 385x513 sowie skaliert auf 3000x4000. Verglichen wird *V1* mit der besten Parametrisierung, jedoch ohne *CRF*.

Auch bei einer genaueren Betrachtung der Bilder ist keine Eigenschaft auszumachen, die die Segmentierung von *V3* verbessert, vielmehr weist sie die selben Merkmale wie in *V1* auf. Eine mögliche Erklärung für die Verschlechterung kann in den neu initialisierten Gewichten des Tiefennetzes gefunden werden. Die Gewichte des Grundnetzes wurden über mehrere Datensets und viele hundert tausende Iterationen trainiert und erzielen deswegen sehr gute Ergebnisse. Die Mischung mit neuen Gewichten bewirkt einen Qualitätsverlust. Die Tatsache, dass die Verschlechterung beim Xception-Net größer ist, unterstützt diese Vermutung, da beim komplexeren Modell mehr neue Gewichte hinzu genommen werden. Des Weiteren könnte das Training des Tiefennetzes das Grundnetz negativ beeinflussen. Durch zufällige Gewichte ist zu erwarten, dass die ersten Trainingsiterationen schlechte Ergebnisse liefern und somit große Anpassungen der Gewichte erfordern. Diese Anpassungen werden auf das gesamte Netz angewendet, ändern also auch das Grundnetz stark, was zu einem schlechteren Training des Grundnetzes führen könnte.
Um diese Theorie zu prüfen, müsste ein Training in zwei Schritten durchgeführt werden. Im ersten wird das Grundnetz konstant gesetzt und ausschließlich Gewichte im Tiefennetz angepasst. Im zweiten Schritt werden dann beide Netze trainiert. Aufgrund der bereits sehr hohen Trainingszeit im Experiment *V3*, die im zweistufigen Training verdoppelt wird, kann diese Trainingsart in dieser Arbeit nicht durchgeführt werden.

### 5.5.4 Experiment 4: Tiefeninformationen & Ausschnitte (V4)

Im letzten Experiment wird *V3* mit *V2* kombiniert - Tiefeninformationen werden auf drei Bildausschnitte angewandt. Die besten Trainingsparameter aus *V2* (Abbildung 5.29 (b)) sowie die besten Tiefenfaktoren aus *V3* übernommen.

Der Trainingsverlauf (Abbildung 5.44) gibt das erwartete Bild ab. Ähnlich zu *V2* schwankt der Loss sehr stark und unvorhersehbar. Genau wie in *V3* ist das allgemeine Lossniveau aufgrund der zufällig initialisierten Gewichte und komplexerer Modelle höher.
Genau wie in den vorherigen Experimenten sinkt der Validierungsfehler jedoch beständig über den Trainingszeitraum.

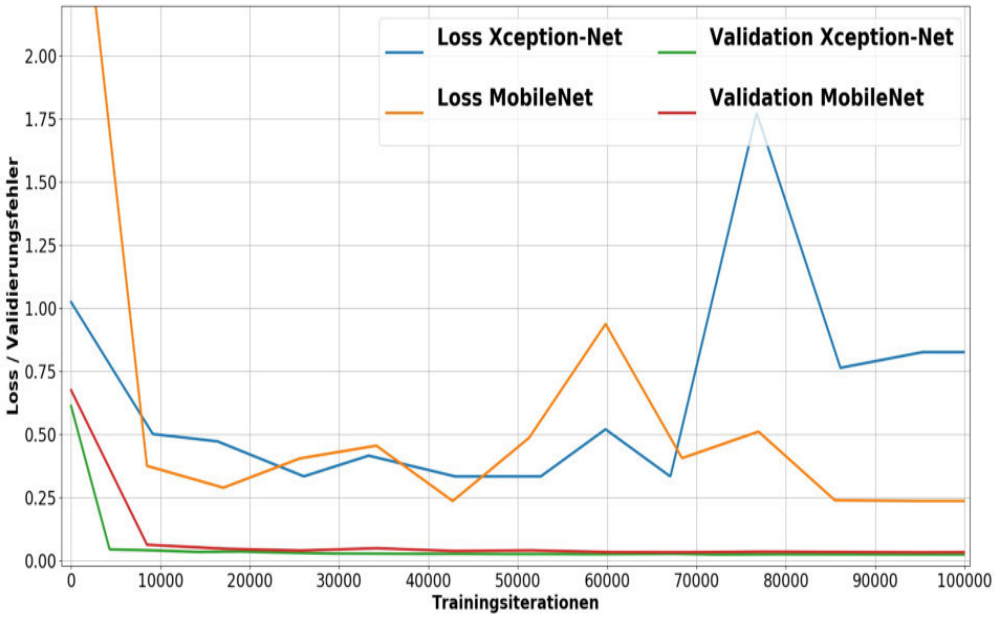



Abbildung 5.44: Der Loss verhält sich vergleichbar mit *V2* (Abbildung 5.30). Erneut konvergiert der Loss nicht. Aufgrund der neuen Gewichte ist wie in *V3* das allgemeine Lossniveau höher. Die Losskurve beschreibt den softmax cross-entropy Loss (Kapitel 2.1.1), der Validierungsfehler gemäß *MIOU* (Kapitel 2.3)

