MONA KÖHLER

Deep-Learning-basierte semantische Segmentierung von Indoor-RGBD-Szenen für den Einsatz auf einem mobilen Roboter

MASTERARBEIT

Technische Universität Ilmenau
Fakultät für Informatik und Automatisierung
Fachgebiet Neuroinformatik und Kognitive Robotik

Deep-Learning-basierte semantische Segmentierung von Indoor-RGBD-Szenen für den Einsatz auf einem mobilen Roboter

Masterarbeit zur Erlangung des akademischen Grades Master of Science

Mona Köhler

Betreuer: M.Sc. Daniel Seichter

Verantwortlicher Hochschullehrer:
Prof. Dr. H.-M. Groß, FG Neuroinformatik und Kognitive Robotik

Die Masterarbeit wurde am 21.04.2020 bei der Fakultät für Informatik und Automatisierung der Technischen Universität Ilmenau eingereicht.

DOI: 10.22032/dbt.48299

URN: urn:nbn:de:gbv:ilm1-2021200033

TU Ilmenau | Universitätsbibliothek | ilmedia, 2021
http://www.tu-ilmenau.de/ilmedia

Danksagung

An erster Stelle möchte ich mich besonders bei meinem Betreuer Daniel Seichter bedanken. Daniel hat sich immer Zeit genommen, um mit mir über meine nicht enden wollende Rechercheergebnisse, die große Anzahl an möglichen Netzwerkarchitekturen, Implementierungsdetails und die hier vorliegende, etwas länger gewordene, Ausarbeitung zu sprechen.
Mein Dank gilt weiterhin dem Fachgebiet Neuroinformatik und Kognitive Robotik für die Bereitstellung der Rechenressourcen, ohne die diese Masterarbeit nicht möglich gewesen wäre. Ich danke außerdem meiner langjährigen Freundin Stefanie Wolf für die Gestaltung des Einbands dieser Masterarbeit. Darüber hinaus bedanke ich mich bei Samuel Klamandt und meinem Freund Frank Reißmann für das Gegenlesen und die hilfreichen Kommentare. An dieser Stelle möchte ich auch meinen Eltern danken – dafür, dass sie mir mein Studium ermöglicht und immer an mich geglaubt haben.

Kurzfassung

Eine pixelgenaue semantische Segmentierung bildet die Grundlage für ein umfassendes Szenenverständnis. Semantisches Wissen über die Struktur und den Aufbau von Indoor-Szenen kann mobilen Robotern bei verschiedenen Aufgaben nützlich sein. Unter Anderem kann dadurch die Lokalisierung, die Hindernisvermeidung, die gezielte Navigation zu semantischen Entitäten oder die Mensch-Maschine-Interaktion unterstützt werden. Durch den Einsatz von effizienten RGB-Verfahren konnten zuletzt bereits gute Segmentierungsergebnisse erzielt werden. Bei zusätzlicher Berücksichtigung von Tiefendaten kann die Segmentierungsleistung in der Regel noch weiter verbessert werden.

In dieser Masterarbeit werden daher Verfahren zur effizienten semantischen Segmentierung und zur RGBD-Segmentierung kombiniert. Auf Basis einer breiten Recherche zu beiden Themengebieten wird ein eigener, effizienter Deep-Learning-basierter RGBD-Segmentierungsansatz entwickelt. Mittels ausführlicher Experimente zu verschiedenen Bestandteilen der Netzwerkarchitektur wird gezeigt, wie die Segmentierungsleistung Schritt für Schritt erhöht werden kann. Neben der Segmentierungsleistung wird dabei stets auf eine geringe Inferenzzeit geachtet. Das beste, in dieser Masterarbeit entwickelte, Netzwerk erzielt auf dem einschlägigen Indoor-RGBD-Datensatz SUN RGB-D mit einer mean Intersection over Union (mIoU) von 47.62 vergleichbare Ergebnisse zum State of the Art. Dennoch ist die Verarbeitungsfrequenz mit 13.2 Frames pro Sekunde auf einem NVIDIA Jetson AGX Xavier deutlich höher und ermöglicht somit den Einsatz auf einem mobilen Roboter.

Abstract

Pixel accurate semantic segmentation lays the foundation for comprehensive scene understanding. Semantic knowledge about the structure and the setup of indoor scenes may support mobile robots in various tasks, such as localization, obstacle avoidance, targeted navigation to semantic entities, or human-machine interaction. Recently, precise segmentations have been achieved utilizing efficient RGB methods solely. However, incorporating depth images as well can further improve segmentation performance.
Therefore, in this master thesis, methods for both efficient semantic segmentation and RGBD segmentation are examined. Based on a broad literature research on both topics, a novel efficient deep learning-based RGBD segmentation approach is derived. With comprehensive experiments to various parts of the network architecture, the segmentation performance is improved step by step. Besides the segmentation performance, low inference time is of great importance for mobile applications. The best network achieves a comparable mean Intersection over Union (mIoU) of 47.62 to the state of the art on the relevant indoor RGBD segmentation dataset SUN RGB-D, while enabling a significantly higher frame rate of 13.2 frames per second on a NVIDA Jetson AGX Xavier and, thus, is well suited for usage on mobile robots.

Inhaltsverzeichnis

Abkürzungsverzeichnis

ASPP	Atrous Spatial Pyramid Pooling
BN	Batch Normalization
DAB	Depthwise Asymmetric Bottleneck
eASPP	efficient Atrous Spatial Pyramid Pooling
EESP	Extremely Efficient Spatial Pyramid
ESP	Efficient Spatial Pyramid
FLOPs	Floating Point Operations
FPS	Frames pro Sekunde
HHA	geozentrische Codierung der Tiefenbilder bestehend aus der horizontalen Disparität (engl. Horizontal Disparity), der Höhe über dem Boden (engl. Height above the Ground) und der Winkel der Oberflächennormalen zur angenommenen Gravitationsrichtung (engl. Angle).
IoU	Intersection over Union
IRB	Inverted Residual Bottleneck
LR-ASPP	Lite Reduced Atrous Spatial Pyramid Pooling
MEU	Mutual Embedding Upsample
mIoU	mean Intersection over Union
NBt-1D	Non-Bottleneck-1D
PFCU	Parallel Factorized Convolution Unit
PPM	Pyramid Pooling Modul
SE	Squeeze-and-Excitation
SSMA	Self-Supervised Model Adaptation
SS-NBt	Split Shuffle Non Bottleneck

Symbolverzeichnis

d	Dilationrate
k	Kernelgröße
r	Reduktionsfaktor der Kanalanzahl bei einer Convolution
C_{class}	Klassenanzahl des Datensatzes
C_{in}	Kanalanzahl des Input-Tensors
C_{out}	Kanalanzahl des Output-Tensors
G	Anzahl der Gruppen bei der Group Convolution
H_{in}	Höhe des Input-Tensors
H_{out}	Höhe des Output-Tensors
W_{in}	Breite des Input-Tensors
W_{out}	Breite des Output-Tensors

Legende

Die vorliegende Masterarbeit folgt einer einheitlichen Farbcodierung für Operationen und spezielle Feature Maps in Deep-Learning-Architekturen. In dieser Legende sollen alle verwendeten Operationen erklärt werden.

3×3 Convolution. dil steht für eine Dilated Convolution. Ist die Dilationrate d=1, so wird sie in der Abbildung weggelassen.

3×1 Factorized Convolution.

3×3 Depthwise Convolution.

3×1 Factorized Depthwise Convolution.

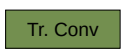
1×1 Convolution. Wird häufig zur Kanalreduktion oder -expansion verwendet. c gibt die Anzahl der Outputkanäle an.

1×1 Group Convolution.

Channel Split. Teilt den Input-Tensor entlang der Kanalachse in mehrere Gruppen auf, die danach separat verarbeitet werden (wie bei der Group Convolution).

Channel Shuffle. Wird für den Informationsaustausch zwischen den Kanälen bei Verwendung von Group Convolutions eingesetzt.

Hochskalieren. Zumeist bilineares Upsampling. Wird hauptsächlich im Decoder eingesetzt.



Transposed Convolution. Wird hauptsächlich im Decoder eingesetzt.

Global Average Pooling. Mittelt den Input-Tensor über die räumliche Dimension $H \times W$. Der Output-Tensor hat die Größe $C \times 1 \times 1$.

Max Pooling. Berechnet jeweils über die räumliche Dimension 2×2 das Maximum des Input-Tensors. Typischerweise entspricht der Stride der Pooling-Größe.

Encoder Feature Maps im Zusammenhang mit der effizienten Segmentierung.

Decoder Feature Maps im Zusammenhang mit der effizienten Segmentierung.

RGB-Features im Zusammenhang mit der RGBD-Segmentierung.

Tiefen-Features im Zusammenhang mit der RGBD-Segmentierung.

Features, die sowohl RGB- als auch Tiefen-Features enthalten im Zusammenhang mit der RGBD-Segmentierung. Zumeist Features des Decoders, nach der Fusionierung von RGB- und Tiefen-Features.

Addiert zwei oder mehrere Input-Tensoren elementweise. Die Input-Tensoren müssen in Höhe H, Breite W und Kanalanzahl C übereinstimmen.

Konkateniert zwei oder mehrere Input-Tensoren entlang der Kanalachse. Die Input-Tensoren müssen in Höhe H und Breite W übereinstimmen.

Wichtet den einen Input-Tensor mit einem anderen Input-Tensor (dem Wichtungs-Tensor). Stimmen alle Dimensionen (Höhe H, Breite W und Kanalanzahl C) der beiden Tensoren überein, so wird jedes Element separat gewichtet. Hat der Wichtungs-Tensor beispielsweise die Größe $C \times 1 \times 1$, entspricht das einer kanalweisen Wichtung des Input-Tensors. Jedes Element einer Feature Map wird somit mit dem gleichen Wert gewichtet.

Kapitel 1

Einleitung

1.1 Motivation

Eine pixelgenaue semantische Segmentierung bildet die Grundlage für ein umfassendes Szenenverständnis. Semantisches Wissen über die Struktur und den Aufbau von Indoor-Szenen kann mobilen Robotern bei verschiedenen Aufgaben nützlich sein. Unter Anderem kann dadurch die Lokalisierung, die Hindernisvermeidung [Hua et al., 2019], die gezielte Navigation zu semantischen Entitäten oder die Mensch-Maschine-Interaktion unterstützt werden.
Mithilfe immer aufwendiger und komplexer werdenden Deep-Learning-Architekturen [Chen et al., 2015], [Chen et al., 2018a], [Chen et al., 2017], [Chen et al., 2018b], [Zhao et al., 2017] konnte die Segmentierungsleistung stets verbessert werden. Diese Ansätze haben jedoch eine hohe Inferenzzeit und einen hohen Speicherbedarf. Für die Anwendung auf mobilen Plattformen wie den Robotern des Fachgebiets Neuroinformatik und Kognitive Robotik sind sie daher ungeeignet.
Aus diesem Grund wurden zuletzt vermehrt effiziente Segmentierungsarchitekturen entwickelt. Über verschiedene Optimierungen der Netzwerkarchitektur kann die Inferenzzeit und der Speicherbedarf um ein Vielfaches gesenkt werden. Dadurch ist es nun möglich, eine echtzeitfähige ($\geq$ 10 Frames pro Sekunde (FPS)) Segmentierung auf mobilen Plattformen – wie beispielsweise einer NVIDIA Jetson[1] – zu erhalten.

[1] Weiterführende Informationen zu verschiedenen NVIDIA Jetsons: `https://www.nvidia.com/de-de/autonomous-machines/embedded-systems/` – Abgerufen am 06.03.2020

Ein weiteres Forschungsfeld, neben der effizienten semantischen Segmentierung, ist die RGBD-Segmentierung. Neben dem RGB-Bild wird zusätzlich ein Tiefenbild verwendet, um die Segmentierung zu berechnen (siehe Abbildung 1.1). In der Regel kann die Segmentierung dadurch weiter verbessert werden.

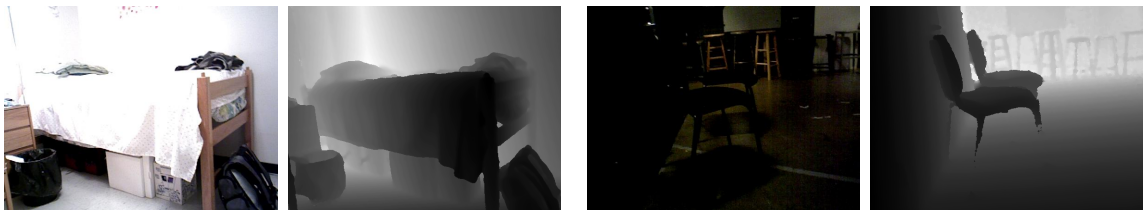
(a) RGB-Bild **(b)** Tiefenbild **(c)** Segmentierung

Abbildung 1.1: RGBD-Segmentierung
Das Ziel der RGBD-Segmentierung ist es mithilfe des RGB-Bilds (a) und des Tiefenbilds (b) die semantische Segmentierung der Eingabeszene (c) zu berechnen. In Abbildung 6.29 (Seite 129) sind die Farben der Segmentierung den einzelnen Klassen zugordnet. (Bilder aus dem SUN RGB-D Datensatz [Song et al., 2015])

Im Tiefenbild gibt jeder Pixelwert die Entfernung zur Kamera an. Das Tiefenbild enthält damit nützliche geometrische Informationen über die Szene. Dadurch ist es möglich die Segmentierungsleistung – vor allem an Objektkanten – zu verbessern. Im Indoor-Bereich kann das Tiefenbild auch bei der Klassenentscheidung helfen. Beispielsweise kann dadurch das Bild einer Person von einer tatsächlichen Person unterschieden werden.

Des Weiteren sind Tiefendaten robuster und unabhängig von der Belichtung. Bei einer Überbelichtung der Szene oder einer schlechten Beleuchtung, wie sie häufig im Indoor-Bereich vorkommt, kann Tiefe Informationen über die Bereiche liefern, die im RGB-Bild nicht mehr zu erkennen sind (siehe Abbildung 1.2).

Die komplementären Features von RGB und Tiefe können somit für eine verbesserte Segmentierungsleistung ausgenutzt werden.



überbelichtet unterbelichtet

Abbildung 1.2: Schwierige Beleuchtungssituationen
*In den über- und unterbelichteten RGB-Bildern sind einige Bereiche schwer erkennbar. Im zugehörigen Tiefenbild sind die Objektgrenzen deutlich besser zu erfassen. (Bilder aus dem SUN RGB-D Datensatz [*Song *et al., 2015])*

1.2 Zielstellung

Das Ziel dieser Masterarbeit ist es, effiziente semantische Segmentierung mit RGBD-Segmentierung zu kombinieren. Dafür soll ein effizienter Deep-Learning-basierter RGBD-Segmentierungsansatz für den Einsatz auf einem mobilen Roboter implementiert und trainiert werden. Hierfür erfolgt zunächst eine breite Literaturrecherche zu beiden Themengebieten. Aufbauend auf den Erkenntnissen des State of the Arts wird eine eigene Netzwerkarchitektur entwickelt. Für das Training und die Evaluierung des Netzwerks soll der Indoor-Segmentierungs-Datensatz SUN RGB-D verwendet werden. Über verschiedene Modifikationen der Netzwerkarchitektur soll ein Ansatz mit hoher Segmentierungsleistung, der dennoch die Echtzeitgrenze von 10 FPS erreicht, ermittelt werden. Für die Anwendung auf einem mobilen Roboter soll die, in PyTorch [Paszke et al., 2019] zu entwickelnde, Architektur mit TensorRT[2] für die Inferenz optimiert werden. Hierdurch kann eine höhere Verarbeitungsfrequenz erreicht werden, was es erleichtert oberhalb der definierten Echtzeitgrenze zu bleiben. Als Zielplattform soll ein NVIDIA Jetson AGX Xavier[3] zum Einsatz kommen.

[2] `https://developer.nvidia.com/tensorrt` – Abgerufen am 06.03.2020
[3] `https://www.nvidia.com/de-de/autonomous-machines/embedded-systems/jetson-agx-xavier/` – Abgerufen am 06.03.2020

1.3 Abgrenzung

Diese Masterarbeit fokussiert sich ausschließlich auf Deep-Learning-basierte Ansätze, da diese in den letzten Jahren die besten Ergebnisse erzielten. Ansätze, welche beispielsweise mit Conditional Random Fields [Sutton und McCallum, 2011] oder Superpixeln [Gupta et al., 2013] arbeiten, finden daher in dieser Masterarbeit keine Betrachtung. Darüber hinaus werden als Anwendungsgebiet lediglich Indoor-Bereiche in Betracht gezogen. Eine Segmentierung im Outdoor-Bereich unterliegt möglicherweise anderen Herausforderungen, wie unterschiedlichen Wetterbedingungen. Dennoch sind Ansätze für den Outdoor-Bereich im Allgemeinen auch auf den Indoor-Bereich übertragbar. Zudem sind Indoor-Szenen durch die vielen, kleinen, unterschiedlichen Objekte oftmals anspruchsvoller zu segmentieren als Outdoor-Szenen.
Des Weiteren erfolgt die Segmentierung lediglich auf Einzelbildern anstatt auf Videosequenzen. Mithilfe von Videosequenzen könnten zeitliche Informationen die semantische Segmentierung zusätzlich unterstützen. Allerdings existieren noch sehr wenige videobasierte Datensätze, vor allem für den Indoor-Bereich. Des Weiteren beschäftigen sich aktuelle Publikationen vorwiegend mit Einzelbildern.

1.4 Aufbau der Masterarbeit

Zunächst werden in Kapitel 2 wichtige Grundlagen zum Verständnis dieser Masterarbeit geklärt. Anschließend folgt in Kapitel 3 ein Überblick über den State of the Art zur effizienten semantischen Segmentierung. Danach wird in Kapitel 4 der State of the Art zur RGBD-Segmentierung vorgestellt. Mithilfe der Erkenntnisse aus diesen beiden Kapiteln wird daraufhin in Kapitel 5 der eigene Ansatz abgeleitet. Neben der Netzwerkarchitektur werden hier auch der verwendete Datensatz und weitere wesentliche Details zum Trainieren und Evaluieren des Netzwerks vorgestellt. Die entworfene Netzwerkarchitektur wird daraufhin in Kapitel 6 ausführlich experimentell untersucht und weiterentwickelt. Zuletzt wird in Kapitel 7 ein Fazit gezogen und ein Ausblick für weitere, nachfolgende Arbeiten gegeben.

Kapitel 2

Grundlagen

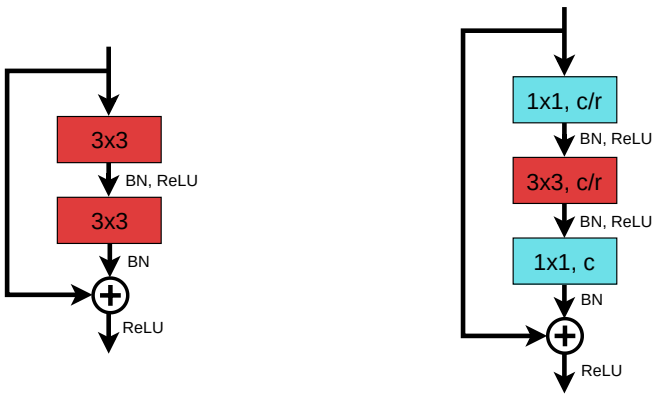
Dieses Kapitel soll die notwendigen Grundlagen zum Verständnis dieser Masterarbeit klären. Abschnitt 2.1 erklärt wesentliche Basisnetzwerke und -blöcke. In Abschnitt 2.2 werden verschiedene Arten der Convolution (deutsch: Faltung) dargestellt. Anschließend wird in Abschnitt 2.3 der grundlegende Aufbau moderner Segmentierungsarchitekturen erklärt. Zuletzt wird in Abschnitt 2.4 auf Grundlagen zu Tiefenbildern eingegangen.

2.1 Basisnetzwerke und Residual-Blöcke

In vielen Segmentierungsnetzwerken werden sogenannte *Basisnetzwerke* verwendet. Ein Basisnetzwerk bezeichnet dabei ein tiefes neuronales Netzwerk, das ursprünglich auf Bildklassifikation trainiert wurde. Über sogenannte *Heads* (deutsch: Aufsätze), die auf das Basisnetzwerk aufgesetzt werden, kann ein solches Basisnetzwerk für unterschiedlichste Anwendungen eingesetzt werden.
Bekannte Basisnetzwerke sind unter anderem das VGG16 und VGG19 der Visual Goemetry Group [Simonyan und Zisserman, 2014] oder das ResNet18, ResNet34, ResNet50, ResNet101 und ResNet152 von [He et al., 2016a]. Die Zahl am Ende des Netzwerknamens steht für die Anzahl an Convolutional Layer (deutsch: Faltungsschichten) im gesamten Basisnetzwerk. Für Anwendungen mit geringer, geforderter Inferenzzeit sind daher vor allem die Varianten mit wenigen Layern interessant.
Während das VGG16 eine lineare Netzwerkarchitektur aufweist, werden in den ResNets sogenannte *Residual-Blöcke* verwendet. Da viele Architekturen auf den Residual-Blöcken aufbauen, sollen diese hier kurz erklärt werden.

Residual-Blöcke

Der Kerngedanke der Residual-Blöcke besteht darin, dass es zusätzlich zum normalen Pfad durch die Convolutional Layer noch einen *Shortcut* (deutsch: Abkürzung) – das *Identity Mapping* (deutsch: Identitätsabbildung) – gibt. Über diesen Shortcut können die Feature Maps (deutsch: Merkmalskarten) einfach durch das Netzwerk weitergereicht werden. Am Ende des Blocks werden die Feature Maps des Convolutional-Pfads mit den Feature Maps des Shortcuts elementweise addiert. Das hat zur Folge, dass im Convolutional-Pfad nur noch die Abweichung zum Input – das *Residual* – gelernt werden muss. Eine Aneinanderreihung solcher Residual-Blöcke ermöglicht es sehr tiefe Netzwerke zu trainieren, wie in [He et al., 2016b] mit dem ResNet1001 gezeigt wurde. Mit den Shortcuts existiert ein einfacher Weg für den Gradienten, wodurch frühe Schichten im Netzwerk besser lernen können.
In [He et al., 2016a] werden zwei verschiedene Residual-Blöcke vorgestellt. Im ResNet18 und Resnet34 wird der sogenannte *Non-Bottleneck-Block* verwendet, während im ResNet50, ResNet101 und ResNet152 ein *Bottleneck-Block* eingesetzt wird. Die beiden Blöcke sind in Abbildung 2.1 dargestellt und sollen nun erklärt werden.