Die Ergebnisse von *V4* sind in Abbildung 5.45 aufgezeigt. Im Vergleich zu *V2* ist ähnlich wie bei *V3* eine Verschlechterung beim Xception-Net zu beobachten, die auf die gleiche Begründung zurückzuführen ist.
Auch der geringe *MIOU* in den Ausschnitten ist mit der gleichen Beobachtung, wie in *V2* zu erklären.
Das *CRF* Post-Processing erreicht eine ähnliche Qualität, wie auch in *V2*.

| | 222x296 | 3000x4000 (Horizont) | 3000x4000 *RoI* (Horizont) |
|---|---|---|---|
| Xception *V2* | 6.38% | 2.67% | 5.37% |
| MobileNet *V2* | 7.71% | 3.78% | 6.58% |
| Xception | 6.45% | 2.75% | 5.58% |
| MobileNet | 7.56% | 3.69% | 6.50% |
| Xception (b) + *CRF* | 6.10% | 2.55% | 5.25% |
| MobileNet + *CRF* | 6.73% | 3.29% | 5.82% |

Abbildung 5.45: Fehler bzgl. des *MIOU* und *RoI-MIOU* des vierten Versuchs für beide Netze, jeweils in Eingabegröße 222x296 sowie skaliert auf 3000x4000. Verglichen wird *V2* mit der besten Parametrisierung, jedoch ohne *CRF*.

Ähnlich wie auch in *V3* sind keine neuen Eigenschaften der Segmentierungen zu beobachten. Zwar ist beim MobileNet der *MIOU* leicht besser, jedoch kann nicht bestimmt werden, ob die Tiefeninformationen oder ausschließlich das komplexere Modell für diese Verbesserung sorgt. Um dies zu entscheiden, wird das Experiment mit MobileNet wiederholt, ohne jedoch die Tiefeninformationen in das Netz zu geben. Dieser Versuch erzielt nahezu identische Ergebnisse, wie mit Tiefeninformationen. Dies bestätigt die Vermutung, dass ausschließlich das komplexere Modell für die Qualitätssteigerung sorgt.

In den Experimenten *V3* und *V4* wurde ein neues Modell entworfen, dass Tiefeninformationen mit RGB-Informationen vereint. Es konnten jedoch keine entscheidenden Verbesserungen erreicht werden. Die Segmentierungen erlangen keine neuen Eigenschaften und unterscheiden sich kaum von den vorangegangenen Experimenten. Vielmehr konnte gezeigt werden, dass die Veränderungen der Ergebnisse bei MobileNet von den Tiefeninformationen unabhängig ist.
Nach dem *CRF* Post-Processing nähern sich die Ergebnisse weiter an *V1* und *V2* an.
Die Komplexität der Modelle (*FLOPS*) wird jedoch bei Xception-Net um einen Faktor von ca. 1.2 und bei MobileNet um ca. 1.6 erhöht.

## 5.6 Zusammenfassung

Im Folgenden werden alle durchgeführten Experimente zusammengefasst und gegenübergestellt. In Abbildung 5.46 werden repräsentative Ergebnisse jedes Experimentes aufgeführt. Für einen geeigneten Vergleich aller Experimente wird der Fehler bezüglich des *RoI-MIOU* in 3000x4000 Pixel Bildern bis zur Horizontlinie betrachtet. Es werden alle Experimente mit beiden *DCNNs* aufgeführt, wobei in jeder Spalte die verwendeten Konfigurationen angegeben werden.
Die *FLOPS* beziehen sich dabei ausschließlich auf die *DCNNs*, die *FPS* beziehen sich auf den kompletten Ansatz, ausgeführt auf dem Testsystem (Kapitel 2.5).
Im Vergleich aller Experimente ist zu beobachten, dass Xception-Net innerhalb eines Experimentes einen besseren *MIOU* erreicht, MobileNet jedoch stets eine weitaus geringere Komplexität benötigt und höhere *FPS* erreicht. Ebenso erreicht das Post-Processing mit einem *CRF* in allen Experimenten eine deutliche Qualitätssteigerung, die beim MobileNet stets höher ausfällt, jedoch erfolgt auch eine Verringerung der *FPS*.
Die Experimente *V3* und *V4* haben aufgrund des Tiefennetzes eine höhere Komplexität und erreichen somit auch generell geringere *FPS*. Die Qualität der beiden Experimente unterscheidet sich von den ersten Experimenten nur um ca. 0.2%. Beim Xception-Net ergibt sich eine Verschlechterung, beim MobileNet eine geringe Verbesserung durch das komplexere Modell. Die stark erhöhte Komplexität, ohne einen Qualitätsgewinn ist jedoch nicht vertretbar.
Die genauesten Ergebnisse werden für Xception-Net in *V2* und für MobileNet in *V4* erreicht, jedoch ist der weitaus effizientere Ansatz *V2* für MobileNet lediglich 0.07% schlechter. Generell ist zu beobachten, dass die Experimente mit Ausschnitten denen mit ganzen Bilder sowohl überlegen sind.
Die *Baseline* Experimente erreichen nicht die Qualität von *V2*. Beide Netze erreichen jeweils einen höheren *MIOU* und benötigen weniger *FLOPS*. Der Vergleich vom *V2*-MobileNet mit *V1*-Xception-Net gilt es an dieser Stelle herauszustellen. Beträgt die Differenz der Netze in *V1* noch ca. 1.2% kann *V2* diese auf 0.7% verringern, während gleichzeitig die Komplexität weiter sinkt.
Über alle Experimente fällt auf, dass lediglich MobileNet die geforderte Hürde der Echtzeitfähigkeit mit einem *FPS* überwindet. Xception-Net erzielt maximal 0.6 *FPS*, während MobileNet bis zu drei *FPS* erreicht. Es fällt auf, dass das *CRF* beim MobileNet einen größeren Einfluss hat, als beim Xception-Net. Diese Beobachtung ist darauf zurückzuführen, dass die Rechenzeit des *CRFs* ausschließlich von der Eingabegröße abhängt und somit innerhalb eines Experiments für beide Netze identisch ist. So fällt sie im Vergleich der höheren Rechenzeit vom Xception-Net weniger ins Gewicht.
Das beste Experiment dieser Arbeit in Beachtung der Qualität sowie der *FPS* ist *V2*. Die Segmentierung erzielt die besten Ergebnisse für Xception-Net und annähernd die besten für MobileNet, während MobileNet mit über zwei *FPS* schnell genug für die Echtzeitfähigkeit ist.
Für eine finale Überprüfung der Qualität des Ansatzes wird *V2* auf die Bilder der zwei

bisher unverwendete Gärten (siehe Kapitel 4.2) angewandt. Auch in diesen Gärten beträgt der Fehler der *RoI-MIOU* auf 3000x4000 Pixel nur 6.64% und liegt damit in der selben Qualität.

| *DCNN* | Atrous Rate | Aus-schnitte | Gewichteter Loss | *CRF* | Tiefen-netz | *RoI-MIOU* (Horizont) | *FLOPS* | *FPS* |
|---|---|---|---|---|---|---|---|---|
| **V1 - Baseline** | | | | | | | | |
| Xception | [3,6,9] | | | | | 5.93% | $8.187 \cdot 10^{10}$ | **0.6** |
| Xception | [3,6,9] | | ✓ | | | 5.84% | $8.187 \cdot 10^{10}$ | **0.6** |
| Xception | [3,6,9] | | ✓ | ✓ | | 5.70% | $8.187 \cdot 10^{10}$ | 0.5 |
| MobileNet | - | | | | | 7.97% | $4,155 \cdot 10^{9}$ | 2.8 |
| MobileNet | - | | ✓ | | | 7.07% | $4.155 \cdot 10^{9}$ | **3.0** |
| MobileNet | - | | ✓ | ✓ | | 6.14% | $4.155 \cdot 10^{9}$ | 1.7 |
| **V2 - Ausschnitte** | | | | | | | | |
| Xception | [2,3,5] | ✓ | ✓ | | | 5.37% | $\mathbf{7.995 \cdot 10^{10}}$ | 0.5 |
| Xception | [2,3,5] | ✓ | ✓ | ✓ | | **5.23%** | $\mathbf{7.999 \cdot 10^{10}}$ | 0.4 |
| MobileNet | - | ✓ | ✓ | | | 6.58% | $\mathbf{4.059 \cdot 10^{9}}$ | 2.3 |
| MobileNet | - | ✓ | ✓ | ✓ | | 5.89% | $\mathbf{4.059 \cdot 10^{9}}$ | 1.5 |
| **V3 - Tiefeninformationen** | | | | | | | | |
| Xception | [3,6,9] | | ✓ | | ✓ | 6.09% | $8.325 \cdot 10^{10}$ | 0.5 |
| Xception | [3,6,9] | | ✓ | ✓ | ✓ | 5.70% | $8.325 \cdot 10^{10}$ | 0.4 |
| MobileNet | - | | ✓ | | ✓ | 7.04% | $6.768 \cdot 10^{9}$ | 1.3 |
| MobileNet | - | | ✓ | ✓ | ✓ | 6.12% | $6.768 \cdot 10^{9}$ | 1.0 |
| **V4 - Tiefeninformationen & Ausschnitte** | | | | | | | | |
| Xception | [2,3,5] | ✓ | ✓ | | ✓ | 5.58% | $8.133 \cdot 10^{10}$ | 0.5 |
| Xception | [2,3,5] | ✓ | ✓ | ✓ | ✓ | 5.25% | $8.133 \cdot 10^{10}$ | 0.4 |
| MobileNet | - | ✓ | ✓ | | ✓ | 6.50% | $6.614 \cdot 10^{9}$ | 1.5 |
| MobileNet | - | ✓ | ✓ | ✓ | ✓ | **5.82%** | $6.614 \cdot 10^{9}$ | 1.1 |