(a) ResNet-Non-Bottleneck-Block **(b)** ResNet-Bottleneck-Block

Abbildung 2.1: ResNet-Blöcke
Abgebildet sind die beiden ResNet-Blöcke. (a) Der ResNet-Non-Bottleneck-Block wird im ResNet18 und ResNet34 eingesetzt. (b) Der ResNet-Bottleneck-Block findet im ResNet50, ResNet101 und ResNet152 Anwendung. BN ist die Batch Normalization [Ioffe und Szegedy, 2015]. c/r steht für die Reduktion der Kanalanzahl c um den Faktor $\frac{1}{r}$. (Grafiken angelehnt an [He et al., 2016a])

ResNet-Non-Bottleneck-Block. Im ResNet-Non-Bottleneck-Block werden zwei Convolutional Layer mit einer Kernelgröße $k \times k$ von jeweils 3×3 verwendet. Zwischen den beiden Convolutional Layern werden die Feature Maps batch-weise normalisiert[1] und mit der Ausgabefunktion ReLU versehen. Auch nach der zweiten Convolution wird Batch Normalization verwendet. Die Ausgabefunktion wird jedoch erst nach der elementweisen Addition mit dem Shortcut angewandt. Dadurch gehen negative Aktivierungen im Convolutional Pfad nicht verloren.[2]

ResNet-Bottleneck-Block. Im ResNet-Bottleneck-Block wird zuerst die Anzahl der Feature Maps mittels einer 1×1 Convolution um den Faktor $\frac{1}{r}$ reduziert. Dadurch findet eine Projektion in einen Unterraum statt und das Netzwerk lernt sich auf die wesentlichen Features zu konzentrieren. Das hat zur Folge, dass die daran anschließende 3×3 Convolution mit weniger Feature Maps berechnet werden muss und somit effizienter ist. Zuletzt wird die Kanalanzahl mittels einer 1×1 Convolution wieder um den Faktor r expandiert. Dadurch haben der Output-Tensor und der Input-Tensor des Blocks die gleiche Kanalanzahl. Dies ist für die elementweise Addition beider Tensoren notwendig. Batch Normalization und Ausgabefunktionen werden ähnlich wie im Non-Bottleneck-Block verwendet.[3]

Ein weiteres Basisnetzwerk ist das DenseNet von [Huang et al., 2017]. Das DenseNet ist eine Erweiterung des ResNets. Der wesentliche Unterschied zum ResNet-Block besteht in der Konkatenation am Ende eines Blocks anstelle der elementweisen Addition. Weitere Informationen zum DenseNet sind im Anhang A.1.1 zu finden.

2.2 Verschiedene Arten der Convolution

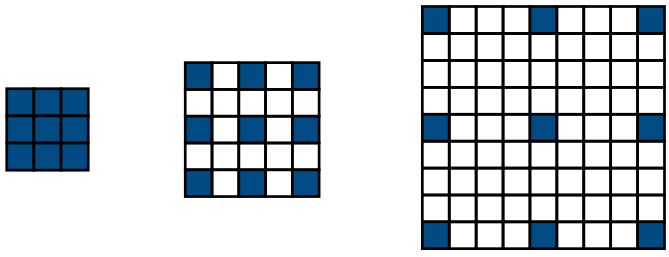
Neben der Standard-Convolution gibt es noch weitere, andere Arten der Convolution, die unterschiedliche Zwecke erfüllen. Mithilfe der Dilated Convolution (siehe Abschnitt 2.2.1) soll das rezeptive Feld[4] vergrößert werden, um mehr Kontext-

[1] Batch Normalization wurde in [Ioffe und Szegedy, 2015] eingeführt.
[2] Weitere Non-Bottleneck-Blöcke werden in Abschnitt 3.2.1 erklärt.
[3] Weitere Bottleneck-Blöcke werden in Abschnitt 3.2.2 erklärt.
[4] Das rezeptive Feld ist die Größe des Bereichs im Eingangsbild, welches einen Einfluss auf die Aktivierung des aktuellen Layers hat. Vor allem für die semantische Segmentierung ist es wichtig, dass das rezeptive Feld große Bereiche des Eingabebilds umfasst.

informationen miteinbeziehen zu können. Dahingegen kann mit der Factorized Convolution (Abschnitt 2.2.2), der Depthwise Separable Convolution (Abschnitt 2.2.3) und der Group Convolution (Abschnitt 2.2.4) der Rechenaufwand reduziert werden. Nachfolgend sollen diese verschiedene Arten der Convolution erklärt werden.

2.2.1 Dilated Convolution

Die *Dilated Convolution* (manchmal auch *Atrous Convolution* genannt) wurde ursprünglich in [Yu und Koltun, 2016] eingeführt. Dabei wird die Standard Convolution um einen zusätzlichen Parameter – der Dilationrate d – erweitert. Die Dilationrate beeinflusst die Streckung des Kernels. Bei einer Dilationrate von d=2 wird nur auf jedem zweiten Pixel ein Kernelelement aufgesetzt, bei einer Dilationrate von d=4 nur auf jedem vierten Pixel (siehe Abbildung 2.2). Das heißt es werden $d-1$ Nullen zwischen den Kernelelementen eingefügt. Die Standard Convolution wird als Spezialfall definiert mit einer Dilationrate von d=1.



(a) d=1 **(b)** d=2 **(c)** d=4

Abbildung 2.2: Dilated Convolution
3×3 *Convolution-Kernel mit unterschiedlichen Dilationraten d. Die Standard Convolution (a) ist als Spezialfall der Dilated Convolution anzusehen.*

Bei einer Kernelgröße k wächst das rezeptive Feld mit jeder Convolution um $(k-1) \cdot d$. Im Vergleich zur Standard Convolution kann somit – mit zunehmender Dilationrate – ein exponentielles Wachstum des rezeptiven Felds erreicht werden. Dadurch ist es möglich mehr Kontextinformationen miteinzubeziehen, was vor allem für die semantische Segmentierung wichtig ist.
Ein größeres rezeptives Feld könnte auch durch größere Kernels oder mehrere Convolutions erreicht werden. Dadurch würde allerdings der Rechenaufwand und die Parameteranzahl steigen. Bei der Dilated Convolutions bleibt hingegen die Anzahl der Rechenoperationen und Parameter, im Vergleich zur Standard Convolution, konstant.

2.2.2 Factorized Convolution

Bei der *Factorized Convolution* (manchmal auch als *Asymmetric Convolution* bezeichnet) wird eine $k \times k$ Convolution in eine $k \times 1$ und eine $1 \times k$ Convolution aufgeteilt. Dadurch reduziert sich der Rechenaufwand von k^2 auf $2k$.

Wie in Abbildung 2.3 zu erkennen ist, ist der Sobel-Operator separierbar und liefert durch eine Faktorisierung der Convolution das gleiche Ergebnis.

$$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & -1 \end{bmatrix}$$

Abbildung 2.3: Factorized Convolution
Der Sobel-Operator ist separierbar. Anstelle eines 3×3 *Filters kann auch ein* 3×1 *und ein* 1×3 *Filter verwendet werden.*

Da nicht alle Filter separierbar sind, ist das Netzwerk eingeschränkt in den zu erlernenden Filtern. Dadurch sinkt auch die Repräsentationskraft des Netzwerks. Häufig wird daher zwischen der 3×1 und der 1×3 Convolution eine nichtlineare Ausgabefunktion eingesetzt, um die Repräsentationskraft wieder zu erhöhen.

Factorized Convolutions wurden für die semantische Segmentierung erstmals im ENet von [Paszke et al., 2016] eingesetzt.

2.2.3 Depthwise Separable Convolution

Depthwise Separable Convolutions wurden bereits in [Sifre, 2014] verwendet und durch die Xception-Architektur von [Chollet, 2017] bekannt. Um die Funktionsweise der Depthwise Separable Convolution besser zu verstehen, soll zunächst ein Rückblick auf die Standard Convolution gegeben werden.

Rückblick: Standard Convolution

Bei der Standard Convolution wird für jede der Input Feature Maps C_{in} ein Filter benötigt, um insgesamt eine Output Feature Map zu erzeugen. Dies wird hier als ein Filter-Set bezeichnet. Da der Output-Tensor meistens aus mehreren Feature Maps C_{out} besteht, werden mehrere solcher Filter-Sets benötigt. Die Standard Convolution ist in Abbildung 2.4 dargestellt. Die Anzahl der Rechenoperationen beträgt hier $H_{out} \cdot W_{out} \cdot k_1 \cdot k_2 \cdot C_{in} \cdot C_{out}$. Dabei ist $H_{out} \times W_{out}$ die Output-Größe der Feature Maps. Bei Verwendung von same-Padding[5] und einem Stride von 1 ist diese gleich der Input-Größe $H_{in} \times W_{in}$. k bezeichnet die Kernelgröße. Prinzipiell ist $k = k_1 = k_2$. Wie bei der Factorized Convolution können diese aber auch verschieden sein.

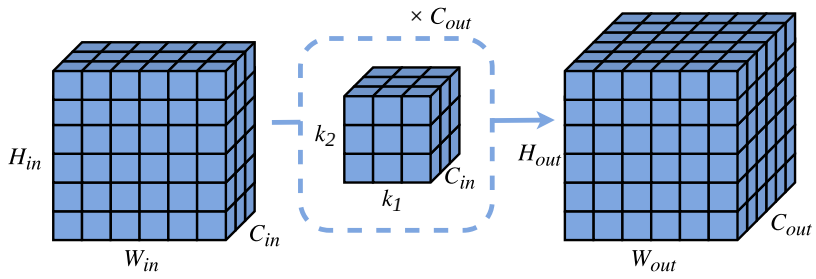


Abbildung 2.4: Standard Convolution
Um eine Output Feature Map zu erzeugen, wird für jede Input Feature Map ein Filter benötigt (= ein Filter-Set). Die Anzahl der Filter-Sets ergibt sich durch die Anzahl der gewünschten Output Feature Maps. (Grafik angelehnt an [Bai, 2019])

Depthwise Separable Convolution

Die Depthwise Separable Convolution besteht – ähnlich wie die Factorized Convolution – aus zwei getrennten Operationen. Dabei ist die erste Operation die *Depthwise Convolution* und die Zweite eine *Pointwise Convolution*. Im Gegensatz zur Factorized Convolution findet hier keine räumliche Separierung, sondern eine Separierung entlang der Kanalachse statt.

[5] Same-Padding bewirkt, dass die Output-Größe gleich der Input-Größe ist. Bei einer ungeraden Kernelgröße $k \times k$ werden an jeder Seite des Input-Tensors $\frac{k-1}{2}$ Nullen angefügt.

Depthwise Convolution. Bei der *Depthwise Convolution* entsteht mit jedem Filter eine eigene Output Feature Map (siehe Abbildung 2.5). Die Output Feature Maps der einzelnen Filter werden konkateniert, um den Output-Tensor zu ergeben. Hierbei entspricht die Anzahl der Output Feature Maps der Anzahl der Input Feature Maps und der Anzahl der Filter. Dabei findet allerdings kein Informationsaustausch zwischen den Kanälen statt.

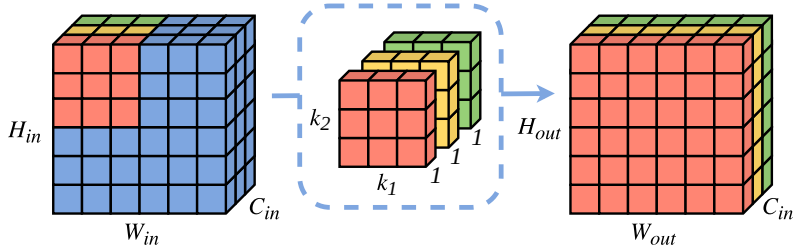


Abbildung 2.5: Depthwise Convolution
Mit jedem Filter entsteht eine eigene Output Feature Map. (Grafik angelehnt an [Bai, 2019])

Pointwise Convolution. Die *Pointwise Convolution* (oder auch einfach 1×1 Convolution genannt) ist der zweite Teil der Depthwise Separable Convolution. Diese ist für den Informationsaustausch zwischen den Kanälen zuständig. Die Pointwise Convolution bezieht alle Input Feature Maps ein. Die Filtergröße beträgt jedoch nur 1×1, wie in Abbildung 2.6 zu erkennen ist. Damit berechnet jedes Filter-Set der Pointwise Convolution eine Linearkombination der Feature Maps. Die Anzahl der Filter-Sets für die Pointwise Convolution bestimmt die Anzahl der Output Feature Maps.

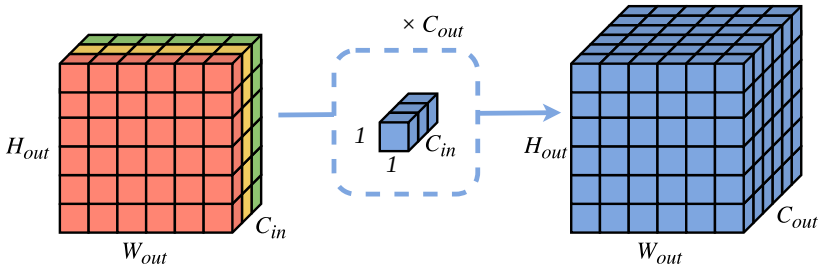


Abbildung 2.6: Pointwise Convolution
Informationsaustausch zwischen den Kanälen. (Grafik angelehnt an [Bai, 2019])

Durch die Depthwise Separable Convolution reduziert sich der Rechenaufwand auf $H_{out} \cdot W_{out} \cdot k_1 \cdot k_1 \cdot C_{in}$ für die Depthwise Convolution (Im Gegensatz zur Standard Convolution fehlt hier die Multiplikation mit C_{out}). Für die Pointwise Convolution kommt noch $H_{out} \cdot W_{out} \cdot C_{in} \cdot C_{out}$ hinzu. Insgesamt werden somit – im Vergleich zur Standard Convolution – nur noch $\frac{1}{C_{out}} + \frac{1}{k_1 \cdot k_2}$ der Rechenoperationen benötigt. In den meisten Fällen ist die Anzahl der Outputkanäle deutlich höher als die Filtergröße, und es wird ein quadratischer Filter mit $k = k_1 = k_2$ verwendet. Für einen groben Überschlag verringert sich der Rechenaufwand im Vergleich zur Standard Convolution also um $\frac{1}{k^2}$.

Als Trade-Off für den geringeren Berechnungsaufwand ist die verringerte Repräsentationskraft hinzunehmen.

2.2.4 Group Convolution

Die *Group Convolution* wurde ursprünglich im AlexNet von [Krizhevsky et al., 2012] eingeführt, um das Training auf mehrere GPUs aufzuteilen. Da aktuelle Grafikkarten über ausreichend Speicher und Rechenleistung verfügen, ist eine Aufteilung auf mehrere GPUs für die meisten Anwendungen nicht mehr notwendig. Dennoch kann durch Verwendung von Group Convolutions die Anzahl der Rechenoperationen gesenkt werden. Einige Netzwerke, wie das ShuffleNet aus [Zhang et al., 2018], profitieren sogar von der Group Convolution und erreichen bessere Ergebnisse als mit der Standard Convolution.

Ähnlich wie bei der Depthwise Convolution werden bei der Group Convolution für eine Output Feature Map nicht alle Input Feature Maps C_{in} sondern nur ein Teil benötigt.

Der Input-Tensor wird in G Gruppen aufgeteilt mit jeweils $\frac{C_{in}}{G}$ Feature Maps. Dies wird auch als *Channel Split* bezeichnet. Auf jede Gruppe wird nun separat eine Convolution angewandt. Dabei enthält jede Gruppe $\frac{C_{out}}{G}$ Filter-Sets. Die Output Feature Maps der einzelnen Gruppen werden anschließend entlang der Kanalachse konkateniert. Somit ergibt sich ein Output-Tensor mit C_{out} Feature Maps. Diese Vorgehensweise ist in Abbildung 2.7 dargestellt.

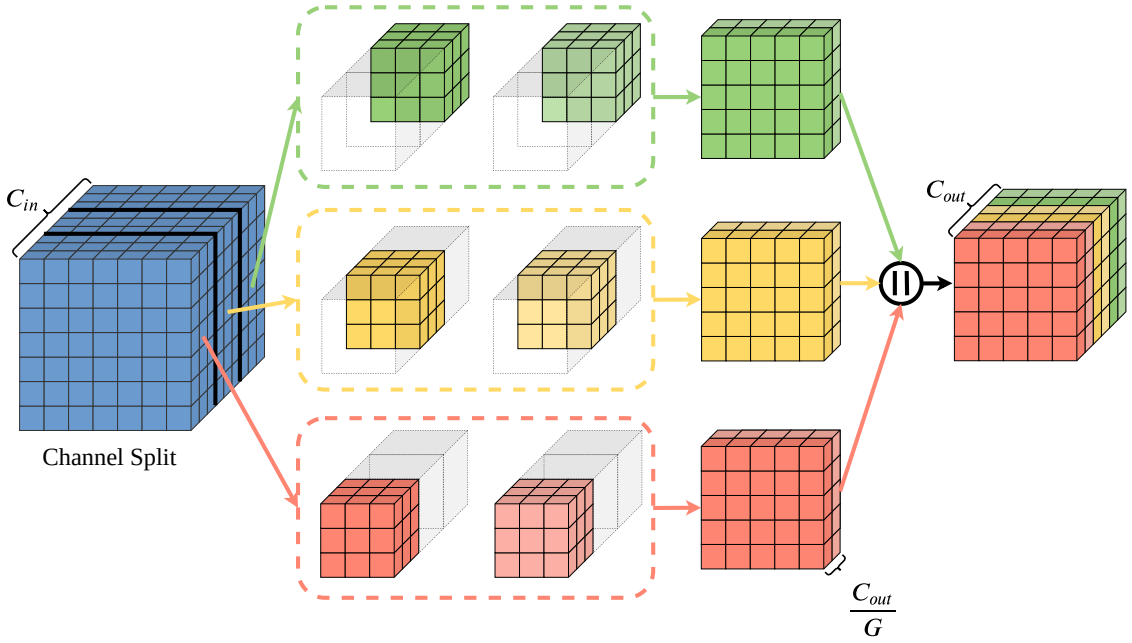


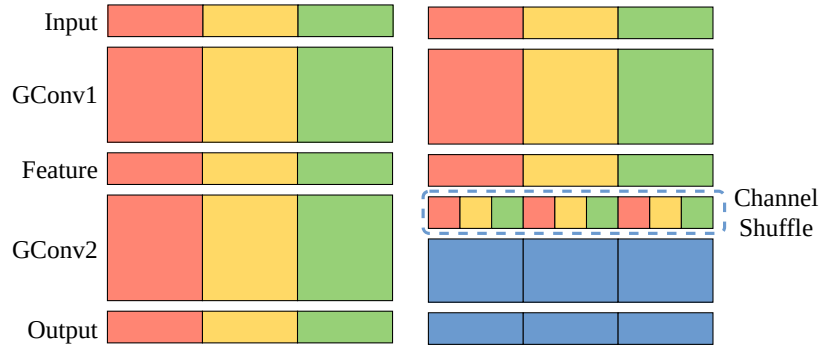
Abbildung 2.7: Group Convolution

*Der Input-Tensor wird in G Gruppen aufgeteilt. Auf jede Gruppe wird separat eine Convolution angewandt. Anschließend werden die resultierenden Feature Maps konkateniert. (Grafik angelehnt an [*Bai*, 2019])*

Durch Verwendung von G Gruppen mit jeweils $\frac{C_{in}}{G}$ Inputkanälen und $\frac{C_{out}}{G}$ Outputkanälen reduziert sich der Rechenaufwand um den Faktor G im Vergleich zur Standard Convolution. Der Extremfall tritt ein, wenn $G = C_{in} = C_{out}$. Dann ist die Group Convolution identisch zur Depthwise Convolution (dem erste Teil der Depthwise Separable Convolution).

Bei mehreren aufeinanderfolgenden Group Convolutions tritt das Problem ein, dass kein Informationsaustausch zwischen den Gruppen stattfindet. Bleibt die Anzahl der Gruppen G konstant und gilt $C = C_{in} = C_{out}$, dann erhält jede Gruppe immer wieder denselben Teil der Input Feature Maps. Dies ist in Abbildung 2.8a veranschaulicht.

Im ShuffleNet aus [Zhang et al., 2018] wurde daher die Operation *Channel Shuffle* eingeführt. Channel Shuffle bezeichnet ein Umschichten des Tensors. Bei einer darauf folgenden Group Convolution erhält somit eine Gruppe Feature Maps aus jeder Gruppe der vorherigen Group Convolution. Somit ist der Informationsaustausch zwischen den Gruppen wiederhergestellt (siehe Abbildung 2.8b).



(a) ohne Channel Shuffle **(b)** mit Channel Shuffle

Abbildung 2.8: Channel Shuffle
Channel Shuffle mit zwei hintereinander geschalteten Group Convolutions. GConv steht für Group Convolution. (a) zwei Convolution Layer mit derselben Anzahl an Gruppen. Jeder Output-Kanal bezieht nur die Input-Kanäle derselben Gruppe mit ein. Kein Informationsaustausch zwischen den Gruppen. (b) Input- und Output-Kanäle sind vollständig miteinander verbunden, wenn GConv2 nach GConv1 Feature Maps aus verschiedenen Gruppen entgegennimmt.
(Grafik angelehnt an [Zhang et al., 2018])

In der Implementierung wird das Channel Shuffle folgendermaßen realisiert: Wenn $C_{out} = G \cdot C_{group}$, dann wird die Kanaldimension zuerst in (G, C_{group}) umgeformt, wodurch der Tensor eine zusätzliche Dimension enthält. Der Tensor wird anschließend entlang dieser beiden Dimensionen transponiert und wieder zurück geformt.

2.3 Aufbau von Segmentierungsarchitekturen

In diesem Abschnitt soll der grundlegende Aufbau moderner Deep-Learning-basierter Segmentierungsarchitekturen erklärt werden. Im Allgemeinen besteht eine Segmentierungsarchitektur aus einem *Encoder* und einem, daran anschließenden, *Decoder*. Dieser Aufbau ist in Abbildung 2.9 veranschaulicht. Einige Ansätze enthalten zusätzlich ein *Kontextmodul* zur expliziten Erfassung von Kontextinformationen. Dieses Kontextmodul wird zwischen dem Encoder und dem Decoder platziert, wie in Abbildung 2.9 zu erkennen ist.
Nachfolgend wird genauer auf den Encoder und den Decoder eingegangen. Konkrete Kontextmodule werden in Abschnitt 3.5 erklärt.

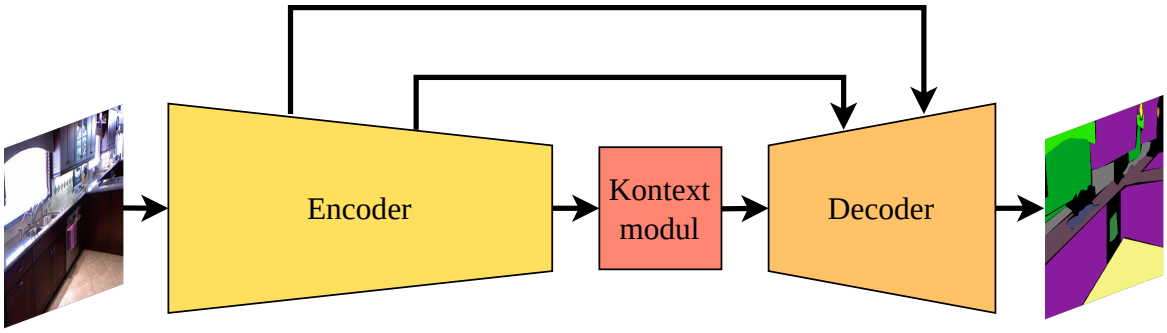


Abbildung 2.9: Grundlegender Aufbau von Segmentierungsarchitekturen
Eine Segmentierungsarchitektur besteht meistens aus einem Encoder und einem, daran anschließenden, Decoder. Einige Architekturen enthalten ein zusätzliches Kontextmodul zwischen dem Encoder und dem Decoder. Manche Ansätze verwenden sogenannte Skip Connections zwischen dem Encoder und dem Decoder, um höherauflösende Feature Maps aus dem Encoder für die Verfeinerung der Feature Maps des Decoders zu verwenden.