Abbildung 5.46: Tabellarische Übersicht der Ergebnisse. Der Fehler bezüglich des *RoI-MIOU* wird in 3000x4000 Bildern bewertet, die an der Horizontlinie abgeschnitten sind. Die Anzahl der *FLOPS* bezieht sich nur auf das verwendete *DCNN*. Bei Ansätzen mit Ausschnitten werden entsprechend mit der Anzahl multipliziert. Die *FPS* Angabe bezieht sich auf die Inferenzzeit über den gesamten Ansatz auf dem Evaluierungssystem (Kapitel 2.5). Die jeweils besten Werte der Spalten für beide Netze sind fett markiert.

# Fazit und Ausblick

Die guten Ergebnisse *Baseline*-Experimente zeigen, dass ein vortrainiertes Modell mit einem vergleichsweise kurzen Training auf ein bis dahin unbekanntes Datenset angewendet werden kann. Auch die Verbesserung der Segmentierung mit einem *CRF*, wie es in der aktuellen Forschung oft verwendet wird, kann bestätigt werden.
Eine Gewichtung des Losses, die die Rasenkante stärker bewertet, zeigt eine Verbesserung der Segmentierungsergebnisse ohne die Komplexität zu erhöhen. Sowohl in dieser Arbeit als auch im Vorbild von Ronneberger et al. [RFB15] stehen Flächen, anstelle von Objekten (z.B. Autos, Personen, etc.) im Vordergrund. In solchen Problemen unterscheiden sich die *Objektkanten* wenig vom Rest der Fläche und die Gewichtung beeinträchtigt nicht die Segmentierung der Innenbereiche.
Das innerhalb der aktuellen Forschung oft dokumentierte Problem der schlechten Segmentierung aufgrund von zu geringer Auflösung kann auch in dieser Arbeit beobachtet werden.
Im Gegensatz zu vergleichbaren Arbeiten kann jedoch das Wissen über das Einsatzszenario und die Kameraperspektive eingesetzt werden um die Auflösung durch skalierte Ausschnitte zu verbessern. Experiment *V2* zeigt, dass die Anpassung der Skalierung die Segmentierungsqualität erhöht auch wenn die Eingabegröße stark verringert werden kann.
Die Experimente *V3* und *V4* weisen keine Verbesserung aufgrund der Tiefeninformationen auf. Die Beobachtungen zeigen, dass neu initialisierte Gewichte ein längeres Training benötigen. Der Informationsgewinn durch geschätzte Tiefeninformationen kann nicht bestätigt werden.
Generell kann in allen Experimenten beobachtet werden, dass das komplexere Modell Xception-Net bessere Ergebnisse erzielt, jedoch durch höhere Komplexität mehr Rechenzeit benötigt. Der Versuch von MobileNet in *V3* jedoch ohne Tiefeninformationen bestätigt diese Beobachtung.
Aus den aufgeführten Experimenten gilt es zu evaluieren, ob ein Ansatz gemäß der Rahmenbedingungen aus Kapitel 2.4 für einen realistischen Einsatz geeignet ist.
Aus der Zusammenfassung in Kapitel 5.6 geht *V2* als bester Ansatz hervor, da sowohl die Komplexität am niedrigsten, als auch der *MIOU* am höchsten ist.
Aufgrund der niedrigeren Komplexität und der damit erreichbaren *FPS* ist MobileNet für den Einsatz auf einer mobilen Plattform am besten geeignet. Die qualitativen Unterschiede zum Xception-Net können im Livebetrieb durch das Betrachten mehrerer Bilder zum Teil ausgeglichen werden.
MobileNet zeigte in den Experimenten eine höhere Gefahr von Fehlsegmentierungen und Anfälligkeit gegenüber extremer Lichteinstrahlung. Die Fehlsegmentierungen können durch eine übergeordnete Programmlogik, die Ergebnisse über mehrere Bilder vereint, erkannt werden. Dies wird vor allem durch die Geschwindigkeit ermöglicht, die das doppelte der geforderten *FPS* erreicht.
Extreme Sonneneinstrahlung tritt nur unter sehr beschränkt möglichen Bedingungen