2.3.1 Encoder

Der Encoder ist hauptsächlich für die Extraktion wesentlicher Features verantwortlich. Hierfür werden häufig allgemeine Basisnetzwerke, wie beispielsweise ResNets, verwendet. Im Encoder wird die räumliche Auflösung der Feature Maps verringert, um den Berechnungsaufwand zu reduzieren. Zum Downsampling werden Strided Convolutions[6] oder Pooling-Layer verwendet. Durch das Downsampling vergrößert sich zudem das rezeptive Feld, wodurch das Netzwerk mehr Kontextinformationen erhält. Einige Encoder modifizieren die Basisnetzwerke durch Verwendung von *Dilated Convolutions*[7]. Dadurch kann ein zu starkes Downsampling vermieden, und dennoch das rezeptive Feld vergrößert werden.

2.3.2 Decoder

An den Encoder schließt sich der Decoder an. Im Decoder wird die räumliche Auflösung der Feature Maps wiederhergestellt. Dies kann über sogenannte *Transposed Convolutions*

[6] Der Stride (deutsch: Schrittweite) eines Convolution Layers gibt an, um wie viele Pixel der Kernel verschoben wird. Typischerweise wird ein Stride von 1 verwendet. Bei Strided Convolutions ist der Stride > 1. Dadurch verringert sich die räumliche Auflösung der Output Feature Maps.
[7] Dilated Convolutions werden in Abschnitt 2.2.1 erklärt.

oder *bilineares Upsampling* erfolgen. Die Transposed Convolution ist die Umkehrung der Convolution, kann aber selbst wieder als Convolution implementiert werden. Mithilfe der Transposed Convolution kann der Decoder das Upsampling lernen.[8] Dieses erlernte Upsampling ist jedoch deutlich rechenaufwendiger als einfaches bilineares Upsampling. Einige Ansätze verwenden zusätzliche *Skip Connections* zwischen dem Encoder und dem Decoder. Dadurch werden höherauflösende Encoder Feature Maps verwendet, um die Segmentierung im Decoder – vor allem an den Objektkanten – zu verfeinern. Diese Skip Connections sind symbolisch in Abbildung 2.9 dargestellt.
Zusätzlich ist der Decoder für die finale Klassifikation der einzelnen Pixel zuständig. Dafür entspricht die Anzahl der Output Feature Maps C_{out} der letzten Convolution gleich der Klassenanzahl des Datensatzes C_{class}. Jede Feature Map speichert somit die Aktivierungen einer Klasse. Für die finale Segmentierung wird das Maximum entlang der Kanalachse bestimmt.

2.4 Grundlagen zu Tiefenbildern

Für die RGBD-Segmentierung wird neben dem RGB-Bild ein zusätzliches Tiefenbild benötigt. Im Tiefenbild gibt jeder Pixel die Entfernung zur Kamera an. Abschnitt 2.4.1 stellt die Entstehung solcher Tiefenbilder dar. Der nachfolgende Abschnitt 2.4.2 beschreibt die Registrierung der RGB- und Tiefenbilder.

2.4.1 Entstehung von Tiefenbildern

Um Tiefenbilder zu erhalten, werden spezielle Kamerasysteme benötigt. Diese lassen sich in passive und aktive Systeme unterteilen. Bei der Verwendung von passiven Stereokameras werden die Tiefenwerte mithilfe von Triangulation und der Epipolargeometrie ermittelt. Aktive Kameras verwenden externe Lichtquellen, wodurch die Tiefenerfassung erleichtert wird. Zur Vermeidung einer visuellen Szenenveränderung wird dabei häufig Licht aus dem nahen Infrarotbereich verwendet. Einige aktive Kameras projizieren ein codiertes *strukturiertes Lichtmuster*. Über den Kamerasensor wird dieses Muster wiedererkannt. Wie bei passiven oder aktiven Stereokameras entsteht

[8] Weiterführende Informationen zur Transposed Convolution können in [Dumoulin und Visin, 2016] nachgelesen werden.

das Tiefenbild mithilfe von Triangulation. *Time-of-Flight*-Systeme messen direkt die Entfernung, indem sie modulierte Lichtsignale auf die Szene senden und die Zeit bis zur Rückkehr des Lichtsignals messen.[9] Aufgrund von Messfehlern, Reflexionen und Verdeckungen enthalten Tiefenbilder häufig Lücken, an denen kein Tiefenwert bestimmt werden kann. Mithilfe zeitlich benachbarter Frames können fehlende Werte bestimmt und damit Lücken ersetzt werden.

2.4.2 Registrierung der RGB- und Tiefenbilder

Für die RGBD-Segmentierung ist es wichtig, dass das RGB- und das Tiefenbild zueinander registriert sind. Das heißt, dass beispielsweise eine Tischecke in beiden Bildern an der exakt selben Pixelposition zu finden sein soll. Um solche registrierten RGBD-Bilder zu erhalten, mussten früher manuell kalibrierte Kamerasysteme verwendet werden. Diese Aufbauten waren meist teuer und nicht mobil. Aktuelle RGBD-Kameras, wie beispielsweise die Kinect2 von Microsoft, sind vorkalibriert und können auch auf mobilen Robotern verwendet werden. Mithilfe der Kameraparameter werden das RGB- und das Tiefenbild dabei zueinander registriert.[10]

Einige Ansätze verwenden eine HHA-Codierung der Tiefenbilder, um die einkanaligen Tiefenbilder in eine dreikanalige geozentrische Repräsentation umzuwandeln. Da diese HHA-Codierung in dieser Masterarbeit nicht verwendet wird, erfolgt eine Erklärung im Anhang A.1.2.

[9] Weiterführende Informationen zur Entstehung von Tiefenbildern und zu RGBD-Kameras sind in [Giancola et al., 2018] und in [Gross, 2019] zu finden.

[10] Quellcode zur Registrierung von RGB- und Tiefenbild für die Kinect2: `https://github.com/OpenKinect/libfreenect2/blob/master/src/registration.cpp`, zuletzt aufgerufen am 11.01.2020

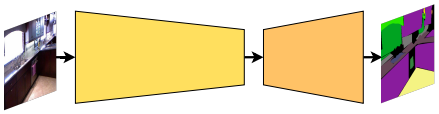
Kapitel 3

State of the Art: Effiziente Segmentierung

Nachdem im Abschnitt 2.3 der grundlegende Aufbau von Segmentierungsarchitekturen dargelegt wurde, fokussiert sich dieses Kapitel auf effiziente Ansätze zur semantischen Segmentierung. Über verschiedene Optimierungen der Netzwerkarchitektur kann eine niedrige Inferenzzeit und ein geringer Speicherbedarf erreicht werden. In diesem Kapitel werden aktuelle Ansätze vorgestellt und wesentliche Konzepte herausgearbeitet.
Zuerst wird in Abschnitt 3.1 der strukturelle Aufbau verschiedener Architekturen erklärt. Anschließend wird in Abschnitt 3.2 auf den Encoder und verschiedene Encoder-Blöcke eingegangen. Abschnitt 3.3 erklärt das Downsampling, also wie im Encoder die räumliche Auflösung reduziert wird. In Abschnitt 3.4 werden effiziente Decoder und Fusionsmöglichkeiten von Encoder Feature Maps in den Decoder dargestellt. Danach wird in Abschnitt 3.5 auf effiziente Kontextmodule eingegangen. Zuletzt werden in Abschnitt 3.6 die vorgestellten Verfahren miteinander verglichen und ein Fazit gezogen.

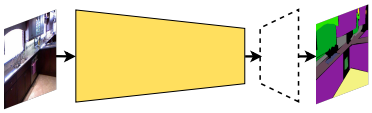
3.1 Struktureller Aufbau

Zuerst wird der strukturelle Aufbau verschiedener Netzwerkarchitekturen zur effizienten semantischen Segmentierung betrachtet und aktuelle Publikationen eingeordnet. Hierbei liegt der Fokus auf dem Grundprinzip und der Motivation dahinter. In Abbildung 3.1 sind die verschiedenen Strukturen schematisch dargestellt und konkrete Netzwerke zugeordnet. Nachfolgend werden diese Strukturen erklärt.



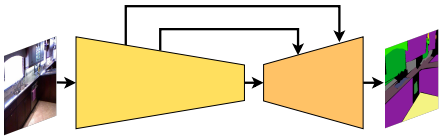
ENet [Paszke et al., 2016]
ERFNet [Romera et al., 2018]
MAVNet [Nguyen et al., 2019]
ESNet (Efficient Symmetric Network) [Wang et al., 2019b]

(a) Lineare Encoder-Decoder-Architektur
(Abschnitt 3.1.1)



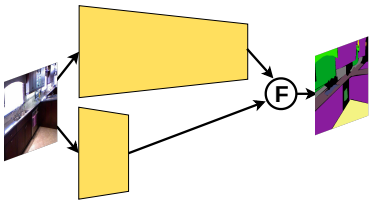
LEDNet [Wang et al., 2019a]
DABNet [Li et al., 2019a]
EDANet [Lo et al., 2019]

(b) Architektur ohne Decoder
(Abschnitt 3.1.2)



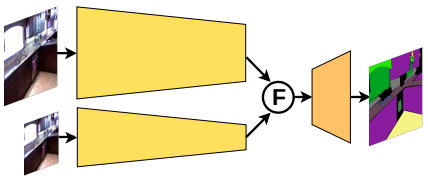
SQNet [Treml et al., 2016]
ESPNet [Mehta et al., 2018]
ESPNetv2 [Mehta et al., 2019b]
DiCENet [Mehta et al., 2019a]
Light-Weight RefineNet [Nekrasov et al., 2018]
MobileNetv3 [Howard et al., 2019]
FPENet [Liu und Yin, 2019]
SwiftNet [Oršić et al., 2019]
Fast-SCNN [Poudel et al., 2019]

(c) Architektur mit Skip Connections zwischen Encoder und Decoder
(Abschnitt 3.1.3)



BiSeNet [Yu et al., 2018]
ESNet (Edge-Based Segmentation Network) [Lyu et al., 2019]
ExtremeC3Net [Park et al., 2019]

(d) Zwei parallele Netzwerke für semantische und Kanten-Features
(Abschnitt 3.1.4)



ICNet [Zhao et al., 2018]
DFANet [Li et al., 2019b]
Variante des SwiftNets [Oršić et al., 2019]

(e) Kaskadierte Architektur
(Abschnitt 3.1.5)

Abbildung 3.1: Struktureller Aufbau von effizienten Segmentierungsarchitekturen
Schematische Abbildungen der verschiedenen Netzwerkstrukturen zur semantischen Segmentierung. F steht für eine Fusion mehrerer Netzwerkzweige.

3.1.1 Lineare Encoder-Decoder-Architektur

Bei den linearen Architekturen (siehe Abbildung 3.1a) kommen keine Skip Connections zwischen Encoder und Decoder vor. Diese Architekturen bestehen aus einem Encoder und einem, sich daran anschließenden, Decoder. Im Encoder werden wesentliche Features extrahiert und die räumliche Auflösung verringert. Im Decoder wird die räumliche Auflösung schrittweise wiederhergestellt und die Klassenentscheidung für die einzelnen Pixel getroffen.

Eine Gemeinsamkeit dieser Ansätze ist die geringe Downsamplingrate von maximal 8, welche deutlich niedriger ist als die typische Downsamplingrate von 32 der ResNets in [He et al., 2016a]. Dadurch gehen weniger räumliche Details verloren und Skip Connections zwischen Encoder und Decoder sind nicht zwingend notwendig.

3.1.2 Architektur ohne Decoder

Bei geringen Downsamplingraten von beispielsweise 8 kann der Decoder auch weggelassen werden (siehe Abbildung 3.1b), um eine höhere Effizienz zu erreichen. Die semantische Segmentierung wird in diesem Fall lediglich über eine, sich an den Encoder anschließende, 1×1 Convolution erreicht. Dabei entspricht die Anzahl der Outputkanäle der Klassenanzahl des Datensatzes. Das Ergebnis wird anschließend bilinear hochskaliert, um der räumlichen Auflösung des Eingabebilds zu entsprechen. Hierbei ist es wichtig, dass im Encoder – oder über ein entsprechendes Kontextmodul[1] – bereits genügend Kontextinformationen gesammelt werden. Das Weglassen des Decoders resultiert häufig nur in einer minimalen Verschlechterung der mean Intersection over Union (mIoU)[2], führt aber zu einer deutlich schnelleren Inferenzzeit.[3]

[1] Auf Kontextmodule wird in Abschnitt 3.5 eingegangen.

[2] Die mIoU wird in Abschnitt 5.7 definiert.

[3] Das LEDNet [Wang et al., 2019a] wird hier eingeordnet obwohl in der Publikation von einem Decoder die Rede ist. Da dort jedoch nach dem Upsampling keine weiteren Layer folgen, wird jener Teil des Netzwerks in dieser Masterarbeit als Kontextmodul bezeichnet und im Anhang A.2.3 erklärt.

3.1.3 Architektur mit Skip Connections zwischen Encoder und Decoder

In vielen Architekturen werden Skip Connections zwischen dem Encoder und dem Decoder eingesetzt (siehe Abbildung 3.1c). Dabei sollen höherauflösende Encoder Feature Maps die Segmentierung im Decoder – vor allem an den Objektkanten und bei kleinen Objekten – verfeinern. Insbesondere bei höheren Downsamplingraten von 16 oder 32 können dadurch bessere Ergebnisse erzielt werden. Der Einsatz von Skip Connections erleichtert somit die einfache Verwendung von klassischen vortrainierten Basisnetzwerken[4], die häufig eine höhere Downsamplingrate aufweisen. Die hohe Anzahl an aktuellen Netzwerken deutet darauf hin, dass dieser Ansatz vielversprechend ist.

3.1.4 Zwei parallele Netzwerke für semantische und Kanten-Features

Durch das Downsampling im Encoder gehen wichtige räumliche Details verloren. Mithilfe von Skip Connections oder lediglich geringen Downsamplingraten kann dieser Informationsverlust gemildert werden. Eine andere Variante ist es, ein zusätzliches flaches Netzwerk für die Beibehaltung räumlicher Details zu verwenden.
Für die Aufgabe der semantischen Segmentierung werden zwei parallele Netzwerke verwendet (siehe Abbildung 3.1d). Im ersten, tieferen Netzwerk werden wesentliche Features extrahiert und Kontextinformationen gesammelt. In diesem Netzwerk ist die Downsamplingrate hoch, um den Berechnungsaufwand gering zu halten. Hierfür können klassische Basisnetzwerke eingesetzt werden. Das zweite, parallele Netzwerk enthält nur wenige Layer mit einer insgesamt geringen Downsamplingrate. Dieses Netzwerk ist dafür verantwortlich räumliche Details beizubehalten und Kanten-Features zu extrahieren.
Am Ende werden die beiden Netzwerke fusioniert. Im ESNet und im BiSeNet werden dafür die Feature Maps beider Netzwerke konkateniert und mit darauf folgenden Convolutional Layern weiter verarbeitet.[5]

[4] Wichtige Basisnetzwerke werden in Abschnitt 2.1 eingeführt.

[5] Die Autoren des Fast-SCNN [Poudel et al., 2019] beschreiben ihr Netzwerk als zwei-zweigiges Netzwerke mit geteilten Low Level Features. Das Netzwerk kann aber auch – wie in dieser Masterarbeit – als eine Architektur mit Skip Connections zwischen dem Encoder und dem Decoder eingeordnet werden.

3.1.5 Kaskadierte Architektur

Bei den kaskadierten Architekturen (siehe Abbildung 3.1e) wird das Bild in unterschiedlichen Auflösungsstufen verarbeitet. Es existieren also mehrere parallele Netzwerkzweige, die am Ende miteinander kombiniert werden. Die kaskadierten Architekturen ähneln in dieser Hinsicht den zwei parallelen Netzwerken für semantische und Kanten-Features. Beim ICNet und einer Variante des SwiftNets werden unterschiedliche Auflösungsstufen des Bildes als Eingabe für drei (beziehungsweise zwei) Encoderzweige verwendet. Im Gegensatz dazu wird im DFANet der Output eines Sub-Netzwerks bilinear hochskaliert und wieder als Input für eine weitere Kaskadenstufe verwendet. Der Input der zweiten Kaskadenstufe setzt sich aus den Low-Level-Features der ersten Kaskadenstufe und dem bilinear hochskalierten Output der ersten Kaskadenstufe zusammen. Insgesamt besteht das DFANet aus drei Kaskadenstufen. Für die Segmentierung werden die Outputs der Kaskadenstufen fusioniert.

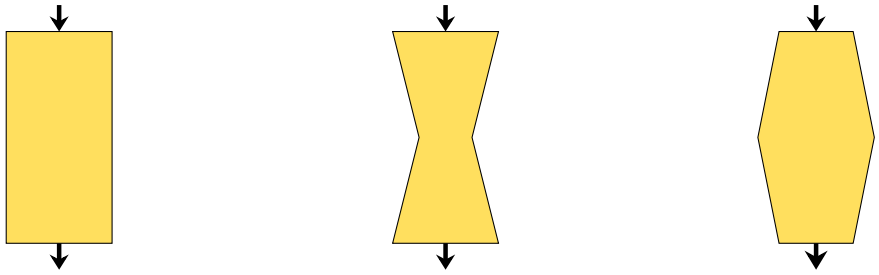
In Abschnitt 3.6 werden die erzielten Ergebnisse mit den verschiedenen Strukturen miteinander verglichen und ein Fazit gezogen.

3.2 Effiziente Encoder-Blöcke

Der Encoder eines Segmentierungsnetzwerks ist hauptsächlich für die Extraktion von wesentlichen Features verantwortlich. Dabei setzt sich der Encoder häufig aus Blöcken zusammen, die im Grundprinzip den Residual-Blöcken[6] ähneln.
In den Encoder-Blöcken werden zumeist Dilated Convolutions[7] eingesetzt, um das rezeptive Feld zu vergrößern und ein zu hohes Downsampling zu vermeiden. Typischerweise nimmt die Dilationrate mit der Netzwerktiefe zu. In vielen Architekturen werden dafür aufsteigende Zweierpotenzen verwendet.[8] Ein Nachteil von Dilated Convolutions ist allerdings die Entstehung von *Gridding-Artefakten*, welche im Anhang A.2.1 erklärt werden.

[6] Residual-Blöcke werden in Abschnitt 2.1 vorgestellt.
[7] Dilated Convolutions werden in Abschnitt 2.2.1 definiert.
[8] Dilationraten mit aufsteigenden Zweierpotenzen kommen beispielsweise im ENet [Paszke et al., 2016], im ERFNet [Romera et al., 2018], im MAVNet [Nguyen et al., 2019], im DABNet [Li et al., 2019a], im EDANet [Lo et al., 2019] und im ICNet [Zhao et al., 2018] vor.

Im Kontext der *effizienten* semantischen Segmentierung werden im Encoder zudem Factorized Convolutions[9], Depthwise Separable Convolutions[10] und Group Convolutions[11] verwendet. Diese helfen den Rechenaufwand zu minimieren.
Die Encoder-Blöcke lassen sich in Non-Bottleneck-Blöcke, Bottleneck-Blöcke und Invertierte Bottleneck-Blöcke einteilen. In Abbildung 3.2 sind diese schematisch dargestellt und konkrete Blöcke zugeordnet. Nachfolgend werden relevante Encoder-Blöcke vorgestellt und deren Unterschiede herausgearbeitet. Zusätzlich sind weitere, weniger relevante Encoder-Blöcke im Anhang A.2.1 zu finden.



Non-Bottleneck-Block
(Abschnitt 3.2.1)
- Non-Bottleneck-1D-Block
- Split-Shuffle-Non-Bottleneck-Block
- Parallel Factorized Convolution Unit

Bottleneck-Block
(Abschnitt 3.2.2)
- ESP-Modul
- EESP-Modul
- DAB-Block
- C3-Modul

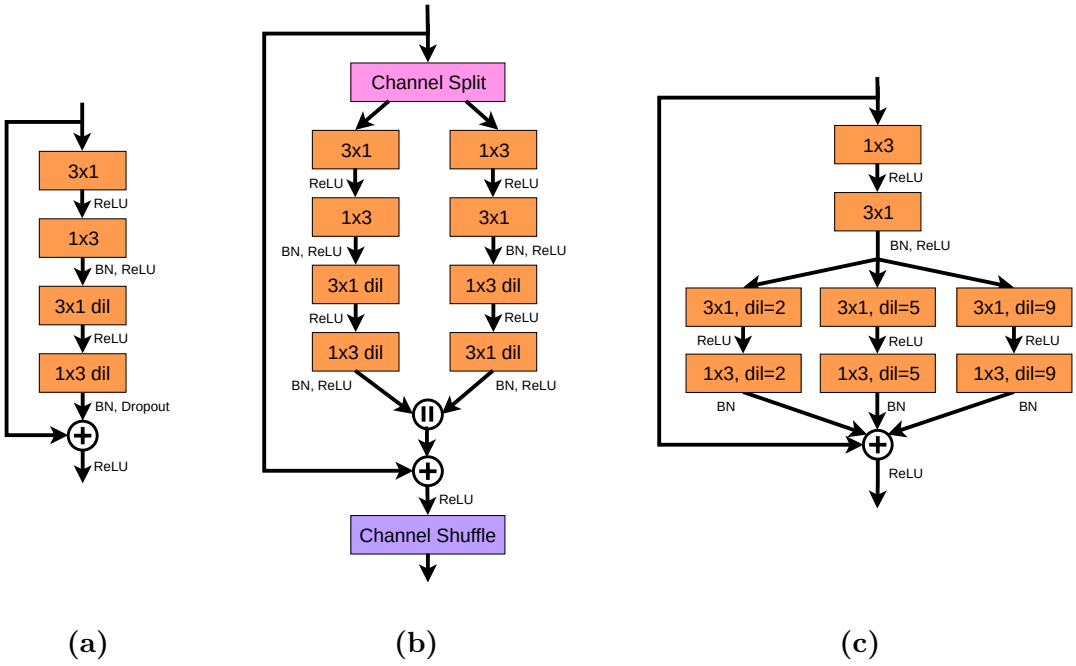
Invertierter Bottleneck-Block
(Abschnitt 3.2.3)
- Inverted-Residual-Bottleneck-Block
- Inverted-Residual-Bottleneck-Block mit Squeeze-and-Excitation
- FPE-Block

Abbildung 3.2: Encoder-Blöcke
Einordnung verschiedener Encoder-Blöcke und Zuordnung konkreter Blöcke. Beim Non-Bottleneck-Block bleibt die Kanalanzahl im gesamten Block konstant. Beim Bottleneck-Block findet zu Beginn eine Kanalreduktion und beim invertierten Bottleneck-Block eine Kanalexpansion statt.

3.2.1 Non-Bottleneck-Blöcke

Der bekannteste Non-Bottleneck-Block ist der ResNet-Non-Bottleneck-Block, welcher in Abschnitt 2.1 erklärt ist. Bei den Non-Bottleneck-Blöcken findet keine Kanalreduktion oder -expansion statt. Die Kanalanzahl bleibt also im gesamten Block konstant. Die hier vorgestellten Blöcke verwenden Factorized Convolutions, um den Rechenaufwand zu reduzieren. In Abbildung 3.3 sind typische Vertreter solcher Blöcke dargestellt.

[9] Factorized Convolutions werden in Abschnitt 2.2.2 definiert
[10] Depthwise Separable Convolutions werden in Abschnitt 2.2.3 definiert.
[11] Group Convolutions werden in Abschnitt 2.2.4 definiert



(a) **(b)** **(c)**

Abbildung 3.3: Non-Bottleneck-Blöcke
(a) Non-Bottleneck-1D-Block des ERFNets [Romera et al., 2018]
(b) Split-Shuffle-Non-Bottleneck-Block des LEDNets [Wang et al., 2019a]
(c) Parallel Factorized Convolution Unit des ESNets von [Wang et al., 2019b]

Non-Bottleneck-1D-Block. Für die semantische Segmentierung wurden Factorized Convolutions erstmals im ENet von [Paszke et al., 2016] und anschließend im ERFNet von [Romera et al., 2018] eingesetzt. Im *Non-Bottleneck-1D-Block* (dargestellt in Abbildung 3.3a) des ERFNets werden die beiden 3×3 Convolutions des ResNet-Non-Bottlenecks durch die faktorisierte Variante ersetzt, wodurch der Block effizienter wird. Zwischen der 3×1 und der 1×3 Convolution wird ein ReLU als zusätzliche Nichtlinearität eingefügt, um die Repräsentationskraft des Netzwerks zu erhöhen. Das zweite Factorized-Convolutions-Paar ist zugleich eine Dilated Convolution.

Im ESNet (An Efficient Symmetric Network) von [Wang et al., 2019b] wird dieser Block als Factorized Convolutions Unit bezeichnet. In späteren Layern des Encoders wird dort anstelle einer Kernelgröße von k=3 ein größerer Kernel mit k=5 verwendet. Dadurch kann ein größeres rezeptives Feld erfasst werden. Zwei aufeinanderfolgende 3×3 Convolution ergeben dieselbe rezeptive Feldgröße wie eine 5×5 Convolution. Durch die Faktorisierung hat jedoch die faktorisierte 5×5 Convolution nur 10 Parameter und

Rechenoperationen im Gegensatz zu den 12 Parametern und Rechenoperationen bei den zwei faktorisierten 3×3 Convolutions.[12] Insofern können damit weniger Layer, bei gleichbleibendem rezeptiven Feld und gleichzeitiger Einsparung von Parametern und Rechenoperationen, verwendet werden.

Split-Shuffle-Non-Bottleneck-Block. Im Split-Shuffle-Non-Bottleneck-Block (dargestellt in Abbildung 3.3b) des LEDNets von [Wang et al., 2019a] wird der Input zuerst in zwei Gruppen aufgeteilt. Jede Gruppe erhält dabei nur die Hälfte der Feature Maps. Anschließend folgen jeweils zwei Paare an Factorized Convolutions, vergleichbar zum Non-Bottleneck-1D-Block. Die beiden Gruppen werden anschließend konkateniert und über die elementweise Addition mit dem Input verknüpft. Am Ende des Blocks schließt sich noch ein Channel Shuffle[13] für den Informationsaustausch zwischen den Gruppen an. Da jede der beiden Gruppen nur die Hälfte der Feature Maps erhält, verringert sich der Rechenaufwand im Vergleich zum Non-Bottleneck-1D-Block um die Hälfte.

Parallel Factorized Convolutions Unit (PFCU). Auch die Parallel Factorized Convolutions Unit (siehe Abbildung 3.3c) des ESNet [Wang et al., 2019b] erweitert den Non-Bottleneck-1D-Block. Anstelle eines zweiten Factorized-Dilated-Convolutions-Paar werden nun allerdings mehrere parallele Factorized Convolutions mit unterschiedlichen Dilationraten verwendet. Dadurch können gleichzeitig Kontextinformationen unterschiedlicher Größe erfasst werden. Die Outputs der drei Zweige werden anschließend mit dem Input elementweise addiert.

Die hier vorgestellten Blöcke nutzen Post-Activation, wie im ResNetv1 [He et al., 2016a], anstelle der im ResNetv2 [He et al., 2016b] vorgeschlagenen Pre-Activation. Das bedeutet, dass Batch Normalization und ReLU erst nach der Convolution angewandt werden. Laut [Lo et al., 2019] hat das den Vorteil, dass zur Inferenz die Batch Normalization mit der vorhergehenden Convolution fusioniert werden kann, was zu einer geringeren Inferenzzeit führt.

[12] genaue Berechnung der Parameter:
faktorisierte 5×5 Convolution: 5×1 und 1×5 ergibt $5 + 5 = 10$ Parameter.
zwei faktorisierte 3×3 Convolutions: Zweimal 3×1 und 1×3 ergibt $(3 + 3) \cdot 2 = 12$ Parameter.

[13] Die Operation Channel Shuffle wird in Abschnitt 2.2.4 erklärt.

3.2.2 Bottleneck-Blöcke

Bei den Bottleneck-Blöcken findet zuerst eine Kanalreduktion statt. Das hat zur Folge, dass die nachfolgenden Convolutions mit weniger Kanälen berechnet werden müssen und somit effizienter sind. Der bekannteste Bottleneck-Block ist der ResNet-Bottleneck-Block, welcher in Abschnitt 2.1 erklärt ist. Die hier vorgestellten Bottleneck-Blöcke verwenden nach der Kanalreduktion parallele Convolutions mit unterschiedlichen Dilationraten, um Kontextinformationen unterschiedlicher Größe zu extrahieren. Der Vorteil – im Vergleich zur Parallel Factorized Convolutions Unit – ist, dass die parallelen Dilated Convolutions mit insgesamt weniger Kanälen berechnet werden müssen, wodurch der Rechenaufwand sinkt.

Efficient-Spatial-Pyramid-Modul (ESP-Modul). Beim ESP-Modul (Efficient Spatial Pyramid) (dargestellt in Abbildung 3.4) des ESPNets von [Mehta et al., 2018] wird die Kanalanzahl zuerst mit einer 1×1 Convolution reduziert. Anschließend werden mittels paralleler 3×3 Convolutions mit unterschiedlicher Dilationrate Kontextinformationen unterschiedlicher Größe extrahiert. Die entstehenden Output Features werden hierarchisch fusioniert: Die Feature Maps des Zweigs mit dem kleinsten rezeptiven Feld werden mit den Feature Maps des Zweigs mit dem nächstgrößeren rezeptiven

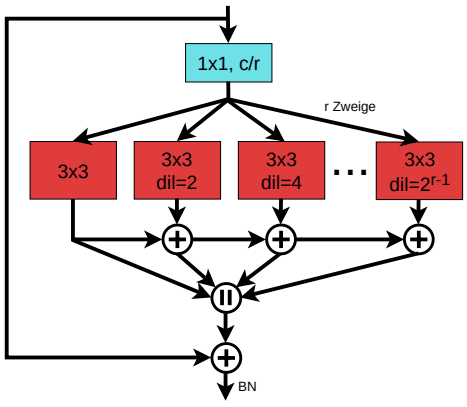


Abbildung 3.4: ESP-Modul
Efficient-Spatial-Pyramid-Modul aus dem Encoder des ESPNets von [Mehta et al., 2018].

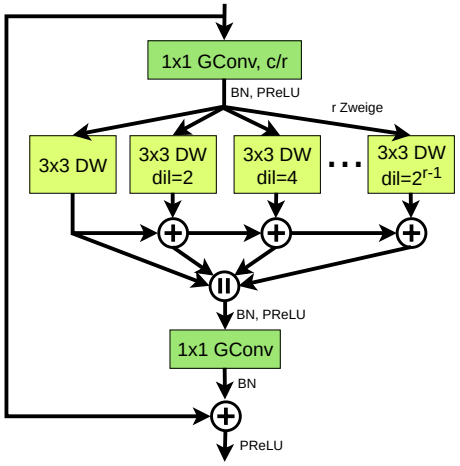


Abbildung 3.5: EESP-Modul
Extremely-Efficient-Spatial-Pyramid-Modul aus dem Encoder des ESPNetv2 von [Mehta et al., 2019b].

Feld elementweise addiert. Der Output wird schrittweise mit den Feature Maps des nächsten Zweigs addiert. Alle Outputs werden anschließend konkateniert. Mittels dieser hierarchischen Feature Fusion können Gridding-Artefakte vermieden werden. Der resultierenden Output-Tensor wird anschließend mit dem Input-Tensor des ESP-Moduls elementweise addiert.

Extremely-Efficient-Spatial-Pyramid-Modul (EESP-Modul). Das EESP-Modul (dargestellt in Abbildung 3.5) des ESPNetv2 von [Mehta et al., 2019b] modifiziert das ESP-Modul folgendermaßen: Die 1×1 Convolution wird durch eine 1×1 Group Convolution ersetzt. Die parallelen Dilated Convolutions sind nun Depthwise Separable Convolutions. Die Pointwise Convolution als zweiter Teil der Depthwise Separable Convolution wird allerdings nicht in jedem Zweig separat, sondern erst nach der Konkatenation aller Zweige angewandt. Die Pointwise Convolution wird dadurch zur Pointwise Group Convolution, wobei die Anzahl der Gruppen G gleich der Anzahl der Zweige entspricht. Diese Variante ist effizienter implementiert, als wenn in jedem Zweig eine Pointwise Convolution berechnet werden muss. Durch diese Modifikationen ist das EESP-Modul deutlich recheneffizienter als das ESP-Modul.

Depthwise-Asymmetric-Bottleneck-Block (DAB-Block). Beim DAB-Block des DABNets (dargestellt in Abbildung 3.6) wird die Kanalanzahl im Gegensatz zu den anderen Bottleneck-Blöcken mit einer 3×3 Convolution reduziert.

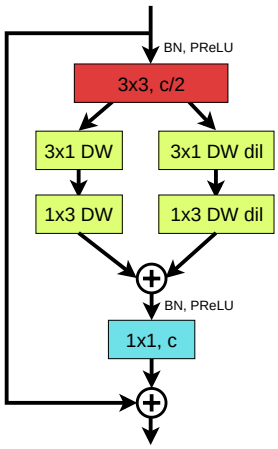


Abbildung 3.6: DAB-Block
Depthwise-Asymmetric-Bottleneck-Block des DABNets von [Li et al., 2019a].

Eine 3×3 Convolution hat mehr Repräsentationskraft und ein größeres rezeptives Feld als eine 1×1 Convolution. Dadurch können insgesamt weniger Layer verwendet werden, was letztendlich die Inferenzzeit verringert. Des Weiteren wird die Kanalanzahl nur um den Faktor $\frac{1}{2}$ reduziert, da die höchste Kanalanzahl des gesamten Netzwerks nur 128 beträgt. Nach der Kanalreduktion kommen zwei Zweige mit jeweils Depthwise Factorized Convolutions zum Einsatz. Im zweiten Zweig ist diese Convolution zugleich eine Dilated Convolution. Somit extrahiert der erste Zweig lokale und der zweite Zweig Kontextinformationen. Schließlich werden beide Zweige elementweise addiert und mit einer 1×1 Convolution zur Kanalexpansion verarbeitet. Diese 1×1 Convolution dient zugleich zum Informationsaustausch zwischen den Kanälen nach der Depthwise Convolution. Das DAB-Modul verwendet PReLU[14] als Nichtlinearität. Laut den Autoren erreicht PReLU vor allem in kleinen Netzwerken bessere Ergebnisse als ReLU.

3.2.3 Invertierte Bottleneck-Blöcke

Inverted-Residual-Bottleneck-Block. Der Inverted-Residual-Bottleneck-Block wurde erstmals im MobileNetv2 von [Sandler et al., 2018] vorgestellt und ist in Abbildung 3.7 dargestellt.

Die Idee ist es – im Gegensatz zum Bottleneck-Block – den Shortcut nicht an die Stelle mit hoher Kanalanzahl, sondern an die Stelle mit einer geringeren Kanalanzahl zu setzen. Mit der ersten 1×1 Convolution wird also die Kanalanzahl expandiert anstatt reduziert. An nächster Stelle wäre die Standard 3×3 Convolution mit der hohen Kanalanzahl sehr rechenaufwendig. Für eine höhere Effizienz wird stattdessen eine Depthwise Convolution verwendet. Schließlich muss die Kanalanzahl mit einer 1×1 Convolution wieder reduziert werden, damit die elementweise Addition mit dem Shortcut funktioniert. Für die semantische Segmentierung wird der Inverted-Residual-Bottleneck-Block im Fast-SCNN [Poudel et al., 2019] und im ESNet (Edge-Based Segmentation Network) [Lyu et al., 2019] verwendet.

[14] Die Ausgabefunktion PReLU wurde erstmals in [He et al., 2015] vorgestellt. Im Unterschied zu ReLU gibt es einen zusätzlichen Parameter α, der im Training mit gelernt wird und den Einfluss der negativen Aktivierungen bestimmt. Die Ausgabe y ergibt sich mit der Eingabe z folgendermaßen: $y = max(\alpha \cdot z, z)$.

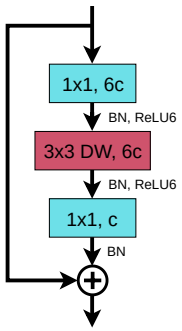


Abbildung 3.7:
Inverted-Residual-Bottleneck-Block
*Inverted-Residual-Bottleneck-Block des MobileNetv2 von [*Sandler *et al., 2018]*

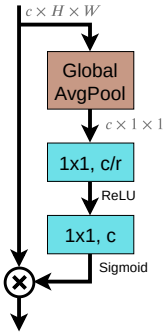


Abbildung 3.8:
Squeeze-and-Excitation-Modul
*Verwendet zur Gewichtung der Feature Maps. Ursprünglich eingesetzt im SENet von [*Hu *et al., 2018].*

Der Inverted-Residual-Bottleneck-Block wurde im MobileNetv3 [Howard et al., 2019] und im EfficientNet [Tan und Le, 2019] um das Squeeze-and-Excitation-Modul des SENets [Hu et al., 2018] erweitert. Daher soll nun zuerst dieses Modul erklärt werden.

Exkurs: Squeeze-and-Excitation-Modul (SE-Modul). Die Idee des Squeeze-and-Excitation-Moduls (dargestellt in Abbildung 3.8) ist es, die Kanäle eines Tensors unterschiedlich zu gewichten und explizite Zusammenhänge zwischen den Kanälen zu modellieren. Diese Vorgehensweise wird auch als *Feature-Rekalibrierung* bezeichnet. Mithilfe von globalen Informationen können informativere Features hervorgehoben und weniger Relevante unterdrückt werden. Konkret wird zuerst ein Global Average Pooling angewandt. Damit wird die räumliche Auflösung $H \times W$ auf 1×1 reduziert und es entsteht ein Deskriptor für die einzelnen Kanäle. Diese Operation wird als *Squeeze* bezeichnet. Daran schließt sich eine *Excitation*-Operation an, um Gewichtungen für die Kanäle zu berechnen. Hierfür wird die Kanalanzahl zuerst mit einer 1×1 Convolution reduziert und anschließend wieder expandiert.[15] Um Nichtlinearitäten einbringen zu können, kommt dazwischen ein ReLU zum Einsatz. Mit der Sigmoid-Ausgabefunktion am Ende werden die Ausgaben auf das Intervall $(0, 1)$ gestaucht. Diese Ausgaben werden anschließend zur Gewichtung des Input-Tensors verwendet.

[15] Da die räumliche Auflösung hier nur 1×1 beträgt, ist ein Fully Connected Layer an dieser Stelle identisch zu einer 1×1 Convolution.

Inverted-Residual-Bottleneck-Block mit Squeeze-and-Excitation. Im MobileNetv3 wird der Inverted-Residual-Bottleneck-Block um das Squeeze-and-Excitation-Modul erweitert. Das Squeeze-and-Excitation-Modul wird dabei nicht erst nach dem Inverted-Residual-Bottleneck-Block, sondern direkt nach der Depthwise Convolution platziert. Somit werden die expandierten (anstelle der reduzierten) Feature Maps gewichtet. Auch im EfficientNet wird dieselbe Kombination verwendet. Beide Netzwerke verwenden für die Depthwise Convolution sowohl 3×3 als auch 5×5 Filter. In beiden Netzwerken wird außerdem anstelle des ReLUs die Ausgabefunktion Swish von [Ramachandran et al., 2018] eingesetzt. Bei Swish ergibt sich die Ausgabe y mit der Eingabe z folgendermaßen: $y = z \cdot sigmoid(z)$. Da die Sigmoid-Funktion aufwendig zu berechnen ist, wird sie im MobileNetv3 durch Hard-Sigmoid ersetzt. In [Howard et al., 2019] ist Hard-Sigmoid definiert als: $y = \frac{ReLU6(z+3)}{6}$. ReLU6 begrenzt die Ausgaben von ReLU auf maximal 6. Somit ergibt sich Hard-Swish zu $y = z \cdot \frac{ReLU6(z+3)}{6}$.

In Abschnitt 3.6 werden die erzielten Ergebnisse mit verschiedenen Encoder-Blöcken gegenübergestellt. In den Experimenten in Abschnitt 6.2 wird der Einfluss auf die mIoU bei Verwendung unterschiedlicher Encoder-Blöcken untersucht.

3.3 Downsampling

Im Encoder eines Segmentierungsnetzwerks wird die räumliche Auflösung verringert, um den Rechenaufwand zu reduzieren und das rezeptive Feld zu vergrößern. In einigen effizienten Segmentierungsarchitekturen wird die räumliche Auflösung im gesamten Netzwerk nur um den Faktor 8 herunterskaliert.[16] Dadurch können räumliche Details beibehalten und der anschließende Decoder einfach gehalten oder sogar komplett weggelassen werden. Höhere Downsamplingraten von 16 oder 32 werden vor allem dann verwendet, wenn klassische vortrainierte Basisnetzwerke – wie das ResNet18 – eingesetzt werden.[17]

[16] Eine Downsamplinrate von 8 verwenden das ERFNet [Romera et al., 2018], das ENet [Paszke et al., 2016], das ESPNet [Mehta et al., 2018], das SQNet [Treml et al., 2016], das FPENet [Liu und Yin, 2019], das DiCENet [Mehta et al., 2019a], das LEDNet [Wang et al., 2019a], das DABNet [Li et al., 2019a], das EDANet [Lo et al., 2019] und das ESNet (An Efficient Symmetric Network) [Wang et al., 2019b].

[17] Eine Downsamplingrate von 16 verwenden das ESPNEtv2 [Mehta et al., 2019b], das MobileNetv3 [Howard et al., 2019] und das ESNet (Edge-Based Segmentation Network) [Lyu et al., 2019].
Eine Downsamplingrate von 32 verwenden das ICNet [Zhao et al., 2018], das Light-Weight RefineNet [Nekrasov et al., 2018], das SwiftNet [Oršić et al., 2019], das BiSENet [Yu et al., 2018], das Fast-SCNN [Poudel et al., 2019] und das DFANet [Li et al., 2019b].

Typischerweise wird beim Downsampling um den Faktor 2 die Kanalanzahl verdoppelt. Das soll den räumlichen Informationsverlust mildern.
Das VGG16 von [Simonyan und Zisserman, 2014] verwendet zum Downsampling ein Pooling gefolgt von einer Convolution zur Kanalexpansion. Diese klassische Methode des Downsamplings hat allerdings einen Nachteil: Da zuerst die räumliche Auflösung verringert und erst danach die Kanalanzahl expandiert wird, stellt das Pooling eine Art Bottleneck dar und wichtige räumliche Features können verloren gehen. Werden diese beiden Operationen vertauscht entsteht allerdings ein hoher Rechenaufwand.
Aus diesem Grund wird im ENet von [Paszke et al., 2016] Pooling parallel zur Convolution ausgeführt und die resultierenden Feature Maps konkateniert. Dieser Block wird als ENet-Initial-Downsampling-Block bezeichnet und ist in Abbildung 3.9 dargestellt. Auch einige weitere Architekturen verwenden diesen Block zum Downsampling.[18]

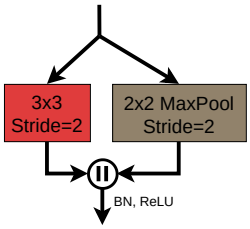


Abbildung 3.9: ENet-Initial-Downsampling-Block
Im ENet von [Paszke et al., 2016] werden zum Downsampling Convolution und Pooling parallel ausgeführt. Die resultierenden Feature Maps werden anschließend konkateniert. Der erste Zweig besteht aus einer 3×3 *Convolution mit Stride 2. Im zweiten Block kommt ein* 2×2 *Max-Pooling mit Stride 2 zum Einsatz.*

Andere Netzwerke verwenden klassische Strided Convolutions oder fügen einem gesamten Encoder-Block einen Stride hinzu. Anstelle der Identity-Verbindung wird im Shortcut Pooling eingesetzt. Diese Strategie verfolgen beispielsweise das ENet in späteren Layern oder das ESPNetv2 von [Mehta et al., 2019b].
Vor allem für die *effiziente* semantische Segmentierung ist frühes Downsampling[19] relevant. Die ersten Layer eines Netzwerks sind durch die hohe räumliche Auflösung meist auch die rechenaufwendigsten. Des Weiteren sind visuelle Informationen oft

[18] Der ENet-Inital-Downsampling-Block kommt im ENet [Paszke et al., 2016], im ERFNet [Romera et al., 2018], im LEDNet [Wang et al., 2019a], im DABNet [Li et al., 2019a] und im EDANet [Lo et al., 2019] zum Einsatz.
[19] Frühes Downsampling wird in folgenden Architekturen eingesetzt: ENet [Paszke et al., 2016], EDANet [Lo et al., 2019], ERFNet [Romera et al., 2018], ESNet (An Efficient Symmetric Network) [Wang et al., 2019b], LEDNet [Wang et al., 2019a] und MAVNet [Nguyen et al., 2019].

räumlich redundant. Daher können die Inputbilder zu Beginn für eine effizientere Repräsentation komprimiert werden. Die ersten Layer des Netzwerks sind außerdem nicht direkt an der Klassifikation beteiligt. Stattdessen sollen diese wesentliche Features extrahieren und den Input für weitere Layer vorverarbeiten.

3.4 Effiziente Decoder

Im Decoder[20] soll die räumliche Auflösung wiederhergestellt werden. Die meisten effizienten Segmentierungsarchitekturen versuchen den Decoder so klein wie möglich zu halten, oder sogar komplett wegzulassen, wie in Abschnitt 3.1.2 gezeigt wurde. Bei höheren Downsamplingraten ist allerdings ein Decoder sinnvoll.
Einige Architekturen verwenden Transposed Convolutions[21] zum Upsampling.[22] Da diese jedoch rechenaufwendig sind und unerwünschte Gridding-Artefakte erzeugen können, kommt in den meisten effizienten Segmentierungsarchitekturen bilineares oder Nearest Neighbor Upsampling zum Einsatz. Nach dem Upsampling werden Blöcke aus dem Encoder oder einfache Convolutions verwendet, um die erhaltenen Features weiter zu interpretieren.
Wird das Bild um den Faktor 16 oder 32 herunterskaliert, setzen viele Architekturen Skip Connections zwischen dem Encoder und dem Decoder ein (siehe Abschnitt 3.1.3). Dafür wird auf die Low Level Feature Maps des Encoders eine 1×1 Convolution aufgesetzt, wie in Abbildung 3.10 dargestellt ist.

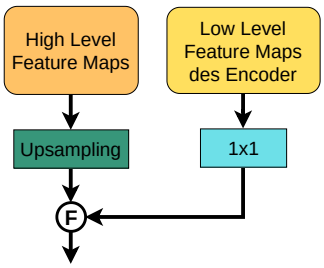


Abbildung 3.10: Fusion von Encoder Feature Maps in den Decoder
Die High Level Decoder Feature Maps werden hochskaliert und die Low Level Feature Maps mit einer 1×1 *Convolution verarbeitet. Die resultierenden Feature Maps werden fusioniert. F (Fusion): elementweise Addition oder Konkatenation entlang der Kanalachse.*

[20] Decoder von Segmentierungsarchitekturen wurden in Abschnitt 2.3.2 eingeführt.
[21] Für eine kurze Erklärung der Transposed Convolution siehe Abschnitt 2.3.2.
[22] Transposed Convolutions kommen im ERFNet [Romera et al., 2018], im SQNet [Treml et al., 2016], im ESPNet [Mehta et al., 2018] und im ESNet (An Efficient Symmetric Network) [Wang et al., 2019b] zum Einsatz.