auf, in denen die Neigungswinkel die Perspektive nach oben verändern. Da ein Rasenmähroboter während der Fahrt auf dem meist unebenen Untergrund stark wackelt, ist es kaum möglich, dass diese Bedingungen über mehrere Sekunden so auftreten. Nur in einem solchen Fall wäre eine Gefahr für die Robotersteuerung gegeben.
Die Segmentierung im Fernbereich ist exakt genug, um eine gute Wegplanung zu ermöglichen. Eine zentimetergenaue Steuerung entlang über 10m entfernter Objekte, wie die Xception-Net-Segmentierungen ermöglichen würden, ist nicht benötigt. Im Nahbereich, in dem diese Genauigkeit gefordert ist, erreicht MobileNet die benötigte Genauigkeit entlang der Objektgrenzen.
In dieser Arbeit wurde ein Deep Learning Ansatz entwickelt, der die aufgeworfenen Probleme zufriedenstellend lösen kann. Es wurde ein Datenset erstellt, mit dem der Ansatz erfolgreich trainiert werden konnte. So können zwei während des Trainings gänzlich unbekannte Gärten in der gleichen Qualität segmentiert werden. Es wurde damit gezeigt, dass ein generelles Modell erstellt wurde, dass unbekannte Gärten segmentiert und somit für das Szenario geeignet ist.
Sowohl Genauigkeit als auch Geschwindigkeit des Ansatzes *V2*-MobileNet erfüllen die geforderte Qualität, sodass ein Einsatz auf einem Realsystem denkbar ist. Der Ansatz wurde auf einem System ohne GPU auf Geschwindigkeit getestet. Auch hier werden die geforderten Werte erreicht. Die erreichten *FPS* sind noch so hoch, dass auch eine schwächere Hardware zur Evaluierung denkbar ist.
Als Eingabe benötigt der Ansatz lediglich RGB-Bilder. Für die Vorverarbeitung (Ausschnitte erzeugen) ist jedoch eine Information über die Perspektive im Livebetrieb nötig. Da eine Kamera in einem Rasenmähroboter fest verbaut wird, stellt auch dies keine Hürde dar.
Mögliche Folgearbeiten können vor allem mit stärkerer Hardware zum Training weitere Entwicklungen bringen. So konnte in dieser Arbeit keine Batchnormalisierung verwendet werden, die nach Aussage der DeepLab Autoren eine große Qualitätssteigerung erwirkt. Ebenfalls kann die Anzahl der Trainingsiterationen erhöht werden. Aus zeitlichen Gründen konnten in dieser Arbeit maximal 100000 Iterationen verwendet werden, was in vielen Versuchen noch nicht zu einem Konvergieren der Losskurve geführt hat. Längeres Training könnte die Qualität ebenfalls steigern.
Besonders die Experimente *V3* und *V4* könnten in einer Folgearbeit weiterentwickelt werden. Bereits in der Evaluierung der Experimente wird die Frage aufgeworfen, ob ein separates Training von Tiefen- und Grundnetz geeigneter ist.
Zudem kann auch versucht werden, das Xception-Net zu komprimieren (vgl. Goodfellow et al. [GBC16], Kapitel 12.1.4), um die Rechenzeit zu reduzieren und somit auch die genauere Segmentierung von Xception-Net auf einer mobilen Plattform zu nutzen.
Ebenfalls kann eine neue Arbeit anstatt theoretisch berechneter Tiefeninformationen echte RGB-D Daten verwenden, um den Informationsgehalt der Tiefeninformationen zu erhöhen. Mit einer Microsoft Kinect Kamera kann kostengünstig (mit einem Infrarot-Projektors und einer Infrarot-Kamera) ein Bild mit echten Tiefeninformationen erstellt werden.
Da die Ergebnisse die Anforderungen eines Realsystem erfüllen, kann basierend auf der