Dadurch kann die Kanalanzahl angepasst und gelernt werden, welche Features für den Decoder relevant sind. Die High Level Decoder Feature Maps werden hochskaliert (meist über bilineare Interpolation), um der räumlichen Auflösung der Encoder Feature Maps zu entsprechen. Anschließend werden die Encoder Feature Maps in den Decoder fusioniert. Dies geschieht häufig über die elementweise Addition.[23]

Neben dieser einfachen Fusionierung gibt es auch andere Architekturen, welche die Features vor der Fusionierung gewichten. Viele dieser Wichtungen bauen auf dem Squeeze-and-Excitation-Modul auf. Nachfolgend sollen einige dieser Architektur-Module erklärt werden.

Lite Reduced Atrous Spatial Pyramid Pooling (LR-ASPP). Das Lite-Reduced-Atrous-Spatial-Pyramid-Pooling-Modul, dargestellt in Abbildung 3.11, setzt auf dem MobileNetv3 von [Howard et al., 2019] auf, um dieses zur semantischen Segmentierung zu verwenden.

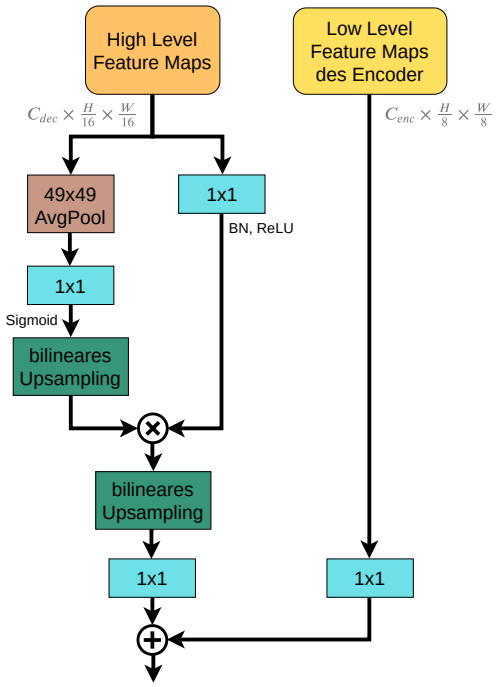


Abbildung 3.11: Lite Reduced Atrous Spatial Pyramid Pooling (LR-ASPP) *LR-ASPP aus dem MobileNet von [Howard et al., 2019].*

[23] Weitere dieser einfachen Fusionierungen sind im Anhang A.2.2 dargestellt.

Die High Level Feature Maps werden vor der Fusionierung, ähnlich wie im Squeeze-and-Excitation-Modul, gewichtet. Anstelle des Global Average Poolings wird ein Average Pooling mit großem Kernel und Stride verwendet, wofür die Autoren allerdings keine Begründung geben. Um Berechnungen einzusparen wird im Wichtungszweig nur eine 1×1 Convolution verwendet. Der Output wird anschließend bilinear hochskaliert, um der Inputgröße zu entsprechen. Zusätzlich wird auch im anderen Zweig eine 1×1 Convolution verwendet. Nach der Wichtung kommt ein bilineares Upsampling zum Einsatz, um der räumlichen Auflösung der Encoder Feature Maps zu entsprechen. Anschließend wird auf die hochskalierten Decoder Feature Maps und auf die Encoder Feature Maps jeweils eine 1×1 Convolution angewandt. Die Anzahl der Output Feature Maps entspricht dabei der Klassenanzahl des Datensatzes. Die resultierenden Feature Maps werden elementweise addiert. Für die semantische Segmentierung wird das Ergebnis schließlich um den Faktor 8 bilinear hochskaliert. Der komplette Decoder besteht also aus einem einzigen LR-ASPP-Modul mit anschließendem Hochskalieren.

Mutual Embedding Upsample (MEU). Das Mutual-Embedding-Upsample-Modul des FPENets von [Liu und Yin, 2019] besteht aus zwei *Attention-Blöcken*, wie in Abbildung 3.12 zu erkennen ist.

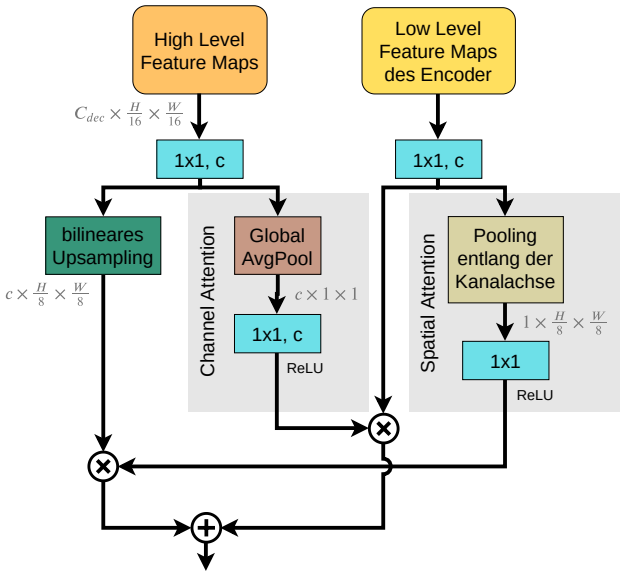


Abbildung 3.12: Mutual-Embedding-Upsample-Modul
Im Mutual-Embedding-Upsample-Modul des FPENets aus [Liu und Yin, 2019] werden Low Level und High Level Feature Maps vor der Fusionierung gewichtet.

Zuerst wird auf die High Level Feature Maps des Decoders und auf die Low Level Feature Maps des Encoders jeweils eine 1×1 Convolution angewandt. Mithilfe der High Level Feature Maps wird anschließend eine *Channel Attention Map* erzeugt: Über ein Global Average Pooling und eine, daran anschließende, 1×1 Convolution entsteht ein Kanaldeskriptor – die Channel Attention Map. Diese Channel Attention Map gibt die Relevanz der einzelnen Kanäle an. Im Fokus steht dabei der globale Kontext. Mit der Channel Attention Map werden anschließend die Low Level Feature Maps gewichtet. Mithilfe der Low Level Feature Maps wird eine *Spatial Attention Map* erzeugt: Über ein Pooling entlang der Kanalachse und einer daran anschließenden 1×1 Convolution werden alle Kanäle zu einer einzigen Feature Map – der Spatial Attention Map – zusammengefasst. Diese gibt die Relevanz der einzelnen Pixel an und hilft dabei Objekte genau zu lokalisieren und die Segmentierung an den Objektgrenzen zu verfeinern. Mit der Spatial Attention Map werden die hochskalierten High Level Feature Maps gewichtet. Die resultierenden, gewichteten Low Level und High Level Feature Maps werden schließlich elementweise addiert. Das Mutual-Embedding-Upsample-Modul findet zweimal im FPENet Anwendung.

Auch im BiSeNet wird eine gewichtete Fusionierung verwendet, welche im Anhang A.2.2 dargestellt ist. In Abschnitt 3.6 werden die verschiedenen Fusionierungen im Decoder nochmal gegenübergestellt.

3.5 Effiziente Kontextmodule

Einige Segmentierungsarchitekturen setzen ein zusätzliches Kontextmodul zwischen dem Encoder und dem Decoder ein, um explizit Kontextinformationen zu sammeln. Typischerweise werden dafür parallele Operationen ausgeführt, um Kontextinformationen unterschiedlicher Größe zu erfassen. Im Rahmen der *effizienten* semantischen Segmentierung soll dieses Kontextmodul nicht zu viele Berechnungen erfordern. Nachfolgend werden verschiedene Kontextmodule erklärt.

Pyramid-Pooling-Modul (PPM). Im Pyramid-Pooling-Modul (siehe Abbildung 3.13) aus dem PSPNet von [Zhao et al., 2017] werden lokale, globale und subglobale Kontextinformationen über vier verschiedene Pooling-Operationen zusammengefasst.

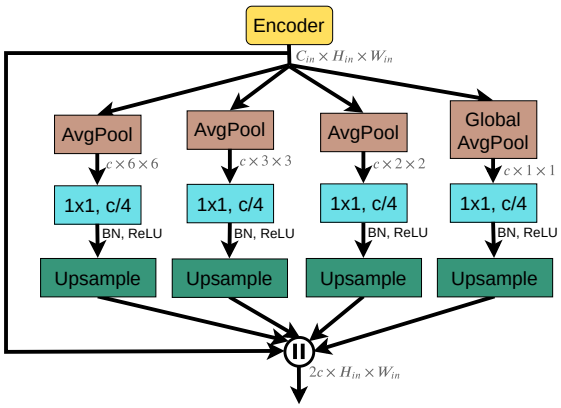


Abbildung 3.13: Pyramid-Pooling-Modul
Das Pyramid-Pooling-Modul aus [Zhao et al., 2017] verwendet parallele Zweige an Pooling-Operationen, um Kontextinformationen zu erfassen. AvgPool steht für Average Pooling.

Eine davon ist ein *Global Average Pooling*. Die Output Feature Maps des Global Average Poolings haben eine räumliche Auflösung von $H_{out} \times W_{out} = 1{\times}1$. Mittels Global Average Pooling werden also alle Feature Maps gemittelt, wodurch eine globale Kontexterfassung erfolgt. Bei den anderen drei Pooling-Operationen wird jeweils Average Pooling angewandt, sodass sich für die Output Feature Maps entsprechend eine räumliche Auflösung von 2×2, 3×3 und 6×6 ergibt. Hierbei werden also unterschiedlich große Regionen der Feature Maps gemittelt, wodurch auch lokale und subglobale Kontextinformationen erfasst werden können. An die Pooling-Operationen schließt sich jeweils eine 1×1 Convolution an, um die Anzahl der Feature Maps auf $1/4$ zu reduzieren. Die Feature Maps aller vier Zweige werden anschließend bilinear hochskaliert, um der Auflösung vor dem Pooling zu entsprechen. Der Input des Pyramid-Pooling-Moduls wird schließlich mit den Outputs der vier Zweige konkateniert. Dadurch enthält der entstehende Output-Tensor doppelt so viele Feature Maps wie der Input.

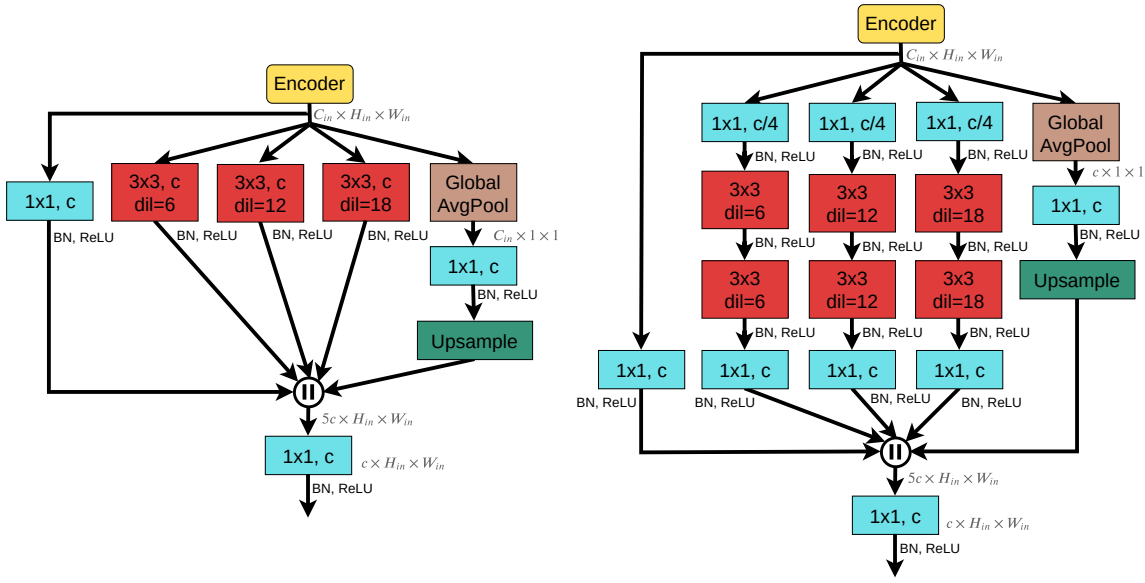
Da das Pyramid-Pooling-Modul nur 1×1 Convolutions enthält, ist dieses effizient berechenbar und wird auch in einigen effizienten Segmentierungsarchitekturen eingesetzt.[24]

[24] Für die effiziente semantische Segmentierung wird das Pyramid-Pooling-Modul im ESNet (Edge-Based Segmentation Network) [Lyu et al., 2019], im Fast-SCNN [Poudel et al., 2019], im ICNet [Zhao et al., 2018] und abgewandelt im SwiftNet [Oršić et al., 2019] verwendet.

Ein weiteres bekanntes Kontextmodul ist das Atrous-Spatial-Pyramid-Pooling-Modul (ASPP), erstmals vorgestellt im DeepLabv2 von [Chen et al., 2018a]. Da dieses Modul viele Rechenoperationen erfordert, wurde es im AdapNet++ aus [Valada et al., 2019] so verändert, sodass die Anzahl der Rechenoperationen um fast 90% sinkt. Die Autoren bezeichnen dieses, deutlich effizientere Modul als Efficient-Atrous-Spatial-Pyramid-Pooling-Modul (eASPP-Modul). Zuerst soll nun das ASPP-Modul erklärt werden, um nachfolgend die Unterschiede im eASPP-Modul darzustellen.

Exkurs: Atrous-Spatial-Pyramid-Pooling-Modul (ASPP-Modul). Im Atrous-Spatial-Pyramid-Pooling-Modul (veranschaulicht in Abbildung 3.14a) aus [Chen et al., 2018a] werden – anstelle des Average Poolings – Dilated Convolutions mit unterschiedlichen Dilationraten verwendet.
Dadurch können Kontextinformationen miteinbezogen werden, ohne (bei entsprechendem Padding) die räumliche Auflösung zu reduzieren. Neben den drei Dilated Convolution Zweigen wird ein zusätzlicher Global-Average-Pooling-Zweig (vergleichbar zum Pyramid-Pooling-Modul) und ein Zweig mit einer 1×1 Convolution verwendet. In allen fünf Zweigen wird die Kanalanzahl auf c reduziert. Die fünf Zweige werden konkateniert und das Ergebnis mit einer 1×1 Convolution verarbeitet, um die Kanalanzahl wieder auf c zu reduzieren.

Efficient-Atrous-Spatial-Pyramid-Pooling-Modul (eASPP-Modul). Das Efficient-Atrous-Spatial-Pyramid-Pooling-Modul (dargestellt in Abbildung 3.14b) aus [Valada et al., 2019] basiert auf zwei Prinzipien: der Kaskadierung von Dilated Convolutions und der Verwendung von Bottlenecks. Die Kaskadierung der Dilated Convolutions führt zu einem größeren rezeptiven Feld im Vergleich zu einer einzelnen Dilated Convolution. Des Weiteren werden bei der Kaskadierung mehr Pixel für die Berechnung mit einbezogen, als wenn nur eine einzige Dilated Convolution mit entsprechend größerem rezeptiven Feld verwendet wird. Das eASPP-Modul besteht, wie das ASPP-Modul, aus fünf Zweigen. Die drei Zweige mit den Dilated Convolutions werden durch die kaskadierten Bottleneck-Zweige ausgetauscht. Mit einer 1×1 Convolution wird die Kanalanzahl zuerst auf $\frac{1}{4}\,c$ reduziert. Dabei steht c für die Kanalanzahl am Ausgang des Kontextmoduls. Anschließend werden zwei Dilated Convolutions verwendet. Zuletzt wird die Kanalanzahl mit einer 1×1 Convolution um den Faktor 4 expandiert.



(a) ASPP **(b)** eASPP

Abbildung 3.14: Kontextmodule mit parallelen Dilated Convolutions
Beide Kontextmodule verwenden parallele Dilated Convolutions mit unterschiedlichen Dilationraten zur Erfassung von Kontextinformationen. AvgPool steht für Average Pooling, dil für die Dilationrate.
(a) Das Atrous-Spatial-Pyramid-Pooling-Modul ist rechenaufwendig und daher nicht für die effiziente semantische Segmentierung geeignet.
(b) efficient-Atrous-Spatial-Pyramid-Pooling-Modul aus AdapNet++ [Valada et al., 2019]. Durch das Bottleneck-Design und die Kaskadierung an Dilated Convolutions ist dieses Modul deutlich recheneffizienter als das ASPP und erzielt dennoch bessere Ergebnisse.

Ein weiteres Kontextmodul mit dem Namen Attention Pyramid Network (APN) aus dem LEDNet [Wang et al., 2019a] ist im Anhang A.2.3 zu finden.

3.6 Fazit zum SotA zur effizienten Segmentierung

Um die vielen Verfahren zur effizienten Segmentierung einordnen zu können, sollen diese hier verglichen und eine Auswahl für diese Masterarbeit getroffen werden. Aufgrund der hohen Anzahl an aktuellen Publikationen werden zuerst ältere Verfahren, wie das ENet [Paszke et al., 2016] und das SQNet [Treml et al., 2016], ausgeschlossen.

Fast alle Netzwerke wurden auf dem Outdoor-Segmentierungsdatensatz Cityscapes[25] trainiert. Wegen fehlender Vergleichbarkeit werden daher Netzwerke ausgeschlossen, welche nicht auf Cityscapes trainiert wurden. Somit fallen das MAVNet [Nguyen et al., 2019], das ExtremeC3Net [Park et al., 2019] und das Light-Weight-RefineNet [Nekrasov et al., 2018] weg.

In Tabelle 3.1 sind alle restlichen recherchierten Netzwerke dargestellt. Die Architekturen sind nach ihrem strukturellen Aufbau (siehe Abschnitt 3.1) gegliedert. Nachfolgend wird die Tabelle kurz beschrieben:
Netzwerk. Name des Netzwerks und zugehörige Quelle.
Konferenz. Angegeben, falls bereits veröffentlicht. Bei noch unveröffentlichten Verfahren ist lediglich die Jahreszahl der Veröffentlichung auf arXiv angegeben.
Encoder. Kurze Zusammenfassung des Encoder-Aufbaus
Downsampling (DS). Downsamplingrate im gesamten Netzwerk und Art und Weise des Downsamplings
Kontextmodul (KM). Nur angegeben, falls vorhanden
Decoder. Kurze Zusammenfassung des Decoder-Aufbaus
Encoder-Decoder-Fusion. Gibt an wie Encoder Feature Maps in den Decoder fusioniert werden (falls vorhanden)
#Param. Parameteranzahl im gesamten Netzwerk. Je geringer die Parameteranzahl, desto kleiner ist das Netzwerk und desto weniger kann es auswendig lernen.
FLOPs. Anzahl an Rechenoperationen in Floating Point Operations im gesamten Netzwerk.
FPS. Verarbeitungsfrequenz in Frames pro Sekunde. Nicht alle Publikationen enthalten Angaben für sowohl FLOPs als auch FPS; zum Teil ist daher nur einer der beiden Werte angegeben. Je mehr FPS das Netzwerk erreicht, desto geringer ist die Inferenzzeit. Da nicht alle Zeiten auf den gleichen Grafikkarten gemessen wurden, ist die Verarbeitungsfrequenz farblich kodiert: Titan X, GTX 1080 Ti, Titan XP, Titan V. Eine direkte Umrechnung der Verarbeitungsfrequenzen von einer Grafikkarte auf eine andere Grafikkarte ist nicht möglich. Es lässt sich jedoch sagen, dass die Titan V die schnellste und die Titan X die langsamste der vier Grafikkarten sind. Die GTX 1080 Ti und die Titan XP sind ähnlich schnell.

[25] Der Outdoor-Segmentierungsdatensatz Cityscapes enthält 5000 Bilder von städtischen Straßenszenen. Die Bilder haben eine hohe räumliche Auflösung von $H \times W = 1024 \times 2048$ und sind pixelgenau annotiert. Zum Benchmarking werden 19 verschiedene Klassen evaluiert. [Cordts et al., 2016]

mIoU. mean Intersection over Union auf dem Cityscapes-Testdatensatz. Verarbeitungsfrequenz und mIoU sind – sofern nicht anders deklariert – für Bilder mit halber Auflösung ($H \times W = 512 \times 1024$) angegeben. Einige wenige Netzwerke wurden auch auf Bildern mit voller Auflösung ($H \times W = 1024 \times 2048$) trainiert. Diese Ergebnisse sind zusätzlich mit v deklariert.

Zusammenfassung. Es ist zu erkennen, dass in jeder Kategorie mIoU-Werte von über 70 erreicht werden. Der strukturelle Aufbau des Netzwerks hat also keinen entscheidenden Einfluss auf die Segmentierungsleistung. Insgesamt werden im Encoder häufig zuerst eine oder mehrere 3×3 Convolutions eingesetzt, bevor die effizienten Encoder-Blöcke Anwendung finden. Architekturen, bei welchen die verwendeten Encoder-Blöcke auf dem Non-Bottleneck-1D-Block aufbauen, verwenden zum Downsampling den ENet-Intial-Downsampling-Block. Die gesamte Downsamplingrate beträgt in diesen Netzwerken nur 8. Andere Verfahren verwenden meistens Strided Convolutions zum Downsampling. Nur wenige Netzwerke setzen Kontextmodule ein, da durch die Dilated Convolutions in den Encoder-Blöcken bereits Kontextinformationen extrahiert werden können. Kontextmodule kommen vor allem dann zum Einsatz, wenn der Encoder keine Dilated Convolutions enthält. Bis auf die beiden Netzwerke mit linearem Aufbau enthält der Decoder nur wenige Convolutions. Der Hauptfokus für die Feature-Extraktion liegt somit beim Encoder. Der Decoder ist nur noch dafür zuständig die berechneten Features zu interpretieren und die Input-Auflösung wiederherzustellen. Es lässt sich feststellen, dass Netzwerke mit geringerer Downsamplingrate seltener Skip Connections zwischen Encoder und Decoder besitzen. Da hier räumliche Details nichts so sehr verloren gehen, sind Skip Connections nicht zwingend notwendig.