Segmentierung ein Robotersystem entwickelt werden. Zusammen mit einem Raddrehsensor kann der Arbeitsraum eigenständig erfasst werden. Aus der Kombination des Sensors, mit der Kamera kann ein *SLAM* implementiert und so ein unbekannter Garten erfasst werden. Als bekannte Ausgangsposition dient die Ladestation des Roboters.
Mit den Segmentierungen dieser Arbeit können mähbare Bereiche in der generierten Karte gekennzeichnet und so eine gezielte Wegplanung durchgeführt werden. In späteren Arbeitsphasen können aktuelle Segmentierungen die Karte weiter aktualisieren.
Ebenso wäre es denkbar die sehr schnellen Segmentierungen von MobileNet (über zwei *FPS*) für eine dynamische Hindernisvermeidung im Nahbereich zu verwenden und zeitgleich auf beispielsweise jedem zweiten Bild ein *CRF* für eine genauere Kartenaktualisierung zu nutzen.
So lassen sich die Ergebnisse dieser Arbeit in bekannte Algorithmen der Robotik integrieren, um in einem Realszenario zu bestehen.

# Glossar

**ASPP** Atrous Spacial Pyramid Pooling. Mehrere Atrous Convolutional Layer mit verschiedener Rate werden parallel angewendet. XI, 26, 44, 49, 53, 56, 58, 59, 61

**Baseline** Baseline Experimente definieren eine Basis, die für weitere Experimente verwendet wird. VIII, 53, 61, 67, 76, 80, 87, 89

**CNN** Convolutional Neural Network. Ein neuronales Netz mit mindestens einem Convolutional Layer. Weit verbreitetes Model in der Bildverarbeitung. VII, XI, 3–5, 7, 8, 19, 22–25, 28–30, 43, 70

**DCNN** Deep Convolutional Neural Network. Ein Convolutional Neural Network mit sehr vielen Layern, das im Themengebiet des Deep Learnings angesiedelt ist. 8, 12, 18, 43, 44, 47, 52, 62, 65, 73, 77, 81, 87, 88

**Decoder Output stride** Faktor der die Größe der Feature Maps angibt, welche als erste Eingabe für den Decoder verwendet wird.. 45

**Focal Length** Die Focal Length (Deutsch Brennweite) gibt die Distand zwischen dem Brennpunkt und der Kameralinse an. 37

**Machine Learning** Klasse von Verfahren aus dem Bereich der Künstlichen Intelligenz, welche Vorhersagen anhand von gelernten Beispielen treffen. Dabei erkennen diese Algorithmen Muster in den Beispielen und verallgemeinern diese. 4, 9

**Matplotlib** Python Bibliothek für die Erstellung und das Anzeigen von Graphen zur Datendarstellung. 17

**MobileNet ASPP** MobileNet mit Decoder und ASPP. 49, 55, 56, 60

**Multi-Modal Layer** Layerart, die Features aus unterschiedlichen Datenquellen vereint und fehlende Informationen einer Quelle ausgleichen kann. 29

**Numpy** Python Bibliothek für numerische Berechnungen und effiziente Verwendung von Matrizen als n-dimensionale Arrays. 16

**NVIDIA Cuda®** Toolkit von NVIDIA zur Entwicklung von Anwendungen für NVIDIA GPUs. `http://www.nvidia.de/object/cuda-parallel-computing-de.html`, Abrufdatum: 08.08.2018. 10, 16

**OpenCV** Python Bibliothek für Bildver- und Bildbearbeitung. 17

**Output stride** Faktor der die minimale Größe der Feature Maps angibt. Die Eingabegröße wird mit diesem Faktor dividiert. 45, 46

**Overfitting** *Auswendig lernen* des Trainingssets. Das Model performt gut auf Trainingsdaten, aber schlecht auf unbekannten Daten. 4, 9

**Principal Point** Realer optischer Mittelpunkt des Bildes. 37

**RoI-MIOU** Region of Interest - Mean Intersection over Union. Standardbewertungsmaß MIOU auf einen wichtigen Bereich. VIII, 49, 52, 53, 56, 73, 82, 84, 86–88

**SLAM** Simultaneous Localization and Mapping. Verfahren zur Kartenerstellung und Lokalisierung innerhalb von unbekannten Umgebungen.. 91

**Tensor** Mathematische Objekte zur Abbildung linearer Beziehungen zwischen Vektoren, Skalaren und anderer Tensoren.. 10

**Underfitting** Das Model ist nicht komplex genug, um das Problem geeignet abzubilden und performt allgemein schlecht. 4