FLOPs vs. FPS. Zudem fällt auf, dass die Anzahl an Rechenoperationen nicht unbedingt mit der Verarbeitungsfrequenz korreliert. Der Intuition entsprechend müsste ein Netzwerk mit wenigen Rechenoperationen eine hohe Verarbeitungsfrequenz erreichen. Bei Betrachtung der Verarbeitungsfrequenzen auf der GTX 1080 Ti erreicht allerdings das SwiftNet bei halber Auflösung mit 26 GFLOPs mehr FPS als beispielsweise das EDANet mit nur 8.97 GFLOPs (134.9 FPS vs. 108.7 FPS). Das hängt unter anderem damit zusammen, dass Factorized Convolutions, Depthwise Separable Convolutions und Group Convolutions zwar die Anzahl an Rechenoperationen verringern,

	Netzwerk (Konferenz) Jahr	Encoder	DS KM	Decoder Encoder-Decoder-Fusion	#Param FLOPs FPS	mIoU
linear	**ERFNet** [Romera et al., 2018]	13 Non-Bottleneck-1D-Blöcke	8 EN-B.	4 Non-Bottleneck-1D-Blöcke, Transposed Convs	2.1 M*	69.7
	IEEE ITS 2018		-	-	26 G* 41.7	
	ESNet [Wang et al., 2019b]	3 Non-Bottleneck-1D-Blöcke k=3, 2 Non-Bottleneck-1D-Blöcke k=5, 3 PFCU	8 EN-B.	2 Non-Bottleneck-1D-Blöcke k=5, 2 Non-Bottleneck-1D-Blöcke k=3, Transposed Convs	1.66 M	70.7
	PRCV 2019		-	-	63	
ohne Decoder	**LEDNet** [Wang et al., 2019a]	13 Split-Shuffle-Non-Bottleneck-1D-Blöcke	8 EN-B.	-	0.94 M	70.6
	ICIP 2019		APN ★	-	71	
	DABNet [Li et al., 2019a]	drei 3×3 Convs, 9 DAB-Module	8 EN-B.	-	0.76 M	70.1
	BMVC 2019		-	-	104.2	
	EDANet [Lo et al., 2019]	13 EDA-Module ★	8 EN-B., St.C.	-	0.68 M	67.3
	2018		-	-	8.97 G 108.7	
mit Skip Connections	**ESPNet** [Mehta et al., 2018]	eine 3×3 Conv, 12 ESP-Module	8 St.B.	ein ESP-Modul, Transposed Convs	0.4 M	60.3
	ECCV 2018		-	Konkatenation	4.5 G* 112*	
	ESPNetv2 [Mehta et al., 2019b]	eine 3×3 Conv, eine 3×3 DW Conv, 17 EESP-Module	16 St.B.	1x1 Convs	0.7 M*	66.2
	CVPR 2019		-	Konkatenation	2.7 G 84*	
	DiCENet [Mehta et al., 2019a]	eine 3×3 Conv, 16 DiCENet-Units □	32 Pool, St.B.	3×3 Convs	0.9 M	63.4
	2019		-	Encoder ähnlich wie bei SE wichten, dann 3×3 Conv, Addition mit Decoder	2.2 G	
	MobileNetv3 [Howard et al., 2019]	eine 3×3 Conv, 15 Inverted-Residual-Bottleneck-Blöcke (k=3 und k=5) zum Teil mit SE	16 St.C.	LR-ASPP	1.51 M	72.6
	ICCV 2019		-	Addition	9.74 G	

	Netzwerk (Konferenz) Jahr	Encoder	DS KM	Decoder Encoder-Decoder-Fusion	#Param FLOPs FPS	mIoU
mit Skip Connections	**FPENet** [Liu und Yin, 2019]	eine 3×3 Conv, 13 FPE-Blöcke ★	8 St.B.	zwei Mutual-Embedding-Upsample-Module	0.4 M	68.0
			-	Mutual-Embedding-Upsample-Modul	5.7 G 102	
	SwiftNet [Oršić et al., 2019]	ResNet18	32 St.C.	drei 3×3 Convs	11.8 M	
	CVPR 2019		PPM (var.)	Addition	26 G v: 39.9 134.9	v: 75.5 70.2
	Fast-SCNN [Poudel et al., 2019]	eine 3×3 Conv, zwei Depthwise Separable Convs, 9 Inverted-Residual-Bottleneck-Blöcke	32 St.C.	eine Depthwise Separable Conv	1.11 M	
	2019		PPM	Addition	v: 75.3 218.8	v: 68.0 62.8
zwei parallele Netzwerke	**BiSENet** [Yu et al., 2018]	Kantennetz: drei 3×3 Convs Segmentierungsnetz: ResNet18	32 St.C.	Feature-Fusion-Modul □	49.0 M	v: 74.7
	LNCS 2018		-	Feature-Fusion-Modul □ Attention-Refinement-Modul ★	v: 65.5	
	ESNet [Lyu et al., 2019]	Kantennetz: MobileNetv2 Segmentierungsnetz: 9 EESP-Module	32 St.B., St.C.	Multilayer-Fusion-Modul □		63.2
	ICIP 2019		PPM	Multilayer-Fusion-Modul □	193 M 34	
kaskadiert	**ICNet** [Zhao et al., 2018]	größte Auflösungsstufe: drei 3×3 Convs kleinere Stufen: PSPNet50	32 St.C.	eine 1×1 Conv	26.5 M*	
	EECV 2018		PPM	Cascade Feature Fusion ★	v: 29.8 G* v: 30.3 67.2*	v: 69.5 67.1
	DFANet [Li et al., 2019b]	3 modifizierte Xception-Netze □	32 St.C.	eine 1×1 Conv	7.8 M	70.3
	CVPR 2019		-	Addition	1.7 G 160	
	SwiftNet (pyr) [Oršić et al., 2019]	jeweils ein ResNet18 für Bilder voller und halber Auflösung	32 St.C.	drei 3×3 Convs	12.9 M	v: 75.1
	CVPR 2019		-	Addition	v: 114 G v: 34	

Tabelle 3.1: Vergleich der Architekturen zur effizienten Segmentierung
KM: *Kontextmodul.* ***DS****: Downsampling (jweils Rate, darunter Methode).* ***EN-B.:*** *ENet-Inital-Downsampling-Block.* ***St.C.:*** *Strided Convolution.* ***St.B.:*** *Strided Block.* ***SE:*** *Squeeze-and-Excitation.* ★***:*** *siehe Anhang.* □***:*** *nicht relevant für die Masterarbeit, siehe Paper für eine Erklärung.* ****:*** *Wert nicht aus Original-Paper.* ***v:*** *volle Auflösung* 1024×2048 *(Standardfall* 512×1024*).* ***GPU:*** *Titan X, GTX 1080 Ti, Titan XP, Titan V.*

aber nicht immer effizient implementiert sind.[26] Daher sind sie zum Teil während der Inferenz langsamer als Standard Convolutions. Ist eine schnelle Inferenzzeit gewünscht, sind somit die Floating Point Operations (FLOPs) kein geeignetes Vergleichsmaß. Allerdings erschweren die angegebenen Verarbeitungsfrequenzen auf unterschiedlichen Grafikkarten den Vergleich. Im Rahmen dieser Masterarbeit werden daher neue Inferenzzeiten mit verschiedenen Encoder-Blöcken auf einem NVIDIA Jetson AGX Xavier als Zielplattform gemessen (siehe Abschnitt 6.2).

Erkenntnisse und Auswahl eines Verfahrens. Neben der Inferenzzeit ist vor allem die mIoU relevant. Ein Blick auf die mIoU auf dem Cityscapes-Test-Datensatz zeigt, dass Netzwerke mit einem ResNet18-Encoder die besten Ergebnisse erzielen. Trotz der hohen Anzahl an Rechenoperationen, ist die Inferenzzeit gering. Aus diesem Grund soll die erste Netzwerkarchitektur dieser Masterarbeit auf einem ResNet18-Encoder basieren.
Um die anderen Encoder-Blöcke nicht zu vernachlässigen, sollen außerdem die Encoder-Blöcke des ResNets durch andere Encoder-Blöcke ersetzt werden. Die Entscheidung für bestimmte Encoder-Blöcke setzt sich aus den gemessenen Inferenzzeiten und der erzielten mIoU des Ursprungsnetzwerks mit diesem Encoder-Block zusammen.
Durch die hohe Downsamplingrate des ResNets von 32 sind außerdem Skip Connections zwischen Encoder und Decoder notwendig. Die guten Ergebnisse der Architekturen ohne Decoder beweisen, dass es ausreicht im Decoder ab einer bestimmten Auflösung nur noch bilinear hochzuskalieren. In Anlehnung an das SwiftNet [Oršić et al., 2019] soll die Segmentierung daher ab einer räumlichen Auflösung von $\frac{H}{4} \times \frac{W}{4}$ nur noch bilinear hochskaliert werden. Der restliche Decoder soll wie im SwiftNet aus nur einer 3×3 Convolution pro Auflösungsstufe bestehen.
Da das ResNet keine Dilated Convolutions besitzt und somit nur begrenzt Kontextinformationen extrahiert, soll nach dem Encoder ein Kontextmodul eingesetzt werden. Aufgrund der häufigen Verwendung des Pyramid-Pooling-Moduls in anderen Architekturen, soll auch in dieser Masterarbeit zuerst das Pyramid-Pooling-Modul eingesetzt werden.

[26] Bereits im ShuffleNet [Zhang et al., 2018] wurde die ineffiziente Implementierung der Depthwise Convolutions erwähnt. Auch weitere Publikationen wie [Wu et al., 2018] und [Qin et al., 2018] befassen sich mit dieser Thematik.

Kapitel 4

State of the Art: RGBD-Segmentierung

Ein weiteres Forschungsfeld, neben der effizienten semantischen Segmentierung, ist die RGBD-Segmentierung. Mithilfe von Tiefenbildern können geometrische Informationen genutzt werden, um die Segmentierung zu verbessern. Das Ziel der RGBD-Segmentierung ist es, die komplementären Features der RGB- und Tiefenbilder für eine erhöhte Segmentierungsleistung auszunutzen. RGB und Tiefe werden hierbei als zwei verschiedene Modalitäten bezeichnet.

Forscher stehen nun vor der Aufgabe Tiefenbilder sinnvoll in Segmentierungsarchitekturen einzubeziehen. Eine erste, naive Idee ist es, das Tiefenbild als zusätzlichen Inputkanal zu verwenden. Mit dem rot-, grün-, blau- und Tiefenkanal erhält das Netzwerk somit einen 4-kanaligen Input. Es stellte sich allerdings schon früh heraus, dass sich dieser Ansatz eher nicht eignet: Das Tiefenbild beeinhaltet eine völlig andere Statistik als das RGB-Bild. Dadurch wird es für das Netzwerk sehr schwierig aus diesen unterschiedlichen Informationen sinnvoll zu lernen.[1]

Aus diesem Grund wird ein intelligenterer Ansatz benötigt, um Tiefenbilder in Segmentierungsarchitekturen einzubeziehen. Dabei werden in dieser Masterarbeit lediglich Ansätze betrachtet, welche ausschließlich auf Deep-Learning-Methoden beruhen. Ein Großteil der Ansätze fokussiert sich darauf Tiefenbilder und RGB-Bilder zuerst in zwei getrennten Netzwerkzweigen separat zu verarbeiten und die resultierenden

[1] In [Hazirbas et al., 2016] erreicht das Netzwerk mit dem 4-Kanal-Input nur eine mIoU von 31.95 im Vergleich zu 37.76 des vorgestellten FuseNets.

Feature Maps später im Netzwerk zu fusionieren. Verschiedene Fusionsmöglichkeiten werden in Abschnitt 4.1 vorgestellt. Eine andere Idee ist es, das Tiefenbild lediglich zum Training und nicht zur Inferenz zu verwenden. Dieser Ansatz hat den Vorteil zur Inferenz keine Tiefenbilder verarbeiten zu müssen, was in einer geringeren Inferenzzeit resultiert. Publikationen mit diesem Ansatz werden in Abschnitt 4.2 dargestellt. Andere Verfahren, vorgestellt in Abschnitt 4.3, verarbeiten 3D-Daten, indem sie das RGB-Bild mithilfe des Tiefenbilds in den 3D-Raum projizieren. Zuletzt werden die vorgestellten Verfahren in Abschnitt 4.4 miteinander verglichen und ein Fazit gezogen.

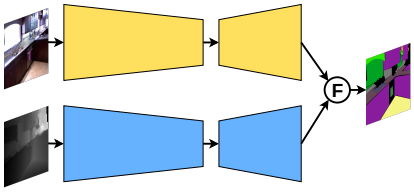
4.1 Fusionierung im Netzwerk

Viele Ansätze verwenden zuerst zwei separate Netzwerkzweige: einen für die RGB- und einen für die Tiefenbilder. Später im Netzwerk werden die beiden Netzwerkzweige fusioniert, um dann letztendlich die Segmentierung auszugeben. Abbildung 4.1 zeigt unterschiedliche Fusionsmöglichkeiten mit zugeordneten Publikationen. Dabei sind diejenigen Architekturen ausgegraut, welche nicht auf dem SUN RGB-D Datensatz[2] trainiert wurden. Diese Architekturen wurden auf anderen RGBD-Datensätzen oder auf Datensätzen bestehend aus anderen Modalitäten wie Infrarot oder Fernerkundungsbilder trainiert. Konzepte für die Fusionierung anderer Modalitäten können jedoch auch für die Fusionierung von RGB und Tiefe übernommen werden. In den nachfolgenden Abschnitten werden die unterschiedliche Fusionsmöglichkeiten dargestellt. Der Fokus liegt hierbei auf der Fusionierung der RGB- und Tiefen-Features und weniger auf der darunterliegenden Netzwerkarchitektur.

4.1.1 Zwei Netzwerke – Fusion am Ende

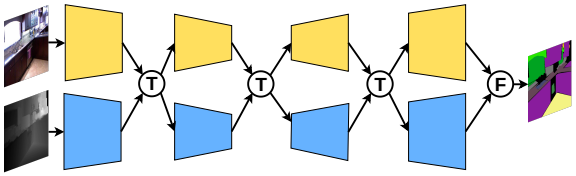
Da ein 4-Kanal-Input schlechte Ergebnisse erzielt, ist eine nächste Idee, jeweils ein Netzwerk für die RGB- und die Tiefenbilder zu trainieren und die beiden Netzwerke am Ende zu fusionieren (siehe Abbildung 4.1a). Im einfachsten Fall wird als gemeinsame Ausgabe der Mittelwert oder das Maximum der beiden Netzwerkausgaben bestimmt.

[2] Die, in dieser Masterarbeit entworfene Architektur wird auf dem SUN RGB-D Datensatz traniert. Dieser Datensatz und eine Begründung für die Wahl des Datensatzes wird in Abschnitt 5.2 beschrieben.



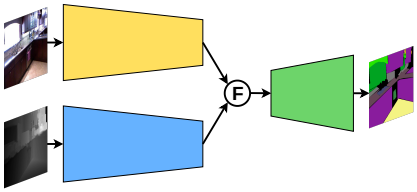
(a) zwei Netzwerke – Fusion am Ende
(Abschnitt 4.1.1)

- [Cheng et al., 2017]



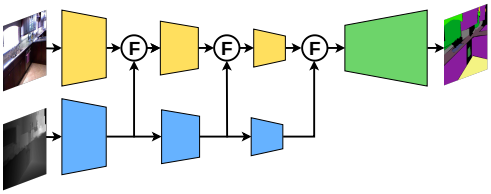
(b) zwei Netzwerke – Feature-Austausch
(Abschnitt 4.1.2)

- [Wang et al., 2016]
- [Xing et al., 2019b]



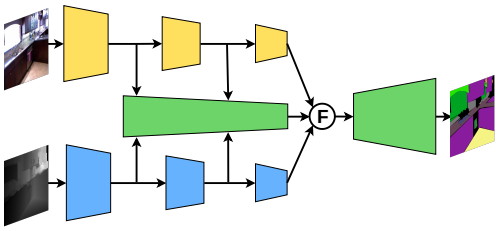
(c) Fusion am Ende der Encoder
(Abschnitt 4.1.3)

- LSTM-CF [Li et al., 2016]
- [Cao et al., 2016]



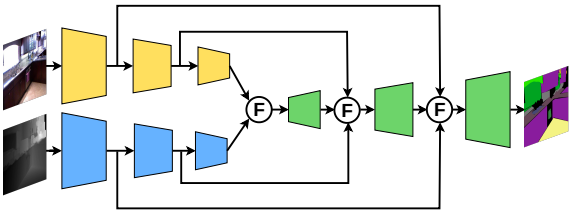
(d) Fusion in den RGB-Encoder
(Abschnitt 4.1.4)

- FuseNet [Hazirbas et al., 2016]
- RedNet [Jiang et al., 2018]
- LDFNet [Hung et al., 2019]
- [Gao et al., 2019]
- Multimodal DenseNet [Mahmood et al., 2018]
- LICODS [Agoes et al., 2019]



(e) Fusion über dritten Encoder
(Abschnitt 4.1.5)

- ACNet [Hu et al., 2019]
- V-FuseNet [Audebert et al., 2018]
- RFBNet [Deng et al., 2019]
- [Piramanayagam et al., 2018]



(f) Fusion im Decoder
(Abschnitt 4.1.6)

- RDFNet [Lee et al., 2017]
- SSMA [Valada et al., 2019]
- [Li et al., 2017]
- MFNet [Ha et al., 2017]

Abbildung 4.1: Fusionierung im Netzwerk für die RGBD-Segmentierung

Schematische Abbildungen für unterschiedliche Fusionierungen von RGB- und Tiefen-Features. Netzwerke, welche nicht auf dem SUN RGB-D Datensatz trainiert wurden, sind ausgegraut. F steht für eine Fusion mehrerer Netzwerkzweige und T für eine Feature-Transformation. Abbildungen angelehnt an [Deng et al., 2019] und [Lee et al., 2017]

In [Cheng et al., 2017] wird eine intelligentere Fusion vorgeschlagen: Mit einem sogenannten *Gated Fusion Layer* wird automatisch der Anteil beider Netzwerkzweige an der Klassifizierung einzelner Pixel gelernt. Das Gated Fusion Layer wird im Anhang in A.3.1 erklärt.
Die Berechnung zweier kompletter Netzwerke ist allerdings sehr rechenaufwendig und daher für den Einsatz auf einem mobilen Roboter eher ungeeignet. Des Weiteren findet die Fusion der RGB- und Tiefen-Features erst ganz am Ende statt. Dadurch können komplementäre Features nicht zielführend ausgenutzt werden.

4.1.2 Zwei Netzwerke – Austausch von Features

Ein weiterer Ansatz besteht darin, die beiden Netzwerke für die RGB- und die Tiefenbilder miteinander kommunizieren zu lassen (siehe Abbildung 4.1b). Das heißt, an verschiedenen Stellen können die Netzwerke Informationen austauschen und komplementäre Features nutzen.
In [Wang et al., 2016] wird für den Feature-Austausch ein *Feature Transformation Network* vorgestellt. Aufgrund der etwas älteren Publikation wird für eine Erklärung des Feature Transformation Networks auf den Anhang A.3.1 verwiesen.
In [Xing et al., 2019b] wird eine weitere Möglichkeit für den Informationsaustausch zwischen zwei getrennten ResNet-basierten Netzwerken vorgestellt. Indem die Identity-Verbindungen der Residual-Blöcke[3] durch *Idempotente Mappings* ersetzt werden, können die beiden Netzwerke Features austauschen. Auf dem SUN RGB-D Datensatz wurde im Vergleich zu Identity-Mappings allerdings nur eine Verbesserung von 47.3 auf 48.5 erzielt. Daher wird auch hier für eine ausführlichere Erklärung auf den Anhang A.3.1 verwiesen.

Im Gegensatz zu zwei völlig getrennten Netzwerken, kommt bei diesen beiden Ansätzen ein Informationsaustausch zwischen den beiden Netzwerken vor. Dennoch ist die Berechnung zweier kompletter Netzwerke rechenaufwendig.

4.1.3 Fusion am Ende der Encoder

Um nicht zwei komplette Netzwerke berechnen zu müssen, gibt es die Möglichkeit nur noch einen Decoder zu verwenden, der gemeinsame Features verarbeitet. Dafür werden die RGB- und Tiefen-Features am Ende der beiden Encoder fusioniert (siehe Abbildung 4.1c).

[3] Residual-Blöcke werden in Abschnitt 2.1 vorgestellt.

In [Cao et al., 2016] werden hierfür die Feature Maps der beiden Encoder einfach konkateniert, um den Input für den Decoder zu bilden. In [Li et al., 2016] wird dahingegen ein Long Short-Term Memorized Context Fusion (LSTM-CF) Modell vorgestellt. Mittels Long Short-Term Memory (LSTM)[4] Layer werden räumliche Abhängigkeiten und Kontextinformationen entlang der vertikalen und der horizontalen Richtung extrahiert.

Bei einer Fusion am Ende der Encoder werden die RGB- und Tiefen-Features nur an einer einzigen Stelle fusioniert. Es kann allerdings hilfreich sein bereits Low-Level-Features zu fusionieren, um somit mehr komplementäre Features nutzen zu können.

4.1.4 Fusion der Tiefen-Features in den RGB-Encoder

Laut [Mahmood et al., 2018] sollten im Allgemeinen stark korrelierte Modalitäten früh und unkorrelierte Modalitäten spät fusioniert werden. RGB und Tiefe sind beispielsweise an Kanten stark korreliert, aber in der Mitte eines texturierten Objekts unkorreliert. Daher ist es sinnvoll an mehreren Stellen im Netzwerk zu fusionieren. Um sowohl Low-Level- als auch High-Level-Features zu fusionieren, können die Features des Tiefen-Encoders nach und nach in den RGB-Encoder fusioniert werden, wie in Abbildung 4.1d dargestellt ist. Typischerweise werden die Feature Maps in jeder Auflösungsstufe einmal fusioniert.

FuseNet. Dieser Ansatz wurde erstmals vom FuseNet in [Hazirbas et al., 2016] verfolgt. Das FuseNet erweitert das SegNet[5] um einen zusätzlichen Encoder für die Tiefenbilder. Vor jeder der fünf Pooling-Operation werden die Tiefen-Features elementweise zu den RGB-Features addiert. Somit werden die Tiefen-Features genutzt, um die Feature-Extraktion des RGB-Encoders zu verbessern.
In [Gao et al., 2019] wird das FuseNet um ein sogenanntes *Global Convolutional Network* als Skip Connections zwischen Encoder und Decoder erweitert. Das Global Convolutional Network wird in Anhang A.3.1 erklärt.

[4] LSTMs wurden bereits in [Hochreiter und Schmidhuber, 1997] vorgestellt. Über verschiedene Gates können LSTMs Informationen speichern und unwichtige Informationen vergessen.

[5] Das SegNet aus [Badrinarayanan et al., 2017] verwendet das VGG16 aus [Simonyan und Zisserman, 2014] bis zum letzten Pooling-Layer als Encoder. Der Decoder besteht aus dem invertierten Encoder.

RedNet. Auch im RedNet von [Jiang et al., 2018] werden die Tiefen-Features in jeder Auflösungsstufe elementweise in den RGB-Encoder fusioniert. Anstelle des VGG16 werden hier als Encoder zwei ResNet50 bzw. ResNet34 aus [He et al., 2016a] eingesetzt, wodurch bessere Ergebnisse erzielt werden. Die summierten RGB- und Tiefen-Features werden außerdem als Skip Connection mit den Decoder Feature Maps addiert. Bei Verwendung des ResNet50 als Encoder wird die Kanalanzahl der Encoder Feature Maps zuerst mit einer 1×1 Convolution reduziert und der Kanalanzahl der Decoder Feature Maps angepasst. Beim ResNet34 entspricht die Kanalanzahl im Encoder bereits der Kanalanzahl im Decoder.

LDFNet. Auch im LDFNet von [Hung et al., 2019] werden die Tiefen-Features elementweise in den RGB-Zweig addiert. Die Autoren erweitern das ERFNet[6] um einen zweiten Encoder für die Tiefenbilder. Dieser zweite Encoder besteht aus Dense-Blöcken[7]. Da aus RGB-Bildern mehr semantische Informationen extrahiert werden können, enthält der Tiefen-Encoder weniger Layer als der RGB-Encoder. Durch die Dense-Struktur ist allerdings die maximale Kanalanzahl im Tiefen-Encoder höher als im RGB-Encoder. Um die Features der beiden Encoder elementweise addieren zu können, wird die Kanalanzahl der Tiefen-Features zuerst mit einer 1×1 Convolution an die Kanalanzahl der RGB-Features angepasst. Die Tiefen-Features werden jeweils vor und nach einem Transition-Layer zum Downsampling in den RGB-Encoder fusioniert. Da Tiefenbilder häufig mit Rauschen und Störungen versehen sind, verwenden die Autoren Luminanz als zusätzlichen Kanal. Die Luminanz wird aus den RGB-Bildern extrahiert und mit den Tiefenbildern konkateniert. Somit ist der Input des zweiten Encoders zweikanalig.

Weitere Architekturen. Das Multimodal DenseNet von [Mahmood et al., 2018] und das LICODS von [Agoes et al., 2019] addieren auch die Feature Maps des Tiefen-Encoders in den RGB-Encoder. Dabei basiert das Multimodal DenseNet auf einem DenseNet und das LICODS auf einem modifizieren SqueezeNet von [Iandola et al., 2016]. Auf die beiden Architekturen wird hier nicht näher eingegangen, da das Multimodal DenseNet für medizinische Bilder entwickelt und das LICODS für einen Datensatz mit nur sieben Klassen optimiert wurde.

[6] Das ERFNet von [Romera et al., 2018] ist eine effiziente Segmentierungsarchitektur. Das ERFNet besteht aus Non-Bottleneck-1D-Blöcken, die in Abschnitt 3.2.1 erklärt werden. Der Encoder hat eine Downsamplingrate von 8. Die ursprüngliche Auflösung wird im Decoder mittels Transposed Convolutions wiederhergestellt.

[7] Dense-Blöcke werden in Anhang A.1.1 erklärt.

4.1.5 Fusion über einen dritten Encoder

Werden die Tiefen-Features nach und nach in den RGB-Encoder fusioniert, so ist die Fusionierung nicht symmetrisch. Während der Tiefen-Encoder nur Tiefeninformationen verarbeitet, verarbeitet der RGB-Encoder nach der ersten Fusion sowohl RGB- als auch Tiefeninformationen. Der RGB-Encoder erhält zusätzliche komplementäre Tiefen-Features, wovon er profitieren kann. Allerdings wird der RGB-Encoder somit auch von den zusätzlichen Tiefen-Features abhängig. Bei stark verrauschten Tiefenbildern kann dadurch die Extraktion von RGB-Features leiden. Des Weiteren könnte auch der Tiefen-Encoder von komplementären RGB-Features profitieren. Um eine symmetrische Fusion zu erreichen, kann ein dritter, virtueller Encoder verwendet werden, wie in Abbildung 4.1e dargestellt ist.
Dieser dritte Encoder ist keiner tatsächlichen Modalität zuzuordnen, sondern verarbeitet lediglich gemeinsame RGB- und Tiefen-Features. Nach und nach werden aus den RGB- und Tiefen-Encodern modalitätsspezifische Features in den dritten Encoder fusioniert. Der dritte Encoder ist dafür zuständig diese modalitätsspezifischen Features zu interpretieren und kombinieren. Die beiden RGB- und Tiefen-Encoder bleiben unberührt und die Fusionierung wird symmetrisch.

Attention Complementary Module. Das ACNet aus [Hu et al., 2019] verwendet ein ResNet für jeden der drei Encoder. Vor der Fusionierung im dritten Encoder werden die RGB- und die Tiefen-Features zuerst jeweils ähnlich wie im Squeeze-and-Excitation-Modul[8] gewichtet, wie in Abbildung 4.2 zu sehen ist.

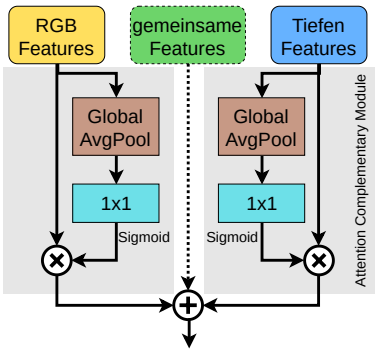


Abbildung 4.2: Fusion im ACNet
Die modalitätsspezifischen Features werden zuerst gewichtet und anschließend addiert. Abbildung nach [Hu et al., 2019]

[8] Das Squeeze-and-Excitation-Modul wird in Abschnitt 3.2.3 erklärt.

Diese Gewichtung wird als *Attention Complementary Module* bezeichnet. Dadurch können die für den dritten Encoder relevante Informationen hervorgehoben und weniger relevante Informationen unterdrückt werden. Anschließend werden die Features der beiden (bzw. nach der ersten Fusion aller drei) Encoder elementweise addiert.

Residual Fusion Block. Im ACNet werden lediglich Features des RGB- und Tiefen-Encoders in den dritten Encoder fusioniert. Im RFBNet aus [Deng et al., 2019] werden zusätzlich Features aus dem dritten Encoder in die beiden modalitätsspezifischen Encoder fusioniert. Dadurch können komplementäre Features dabei helfen die Feature-Extraktion beider modalitätsspezifischen Encoder zu verbessern. In einer *Gated Fusion Unit* des dritten Encoders werden über mehrere Gates und Convolutions unterschiedliche Features berechnet. Die gemeinsamen Features verbleiben im dritten Encoder. Für die RGB- und Tiefen-Encoder werden jeweils zusätzliche modalitätsspezifische, komplementäre Features berechnet. Diese Features werden daraufhin in die beiden modalitätsspezifischen Encoder fusioniert. Der Tiefen-Encoder des RFBNets erhält die Tiefenbilder nur in halber Auflösung. Dadurch kann der Rechenaufwand des Tiefen-Encoders um etwa $\frac{3}{4}$ reduziert werden.

Andere Architekturen. Auch im V-FuseNet von [Audebert et al., 2018] und in [Piramanayagam et al., 2018] werden die beiden Modalitäten über einen dritten Encoder fusioniert. Da beide Architekturen allerdings auf Multisensor-Fernerkundungsbildern – und nicht auf RGBD-Bildern – trainiert wurden, erfolgt die Erklärung dieser beiden Fusionierung im Anhang A.3.1.

4.1.6 Fusion im Decoder

Die RGB- und Tiefen-Features können auch erst im gemeinsamen Decoder fusioniert werden. Über mehrere Skip Connections werden dabei Feature Maps der beiden Encoder in den Decoder fusioniert, wie in Abbildung 4.1f dargestellt ist.
Im MFNet aus [Ha et al., 2017] werden hierfür die Feature Maps beider Encoder konkateniert und anschließend mit den Decoder Feature Maps derselben Auflösungsstufe elementweise addiert.

Semantics-guided-Fusion-Block. Der *Semantics-guided-Fusion-Block* aus [Li et al., 2017] wendet jeweils eine Convolution auf die Feature Maps der beiden Encoder an. Anschließend werden die resultierenden Feature Maps mit den Decoder Feature Maps konkateniert.

Multi-modaler Feature-Fusion-Block. Das RDFNet aus [Lee et al., 2017] erweitert das RefineNet[9] um einen zusätzlichen Tiefen-Encoder. Sogenannte *Multi-Modale Feature-Fusion-Blöcke* erhalten RGB- und Tiefen-Features derselben Auflösungsstufe als Input. Mittels einer 1×1 Convolution, zwei Residual-Blöcken und einer 3×3 Convolution werden zuerst jeweils Features für die anschließende Fusion extrahiert. Die entstehenden Feature Maps werden daraufhin elementweise addiert. Mit einem Residual-Pooling-Block[10] werden weiterhin Kontextinformationen extrahiert. Die fusionierten RGBD-Features zweier Auflösungsstufen werden anschließend wie im RefineNet kombiniert. Somit werden höherauflösende Feature Maps beider Encoder genutzt, um die Segmentierung im Decoder zu verbessern.

Self-Supervised Model Adaptation. Im SSMA aus [Valada et al., 2019] werden die RGB- und Tiefen-Features während der Fusionierung gewichtet, wie in Abbildung 4.3 dargestellt ist. Zunächst werden die RGB- und Tiefen-Features jeweils mit einer 1×1 Convolution verarbeitet und anschließend konkateniert. Danach wird eine Gewichtung berechnet, um bestimmte RGB- oder Tiefen-Features hervorzuheben oder zu unterdrücken. Im Gegensatz zum Squeeze-and-Excitation-Modul erfolgt allerdings keine kanalweise Gewichtung. Stattdessen wird jeder Pixel einzeln gewichtet, was sich folgendermaßen motivieren lässt: In manchen Bildregionen beinhaltet Tiefe mehr hilfreiche Informationen und in anderen Bildregion sind die RGB-Features aussagekräftiger. Das Tiefenbild enthält beispielsweise nur bis zu einer bestimmten Entfernung nützliche Werte. In weiter entfernten Bildregionen ist dahingegen das RGB-Bild informativer. Allerdings kann das Tiefenbild in überbelichteten oder sehr dunklen Bildregionen Informationen liefern, die im RGB-Bild nicht mehr zu erkennen sind. Über eine pixelweise Wichtung kann somit gelernt werden, welche Features welcher Modalität in welchen Bildregionen relevant sind. Ein weiterer Vorteil dieser selbst-überwachten Wichtung ist, dass bei Ausfall eines Sensors dennoch eine sinnvolle Segmentierung berechnet werden kann.

[9] Das RefineNet von [Lin et al., 2017] ist eine Segmentierungsarchitektur, die auf einem ResNet als Encoder aufbaut. Die niedrig auflösenden Feature Maps am Ende des Encoders werden nach und nach mit höherauflösenden Feature Maps aus früheren Layern des Encoders verfeinert. Dafür werden sogenannte RefineNet-Blöcke eingesetzt, die aus mehreren Residual-Convolutional-Blöcken, einer Multi-Resolution-Fusion und einem Chained-Residual-Pooling-Block bestehen.

[10] Der Residual-Pooling-Block besteht aus einem 5×5 Max Pooling mit Stride 1 und einer anschließenden 3×3 Convolution. Die entstehenden Feature Maps werden anschließend mit den Input Feature Maps addiert.

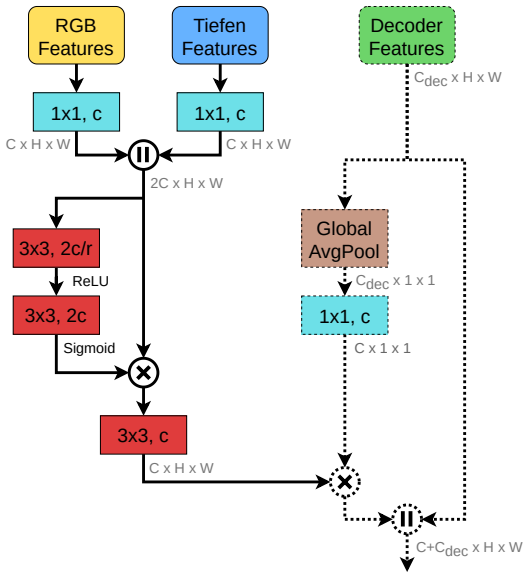


Abbildung 4.3: Self-Supervised Model Adaptation
*Selbstüberwachte Fusion. Abbildung nach [*Valada *et al., 2019]*

Die Wichtung erfolgt folgendermaßen: Mit einer 3×3 Convolution wird zuerst die Kanalanzahl reduziert und die Nichtlinearität ReLU angewandt. Mit einer weiteren 3×3 Convolution wird die Kanalanzahl anschließend wieder expandiert. Die Ausgaben werden über die Sigmoid-Funktion auf $(0, 1)$ gestaucht. Mit diesem Wichtungs-Tensor werden nun die konkatenierten Feature Maps gewichtet. Zuletzt wird die Kanalanzahl der gewichteten Feature Maps mit einer 3×3 Convolution um die Hälfte reduziert.
Die fusionierten RGBD-Features werden danach mit den Decoder Features konkateniert. Allerdings enthalten die fusionierten RGBD-Features in frühen Epochen kaum sinnvolle Informationen und verschlechtern daher eher die Feature-Extraktion im Decoder als diese zu verbessern. Daher werden die fusionierten RGBD-Features nochmals gewichtet. Über ein Global Average Pooling und eine 1×1 Convolution wird ein Kanaldeskriptor der Decoder Feature Maps erstellt. Mit diesem Kanaldeskriptor werden die RGBD-Features vor der Konkatenation gewichtet. Über diese Gewichtung kann eine bessere Korrelation zwischen den RGBD- und den Decoder Features erreicht werden.

Die Fusionierung im Decoder kann auch mit einer, bereits im Encoder stattgefundenen, Fusionierung kombiniert werden. Das RedNet[11] und das ACNet[12] enthalten beispiels-

[11] Im RedNet [Jiang et al., 2018] werden Features aus dem Tiefen-Encoder in den RGB-Encoder fusioniert (siehe Abschnitt 4.1.4).
[12] Das ACNet [Hu et al., 2019] verwendet einen dritten Encoder für die Fusion von RGB- und Tiefen-Features (siehe Abschnitt 4.1.5).

weise beide Skip Connections vom Encoder in den Decoder. Obwohl die eigentliche Fusionierung von RGB und Tiefe bereits im Encoder stattgefunden hat, werden die fusionierten Features auch im Decoder verwendet, um dort die Segmentierung zu verbessern. In einer Variante des RFBNets [Deng et al., 2019] werden zusätzlich zum dritten Encoder die RGB- und Tiefen-Features nochmals wie im SSMA in den Decoder fusioniert.

4.2 Verwendung des Tiefenbilds nur zum Training

Die Fusionierung von RGB und Tiefe im Netzwerk hat einen entscheidenden Nachteil: Da die RGB- und Tiefen-Features zunächst separat in zwei Encoder verarbeitet werden, ist mit einer erhöhten Inferenzzeit zu rechnen. Für den Einsatz auf einem mobilen Roboter ist jedoch eine geringe Inferenzzeit wünschenswert. Einige Ansätze versuchen daher das Tiefenbild ausschließlich zum Training zu verwenden und zur Inferenz mit dem RGB-Bild auszukommen. Während des Trainings kann das Tiefenbild dabei helfen eine repräsentativeres Netzwerk zu trainieren. Zur Inferenz können nun auch Kameras verwendet werden, welche lediglich ein RGB-Bild liefern.

Ein möglicher Ansatz hierfür ist die Verwendung einer zusätzlichen Lossfunktion. Das Depth-Assisted RefineNet [Chang et al., 2018b] verwendet das Tiefenbild, um das Netzwerk mit einer weiteren Lossfunktion zu trainieren. Allerdings verbessert sich die mIoU auf dem SUN RGB-D Datensatz mit dieser zusätzlichen Lossfunktion nur geringfügig, weshalb für eine Erklärung auf den Anhang A.3.2 (Seite 152) verwiesen wird.

Ein weiterer Ansatz ist das Multi Task Learning. Die Netzwerkarchitektur enthält beim Multi Task Learning einen zusätzlichen, zweiten Decoder für die Prädiktion des Tiefenbilds. Auch wenn mit Multi Task Learning bessere Segmentierungsleistungen erzielt werden können als mit einer reinen RGB-Segmentierungsarchitektur, ist es dennoch nicht möglich komplementäre Tiefen-Features auszunutzen. Wenn im RGB-Bild in überbelichteten oder sehr dunklen Bildregionen keine Details mehr erkennbar sind, können auch mit Multi Task Learning dort keine sinnvollen Features extrahiert werden. Daher wird auch hier für weitere Erklärungen auf den Anhang A.3.2 (Seite 153) verwiesen.

4.3 Verarbeitung von 3D-Daten

Eine weitere Möglichkeit Tiefenbilder für die semantische Segmentierung zu verwenden, ist die Verarbeitung von 3D-Daten. Anstelle getrennter RGB- und Tiefenbilder nehmen Menschen ihre Umwelt dreidimensional wahr. Es kann also hilfreich sein, wenn auch ein Deep-Learning-Netzwerk einen dreidimensionalen Input erhält.
In [Zhong et al., 2018] wird das 2D-RGB-Bild mithilfe des Tiefenbilds in den 3D-Raum projiziert. Das Netzwerk besteht aus 3D-Convolutions. Die 3D-Convolutions sind allerdings deutlich rechenaufwendiger als 2D-Convolutions. Ein Großteil der Berechnung ist außerdem überflüssig, da die aus 2D-Bildern projizierten 3D-Daten nur spärlich besetzt sind. Daher kann eine Anwendung in dieser Masterarbeit von vornherein ausgeschlossen werden und eine Erklärung des Ansatzes erfolgt im Anhang A.3.3.

2.5D-Convolution. In [Xing et al., 2019a] werden recheneffizientere *2.5D Convolutions* vorgestellt. Die Grundidee besteht darin mithilfe von Tiefeninformationenen eine 3D-Convolution der Größe $k \times k \times k$ über k 2D-Convolutions der Größe $k \times k$ zu approximieren. Eine beliebige 2D-Convolution eines Netzwerks kann durch die 2.5D-Convolution ersetzt werden. Mithilfe des Tiefenbilds und den intrinsischen Kameraparametern werden die Feature Maps zuerst in den 3D-Raum projiziert und anschließend entlang der Tiefendimension in k Schichten unterteilt. Auf jede Schicht wird eine 2D-Convolution angewandt. Dabei werden diejenigen Kernelelemente ausmaskiert, an denen sich keine Tiefenwerte befinden. Die Outputs der k 2D-Convolutions werden schließlich summiert und ergeben wieder Feature Maps im 2D-Raum.

In [Wang und Neumann, 2018] wird die sogenannte Depth-Aware Convolution vorgestellt, welche die Standard-Convolution um einen Tiefenähnlichkeits-Term erweitert. Da die erzielte mIoU mit diesem Ansatz im Vergleich zu anderen Verfahren deutlich geringer ist, wird für eine Erklärung auf den Anhang A.3.4 (Seite 154) verwiesen.

4.4 Fazit zum SotA zur RGBD-Segmentierung

In diesem Abschnitt werden die verschiedenden Methoden zur RGBD-Segmentierung gegenüber gestellt. In Tabelle 4.1 werden einige Architekturen miteinader verglichen. Da in dieser Masterarbeit der SUN RGB-D Datensatz verwendet wird, enthält die Tabelle lediglich Publikationen, welche die mean Intersection over Union (mIoU) auf dem SUN RGB-D Datensatz angegeben haben.

Kategorisierung	Netzwerk (Konferenz) Jahr	Basisnetzwerk	Modalität zur Inferenz	mIoU
zwei Netzwerke - Austausch von Features	[Xing et al., 2019b] ICIP 2019	ResNet101 mit idempotenten Mappings	RGB + HHA	48.50
Fusion der Tiefen-Features in den RGB-Encoder	**FuseNet** [Hazirbas et al., 2016] ACCV 2016	VGG16	RGB + Tiefe	37.29
	RedNet [Jiang et al., 2018] 2018	ResNet50 ResNet34	RGB + Tiefe RGB + Tiefe	47.80 46.80
Fusion im Decoder	[Li et al., 2017] ICIP 2017	VGG16	RGB + HHA	40.98
	RDFNet [Lee et al., 2017] ICCV 2017	ResNet152	RGB + HHA	47.70
	SSMA [Valada et al., 2019] IJCV 2019	ResNet50 (modifiziert)	RGB + Tiefe RGB + HHA	44.52 45.73
Fusion über dritten Encoder	**ACNet** [Hu et al., 2019] ICIP 2019	ResNet50	RGB + Tiefe	48.10
zusätzliche Lossfunktion	**Depth-assisted RefineNet** [Chang et al., 2018b] ICPR 2018	ResNet101 ResNet152	RGB RGB	46.00 46.30
Multi Task Learning	[Jiao et al., 2019] CVPR 2019	ResNet50	RGB	54.50
Verarbeitung von 3D-Daten	[Xing et al., 2019a] ICIP 2019	ResNet101 mit 2.5D Convolutions	RGB RGB + HHA	47.30 48.20
Depth-Aware Convolution	**Depth-Aware CNN** [Wang und Neumann, 2018] ECCV 2018	VGG16 mit Depth-Aware Convolutions	RGB + HHA RGB HHA	42.00 41.50 40.20

Tabelle 4.1: Vergleich der RGBD-Segmentierung
Enthalten sind Ansätze, bei denen die mIoU auf dem SUN RGB-D Datensatz angegeben ist.

Es ist zuerst zu erkennen, dass die ResNet-basierten Architekturen deutlich bessere Ergebnisse erreichen, als VGG16-basierte Architekturen. Daher soll auch in dieser Masterarbeit eine Architektur basierend auf Residual-Blöcken entworfen werden.
Einige Architekturen verwenden direkt das Tiefenbild als Input. Dahingegen wandeln andere Architekturen das Tiefenbild zuerst in die HHA-Codierung[13] um. Da das RedNet und das ACNet hohe mIoU-Werte ohne die HHA-Codierung erreichen, wird auch in dieser Masterarbeit auf die HHA-Codierung der Tiefenbilder verzichtet.

[13] Die HHA-Codierung der Tiefenbilder wird in Abschnitt A.1.2 erklärt

Des Weiteren ist die Berechnung der HHA-Codierung rechenaufwendig. Für einen effizienten Ansatz ist sie daher weniger geeignet.
Die mit Abstand höchste mIoU von 54.50 wird mit dem Multi-Task-Netzwerk von [Jiao et al., 2019] erreicht. Allerdings konnten in Vorexperimenten keine vergleichbaren Ergebnisse erzielt werden, weshalb in dieser Arbeit ein anderer Ansatz gewählt wird.
Die nächsthöchste mIoU von 48.50 erreicht [Xing et al., 2019b] mit zwei Netzwerken, welche über idempotente Mappings Features austauschen. Wie bereits in Abschnitt 4.1.2 erwähnt ist dieses Ergebnis jedoch weniger auf die idempotenten Mappings sondern hauptsächlich auf die beiden ResNet101 zurückzuführen. Darüber hinaus ist die Berechnung zweier kompletter Netzwerke für die Anwendung auf einem mobilen Roboter zu rechenaufwendig.
Die dritthöchste mIoU von 48.20 erreichen dieselben Autoren [Xing et al., 2019a] mit ihrem Netzwerk aufbauend auf 2.5D Convolutions. Der benötigte Rechenaufwand für das verwendete ResNet101 steigt durch die 2.5D Convolutions allerdings nochmal an. Des Weiteren ist für die 2.5D Convolutions keine Implementierung verfügbar.
Insgesamt sind alle hier präsentierten Netzwerke weit von einer effizienten Architektur entfernt, weshalb die Tabelle keine Angaben bezüglich der Verarbeitungsfrequenz enthält. Selbst das RedNet mit zwei ResNet34-Encodern hat einen aufwendigen Decoder. Im Rahmen dieser Masterarbeit werden daher nur Konzepte für die RGBD-Segmentierung und die Fusionierung von RGB- und Tiefen-Features übernommen. Das darunterliegende Netzwerk orientiert sich an den effizienten Segmentierungsarchitekturen aus Kapitel 3. In vielen Publikationen werden zuerst zwei separate Encoder für die RGB- und die Tiefenbilder verwendet. Später im Netzwerk werden die Features beider Modalitäten fusioniert. Daher soll auch in dieser Masterarbeit eine solche Architektur zum Einsatz kommen und verschiedene Fusionsmöglichkeiten im Netzwerk getestet werden. Dabei fokussiert sich diese Masterarbeit auf die Fusion im Decoder, die Fusion der Tiefen-Features in den RGB-Encoder und die Fusion über einen dritten Encoder. Die selbstüberwachte Fusion im Decoder (SSMA, siehe Abschnitt 4.1.6) soll laut Autoren auch bei schlechter Qualität einer Modalität gute Ergebnisse liefern können. Im Gegensatz dazu wird der RGB-Encoder abhängig von den Tiefen-Features, wenn diese in den RGB-Encoder fusioniert werden sollen. Ein dritter Encoder bedeutet zusätzlichen Rechenaufwand. Aus diesen Gründen wird zuerst die Fusion nach SSMA [Valada et al., 2019] getestet.