**ZGS-Sensor** Zucchetti grass sensor. Sensorik, die Anhand von Schwellenwerten zu mähenden Rasen detektiert. 1

# Akronyme

**CRF** Conditional Random Field. VII, VIII, XI, XII, 4, 11, 12, 23–26, 50–52, 65–67, 72, 73, 77–80, 83–89, 91

**cuDNN** Cuda® Deep Neural Network library. 16

**FLOPS** Floating Point Operations. 13, 59, 86–88

**FPS** Frames per Second. 13, 14, 87–91

**GMM** Gaussian Micture Model. 34, 35

**MCMC** Markov-Chain-Monte-Carlo. 24

**MIOU** Mean Intersection over Union. XII, XIII, 13, 14, 21, 25, 26, 49, 53, 54, 56, 62, 66, 72, 73, 83–87, 89

**rCNN** Recurrent Convolutional Neural Network. VII, XI, 22, 23

**ReLU** Rectified Linear Unit. XIII, 6

**RoI** Region of Interest. XII, 34, 49, 50, 56, 58, 63, 65–67, 72, 73, 79, 80, 83, 84, 86

**VRAM** Video Random Access Memory. 46, 82

# Literatur


[Ber09] Bernini, F. Lawn-mower with sensor. Patent US 7613552B2 (US). Nov. 2009. URL: https://patents.google.com/patent/US7613552.

[BKC15] Badrinarayanan, V., Kendall, A. und Cipolla, R. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *arXiv preprint arXiv:1511.00561* (2015).

[Che+14] Chen, L. et al. Semantic Image Segmentation with Deep Convolutional Nets and Fully Connected CRFs. *CoRR* abs/1412.7062 (2014). arXiv: 1412.7062. URL: http://arxiv.org/abs/1412.7062.

[Che+16] Chen, L. et al. DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs. *CoRR* abs/1606.00915 (2016). arXiv: 1606.00915. URL: http://arxiv.org/abs/1606.00915.

[Che+17] Chen, L. et al. Rethinking Atrous Convolution for Semantic Image Segmentation. *CoRR* abs/1706.05587 (2017). arXiv: 1706.05587. URL: http://arxiv.org/abs/1706.05587.

[Che+18] Chen, L. et al. Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation. *CoRR* abs/1802.02611 (2018). arXiv: 1802.02611. URL: http://arxiv.org/abs/1802.02611.

[Cho16] Chollet, F. Xception: Deep Learning with Depthwise Separable Convolutions. *CoRR* abs/1610.02357 (2016). arXiv: 1610.02357. URL: http://arxiv.org/abs/1610.02357.

[Eit+15] Eitel, A. et al. Multimodal deep learning for robust RGB-D object recognition. In: *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*. IEEE. 2015, 681–687.

[Eve+15] Everingham, M. et al. The Pascal Visual Object Classes Challenge: A Retrospective. *International Journal of Computer Vision* 111, 1 (Jan. 2015), 98–136.

[GBC16] Goodfellow, I., Bengio, Y. und Courville, A. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[He+15] He, K. et al. Deep Residual Learning for Image Recognition. *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

[IS15] Ioffe, S. und Szegedy, C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *CoRR* abs/1502.03167 (2015).

arXiv: 1502.03167. URL: http://arxiv.org/abs/1502.03167.

[KK11] Krähenbühl, P. und Koltun, V. Efficient inference in fully connected crfs with gaussian edge potentials. In: *Advances in neural information processing systems*. 2011, 109–117.

[LBF14] Lai, K., Bo, L. und Fox, D. Unsupervised feature learning for 3d scene labeling. In: *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE. 2014, 3050–3057.

[LSD15] Long, J., Shelhamer, E. und Darrell, T. Fully convolutional networks for semantic segmentation. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, 3431–3440.

[LW] Liu, B. und Wen, Y. CNN and CRF for Semantic Image Segmentation ().

[NHH15] Noh, H., Hong, S. und Han, B. Learning deconvolution network for semantic segmentation. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, 1520–1528.

[PC14] Pinheiro, P. und Collobert, R. Recurrent convolutional neural networks for scene labeling. In: *International Conference on Machine Learning*. 2014, 82–90.

[RFB15] Ronneberger, O., Fischer, P. und Brox, T. U-Net: Convolutional Networks for Biomedical Image Segmentation. *CoRR* abs/1505.04597 (2015). arXiv: 1505.04597. URL: http://arxiv.org/abs/1505.04597.

[RKB04] Rother, C., Kolmogorov, V. und Blake, A. Grabcut: Interactive foreground extraction using iterated graph cuts. In: *ACM transactions on graphics (TOG)*. Bd. 23. 3. ACM. 2004, 309–314.

[San+18] Sandler, M. et al. Inverted Residuals and Linear Bottlenecks: Mobile Networks for Classification, Detection and Segmentation. In: *CVPR*. 2018.

[SLX15] Song, S., Lichtenberg, S. P. und Xiao, J. SUN RGB-D: A RGB-D scene understanding benchmark suite. In: *CVPR*. Bd. 5. 2015, 6.

[Soc+12] Socher, R. et al. Convolutional-recursive deep learning for 3d object classification. In: *Advances in Neural Information Processing Systems*. 2012, 656–664.

[SZ14] Simonyan, K. und Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. *CoRR* abs/1409.1556 (2014). arXiv: 1409.1556. URL: http://arxiv.org/abs/1409.1556.

[Wan+15] Wang, A. et al. Large-Margin Multi-Modal Deep Learning for RGB-D Object Recognition. *IEEE Transactions on Multimedia* 17, 11 (Nov. 2015), 1887–1898. ISSN: 1520-9210. DOI: 10.1109/TMM.2015.2476655.

[YK15] Yu, F. und Koltun, V. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122* (2015).

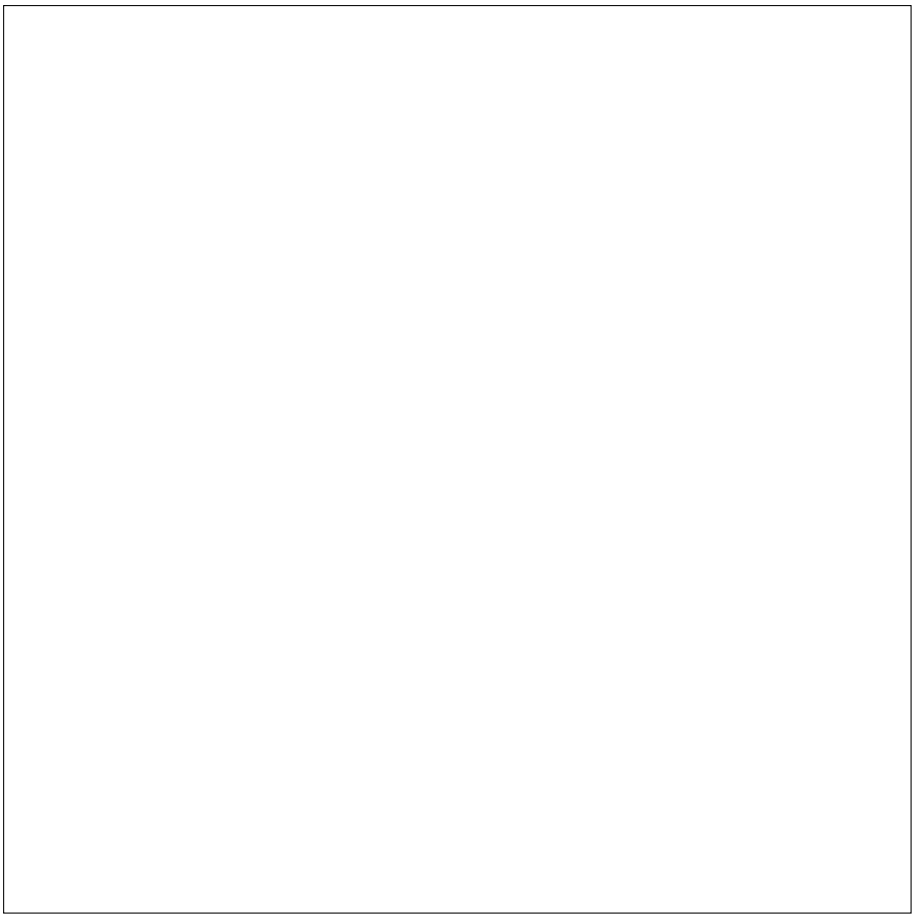
[Zhe+15] Zheng, S. et al. Conditional random fields as recurrent neural networks. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, 1529–1537.

[ZWL16] Zhu, H., Weibel, J.-B. und Lu, S. Discriminative multi-modal feature fusion for rgbd indoor scene recognition. In: *Proceedings of the IEEE Conference*

*on Computer Vision and Pattern Recognition*. 2016, 2969–2976.

# Anhang

# CD



Auf der CD ist neben diesem Dokument enthalten:

- TensorFlow-Models Repository mit Anpassungen für die Arbeit
- Skripte zur Durchführung und Evaluierung der Experimente
- Segmentierungen des am besten geeigneten Ansatzes