Kapitel 5

Effiziente RGBD-Indoor-Segmentierung

Nach der ausführlichen Darstellung des State of the Arts zur effizienten semantischen Segmentierung in Kapitel 3 und zur RGBD-Segmentierung in Kapitel 4 soll in diesem Kapitel das umgesetzte Verfahren zur Deep-Learning-basierten semantischen Segmentierung von RGBD-Indoor-Szenen für den Einsatz auf einem mobilen Roboter dargestellt werden. Zuerst wird in Abschnitt 5.1 die erste entwickelte Netzwerkarchitektur zur effizienten RGBD-Segmentierung vorgestellt. Danach wird in Abschnitt 5.2 der, in dieser Masterarbeit verwendete, Indoor-Datensatz SUN RGB-D vorgestellt. Der folgende Abschnitt 5.3 befasst sich mit der Datenvorverarbeitung für die Netzwerkeingabe. Die Klassenverteilung im SUN RGBD-D Datensatz ist stark unbalanciert. Daher wird im Abschnitt 5.4 die verwendete Klassengewichtung während des Trainings erklärt. Der darauffolgende Abschnitt 5.5 beschäftigt sich mit der Loss-Funktion zum Trainieren des Netzwerks. Zusätzlich zur Netzwerkausgabe wird ein Loss an mehreren Neben-Outputs des Netzwerks berechnet. Diese sogenannte *Multi Scale Supervision* wird in Abschnitt 5.6 erklärt. Abschnitt 5.7 befasst sich mit den Trainingsparametern und den Implementierungsdetails. Zuletzt wird in Abschnitt 5.8 auf die Inferenzzeitmessung eingegangen.

5.1 Basis-Netzwerkarchitektur

Die erste realisierte Netzwerkarchitektur baut auf den Erkenntnissen des State of the Art zur effizienten Segmentierung (siehe Abschnitt 3.6) und zur RGBD-Segmentierung (siehe Abschnitt 4.4) auf. Das entworfene Netzwerk zur effizienten RGBD-Segmentierung kombiniert dabei das effiziente Segmentierungsnetzwerk SwiftNet aus [Oršić et al., 2019] mit dem RGBD-Segmentierungsnetzwerk Self-Supervised Model Adaptation (SSMA) aus [Valada et al., 2019]. Die gesamte Netzwerkarchitektur ist in Abbildung 5.1 dargestellt. Aus dem SSMA wird die Fusion der RGB- und Tiefen-Features und die anschließende Fusion in den Decoder übernommen. Der ResNet18-Encoder, das Kontextmodul und der Decoder stammen aus dem SwiftNet. Wie im SwiftNet werden die Feature Maps im Decoder nur bis $\frac{1}{4}$ der Input-Auflösung verarbeitet und danach lediglich bilinear hochskaliert. Nachfolgend werden die einzelnen Bestandteile des Netzwerks näher erläutert. In den späteren Experimenten in Kapitel 6 sollen diese Bestandteile nach und nach untersucht und durch andere Module ersetzt werden.

Encoder. Für die RGBD-Segmentierung werden zwei Encoder benötigt (siehe Abbildung 5.1a). Dafür kommt jeweils ein ResNet18 zum Einsatz. Die genaue Konfiguration des ResNet18 ist in Tabelle 5.1 zu erkennen. Wie im RedNet [Jiang et al., 2018] und im ACNet [Hu et al., 2019] beträgt die Input-Auflösung $H \times W = 480 \times 640$, wodurch ein Downsampling um den Faktor 32 in beide Dimensionen ermöglicht wird.

Layer	**Output-Dimension** $C \times H \times W$
7×7 Convolution, Stride 2	$64 \times 240 \times 320$
3×3 Max-Pooling, Stride 2	$64 \times 120 \times 160$
2 ResNet-Non-Bottleneck-Blöcke	$64 \times 120 \times 160$
2 ResNet-Non-Bottleneck-Blöcke	$128 \times 60 \times 80$
2 ResNet-Non-Bottleneck-Blöcke	$256 \times 30 \times 40$
2 ResNet-Non-Bottleneck-Blöcke	$512 \times 15 \times 20$

Tabelle 5.1: Konfiguration des ResNet18-Encoder
ResNet18 ohne die letzten Layer zur Klassifikation. Nach dem Max-Pooling und den ersten zwei ResNet-Non-Bottleneck-Blöcken erfolgt das Downsampling jeweils in der ersten Convolution der 2 ResNet-Non-Bottleneck-Blöcke mit einem Stride von 2.

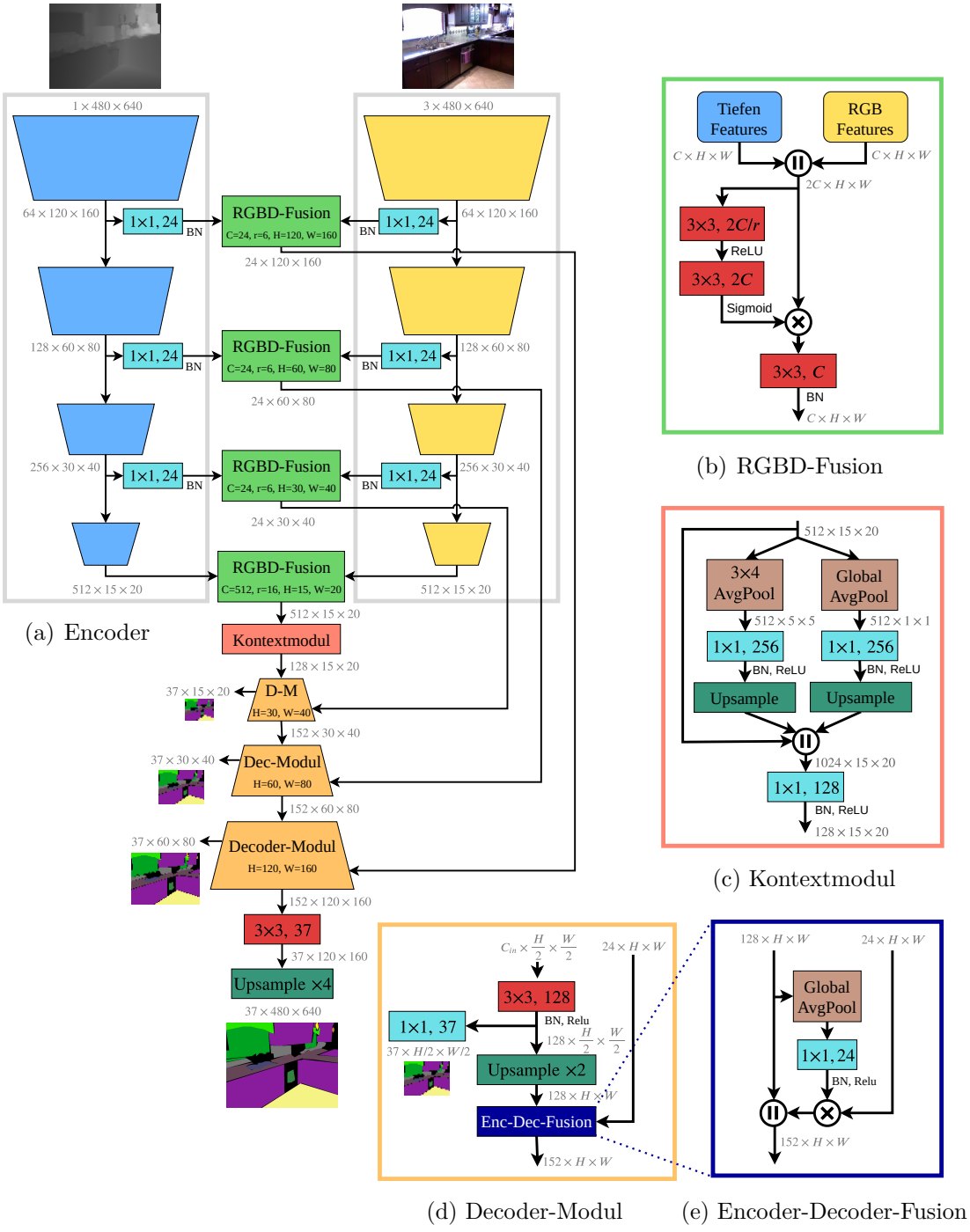


Abbildung 5.1: Netzwerkarchitektur zur effizienten RGBD-Segmentierung
Die einzelnen Netzwerkbestandteile werden im Fließtext näher erläutert.

In den Experimenten in Abschnitt 6.1 wird der ResNet18-Encoder durch einen ResNet34- bzw. einen ResNet50-Encoder ersetzt. Dadurch soll der Einfluss der Netzwerktiefe auf die Segmentierungsleistung untersucht werden. Im nachfolgenden Experiment in Abschnitt 6.2 werden die ResNet-Non-Bottleneck-Blöcke durch andere, effizientere Encoder-Blöcke aus Abschnitt 3.2 ersetzt. Hierbei wird die Frage geklärt, inwiefern sich die Inferenzzeit und die Segmentierungsleistung mit diesen Blöcken ändert.
Sowohl für den RGB-Encoder als auch für den Tiefen-Encoder werden auf ImageNet [Russakovsky et al., 2015] vortrainierte Gewichte verwendet. Der Tiefen-Encoder erhält – im Gegensatz zum RGB-Encoder – nur einen einkanaligen Input. Um dennoch die vortrainierten Gewichte verwenden zu können, werden die ImageNet-Gewichte der ersten Convolution über die Eingabekanäle elementweise addiert.

RGBD-Fusion. Am Ende der beiden Encoder werden die RGB- und Tiefenfeatures fusioniert. Hierfür kommt zunächst die pixelweise Wichtung aus dem SSMA[1] zum Einsatz. Die 1024 konkatenierten Feature Maps werden im Wichtungszweig um den Faktor $\frac{1}{16}$ reduziert. Dieser Faktor wurde aus dem Paper übernommen und entspricht zugleich dem Standard-Reduktionsfaktor des Squeeze-and-Excitation-Moduls[2]. Zusätzlich werden in jeder Auflösungsstufe Features beider Encoder fusioniert. Dafür werden vor jedem Strided-Non-Bottleneck-Block des ResNet-Encoders Features abgegriffen, wie in Abbildung 5.2 zu erkennen ist.[3] Die Kanalanzahl der abgezweigten Features wird zuerst mit einer 1×1 Convolution auf 24 Feature Maps reduziert. Anschließend werden auch diese Features beider Modalitäten fusioniert und gewichtet. Der Reduktionsfaktor im Wichtungszweig beträgt hier nur $\frac{1}{6}$, da die konkatenierten Features aus nur 48 Kanälen bestehen. Sowohl die Feature-Map-Anzahl als auch der Reduktionsfaktor sind aus dem SSMA übernommen. Die fusionierten RGBD-Features werden später über Skip Connections in den Decoder fusioniert. In den Experimenten in Abschnitt 6.4 wird die pixelweise Wichtung durch eine kanalweise Wichtung mittels eines Squeeze-and-Excitation-Moduls ersetzt. Damit wird untersucht ob und inwiefern die

[1] Die RGBD-Fusion aus dem SSMA ist in Abschnitt 4.1.6 erklärt und nochmals in Abbildung 5.1b dargestellt.

[2] Das Squeeze-and-Excitation-Modul wird in Abschnitt 3.2.3 erklärt.

[3] Im Unterschied zum SwiftNet werden die Features nach dem vollständigen vorherigen Block, das heißt erst nach dem ReLU, abgezweigt. Im SwiftNet wurden bessere Ergebnisse erreicht, wenn die Features vor dem ReLU abgegriffen werden. Allerdings zeigte sich in Vorexperimenten zu dieser Masterarbeit, dass das Abzweigen nach dem ReLU bessere Ergebnisse erzielt. Ein möglicher Grund dafür ist die Verwendung von Post-Activation mit der Reihenfolge Conv-BN-ReLU im Decoder dieser Masterarbeit anstelle der Pre-Activation BN-ReLU-Conv im SwiftNet.

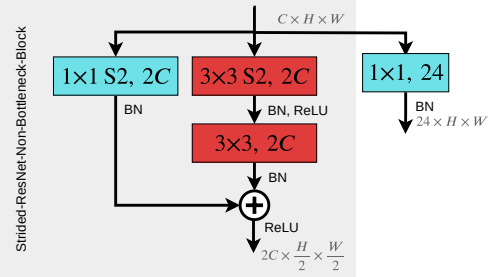


Abbildung 5.2: Abgreifen der Encoder Features für die RGBD Skip Connections
Für die RGBD-Fusion der Encoder Features, die als Skip Connections in den Decoder fusioniert werden, müssen Features aus beiden Encodern abgegriffen werden. Dafür werden jeweils Features vor dem Strided-Non-Bottleneck-Block des ResNets abgezweigt. Die Kanalanzahl der abgezweigten Features wird mit einer 1×1 *Convolution auf 24 reduziert.*

pixelweise Wichtung gegenüber der kanalweisen Wichtung einen Vorteil bringt. Des Weiteren werden hier andere Fusionsmöglichkeiten wie die Fusion über einen dritten Encoder[4] oder die Fusion der Tiefen-Features in den RGB-Encoder[5] getestet.

Kontextmodul. An die fusionierten RGBD-Features nach den beiden Encodern schließt sich ein Kontextmodul an. Zunächst wird hier wie im SwiftNet das Pyramid-Pooling-Modul[6] aus [Zhao et al., 2017] verwendet. Durch die geringe räumliche Auflösung von 15 × 20 ist es allerdings nicht möglich vier verschiedene Pooling-Größen zu wählen. Anstelle der Gridgrößen 1, 2, 3 und 6 des originalen Pyramid-Pooling-Moduls werden hier nur zwei Zweige verwendet mit den Gridgrößen 1 und 5. Das modifizierte Pyramid-Pooling-Modul ist in Abbildung 5.1c dargestellt. Die Gridgröße 1 wird im ersten Zweig mit einem Global Average Pooling erreicht. Im zweiten Zweig wird nicht-überlappendes 3×4 Average Pooling angewandt, was in einer Output-Größe von 5×5 resultiert. Des Weiteren wird nach dem Pooling die Kanalanzahl jeweils nur um die Hälfte reduziert. Das hat zur Folge, dass nach der Konkatenation – wie im originalen Pyramid-Pooling-Modul – die Hälfte der Feature Maps lokale Informationen und die andere Hälfte Kontextinformationen enthält. Da der Decoder mit nur 128 Feature Maps arbeiten soll, wird zusätzlich die Kanalanzahl am Ende des Kontextmoduls mit einer 1×1 Convolution auf 128 reduziert.

[4] Die Fusion von Tiefe und RGB über einen dritten Encoder ist in Abschnitt 4.1.5 erklärt.
[5] Die Fusion der Tiefen-Features in den RGB-Encoder wird in Abschnitt 4.1.4 vorgestellt.
[6] Das Pyramid-Pooling-Modul ist in Abschnitt 3.5 erklärt.

In den Experimenten in Abschnitt 6.5 wird das Pyramid-Pooling-Modul durch das efficient Atrous Spatial Pyramid Pooling (eASPP)-Modul aus [Valada et al., 2019] ersetzt. Dieses Kontextmodul ist rechenaufwendiger, könnte allerdings die Segmentierungsleistung verbessern. Außerdem wird untersucht, ob und inwiefern sich die Segmentierungsleistung verschlechtert, wenn kein Kontextmodul verwendet wird.

Decoder mit Decoder-Modul und Encoder-Decoder-Fusion. Im – an das Kontextmodul folgenden – Decoder werden die Feature Maps in drei Decoder-Modulen Schritt für Schritt hochskaliert. In jedem Decoder-Modul (dargestellt in Abbildung 5.1d) werden die Feature Maps mit einer 3×3 Convolution mit 128 Output Feature Maps verarbeitet und anschließend um den Faktor 2 bilinear hochskaliert. Daraufhin werden die fusionierten RGBD Feature Maps mit entsprechender Auflösung in den Decoder fusioniert. Wie im SSMA werden hierfür die RGBD Feature Maps mit einem Kanaldeskriptor der Decoder Feature Maps gewichtet (siehe Abbildung 5.1e). Danach werden die gewichteten RGBD Feature Maps und die Decoder Feature Maps konkateniert. Während des Training werden zusätzlich die Feature Maps nach der 3×3 Convolution abgezweigt. Über eine 1×1 Convolution mit 37 Output Feature Maps wird eine Segmentierung einer kleineren Auflösungsstufe berechnet. Hierfür wird ein zusätzlicher Loss berechnet (siehe dazu Abschnitt 5.6). In Abschnitt 6.3 wird untersucht, inwiefern ein breiterer und tieferer Decoder Auswirkungen auf die Segmentierungsleistung und die Inferenzzeit hat.
Am Ende des Decoders wird mit einer 3×3 Convolution mit 37 Output Feature Maps die Segmentierung berechnet. Der Output-Tensor wird anschließend um den Faktor 4 bilinear hochskaliert, um der Input-Auflösung zu entsprechen.
Beim Testen wird der Output-Tensor auf die Auflösung der Ground-Truth-Segmentierung bilinear skaliert. Erst danach wird das Maximum entlang der Kanalachse bestimmt.

5.2 Der SUN RGB-D Datensatz

Zum Trainieren und Evaluieren des eben vorgestellten RGBD-Segmentierungsnetzwerks ist ein entsprechender Datensatz notwendig. Für die Anwendung auf einem mobilen Roboter im Indoor-Bereich soll der Datensatz aus Indoor-Szenen bestehen. In Abschnitt 5.2.1 wird die Wahl des Datensatzes begründet. Abschnitt 5.2.2 stellt die

unterschiedlichen Kameras vor, mit denen der SUN RGB-D Datensatz aufgenommen wurde. Danach wird in Abschnitt 5.2.3 erklärt, wie Lücken in den Tiefenbilder geschlossen werden. Zuletzt wird in Abschnitt 5.2.4 die Zusammensetzung des Datensatzes beschrieben.

5.2.1 Die Wahl des Datensatzes

Die, in einer Recherche gefundenen, Datensätze werden anhand mehrerer Kriterien bewertet. Darunter: Datensatzgröße, verwendete Kameras und Realitätstreue der Bilder. Die Datensätze *Standford 2D-3D-Semantic* [Armeni et al., 2017] und *Matterport3D* [Chang et al., 2018a] wurden mit einer Matterport Kamera aufgenommen und bestehen somit aus Panoramaansichten. Für den Einsatz auf einem mobilen Roboter mit einer Microsoft Kinect2 Kamera sind sie daher weniger geeignet. Der *Berkeley B3DO*-Datensatz [Janoch et al., 2011] wird durch die vielen Bilder mit unrealistischen Szenenlayout – wie beispielsweise einer Computermaus auf dem Boden – ausgeschlossen. Dahingegen ist beim *SUN3D*-Datensatz [Xiao et al., 2013] die Ground-Truth-Segmentierung schlecht annotiert. Der *NYUv2* Datensatz [Silberman et al., 2012] wurde noch vor wenigen Jahren häufig eingesetzt, limitiert allerdings durch die geringe Anzahl von nur 1449 RGBD-Bildern. Der *SceneNet RGB-D* Datensatz [McCormac et al., 2017] ist zwar mit 5 Millionen Bildern deutlich größer, besteht allerdings aus rein synthetischen Bildern. Mit einem Vortrainieren auf diesem Datensatz könnte jedoch das Vortrainieren auf ImageNet [Russakovsky et al., 2015] ersetzt werden. Aufgrund der genannten Nachteile anderer Datensätze wird der *SUN RGB-D* Datensatz [Song et al., 2015] ausgewählt. Dieser Datensatz wird in den nachfolgenden Abschnitten ausführlich beschrieben. Das Training auf dem SUN RGB-D Datensatz ermöglicht außerdem eine Vergleichbarkeit mit State-of-the-Art-Publikationen.

5.2.2 Verwendete Kameras zur Aufnahme der RGBD-Bilder

Die Bilder des SUN RGB-D Datensatzes wurden mit vier verschiedenen Kameras aufgenommen: Intel Realsense, Asus Xtion und Microsoft Kinect 1 und 2. Die vier Kameras werden in Tabelle 5.2 nach ihrem Aufnahmeprinzip, der Leistungsaufnahme, der Auflösung und der Bildanzahl im Datensatz miteinander verglichen. In Abbildung 5.3 sind in den ersten beiden Spalten Beispielbilder der vier Kameras zu sehen.

	Realsense	Xtion	Kinect1	Kinect2
Aufnahmeprinzip	IR-Muster	IR-Muster	IR-Muster	Time-of-Flight
Leistungsaufnahme	2.5 W USB	2.5 W USB	12.96 W	15 W - 30 W
Tiefenauflösung	468×628	480×640	480×640	424×512
RGB-Auflösung	1080×1920	480×640	480×640	1080×1920
RGBD-Auflösung	583×681	441×591	427×561	530×730
Trainingsbilder	587	1701	1073	1924
Testbilder	572	1688	930	1860

Tabelle 5.2: Quantitativer Vergleich der 4 RGBD-Kameras
IR steht für Infrarot. RGBD-Auflösung ist die Auflösung der registrierten und verbesserten RGBD-Bilder im Datensatz. Die Auflösungen sind in H×W angegeben. Die Werte sind aus [Song et al., 2015] übernommen. Lediglich die Leistungsaufnahme der Kinect2 stammt aus eigenen Messungen.

Nachfolgend werden wichtige Merkmale und Unterschiede der einzelnen Kameras näher erläutert.[7] Das Hintergrundwissen zu den Kameras kann bei der Erklärung möglicher Unterschiede in der Segmentierungsleistung helfen. Die mean Intersection over Union (mIoU) auf den Bildern der verschiedenen Kameras wird in Abschnitt 6.6.2 (Seite 117) untersucht.

Intel Realsense. Die Intel Realsense ist klein, hat nur ein geringes Gewicht und verbraucht wenig Strom, was gerade für den mobilen Bereich Vorteile bringt. Zur Erstellung des Tiefenbilds wird ein strukturiertes Infrarot-Muster in die Umgebung projiziert.[8] Das resultierende Tiefenbild ist allerdings stark verrauscht und liefert nur bis etwa 3.5 Meter zuverlässige Werte. Mithilfe der RGBD-Bilder der Intel Realsense lässt sich überprüfen, wie gut ein Netzwerk mit Bildern schlechter Qualität zurechtkommt.

Asus Xtion und Kinect1. Die Asus Xtion und die Kinect1 nutzen zur Tiefenbilderstellung ein strukturiertes Muster aus dem nahen Infrarotbereich. Die Asus Xtion ist leichter und benötigt zur Stromversorgung lediglich eine USB-Verbindung. Allerdings ist die RGB-Bildqualität schlechter als bei der Kinect1. Dahingegen benötigt die

[7] Weiterführende Informationen zu den Aufnahmeprinzipen und Kameras sind in [Gross, 2019] zu finden.
[8] Die Entstehung von Tiefenbildern wird in Abschnitt 2.4.1 erklärt.

Kinect1 eine externe Stromversorgung. In den Tiefenbildern beider Kameras ist ein Quantisierungseffekt sichtbar.

Kinect2. Die Kinect2 basiert auf dem Time-of-Flight-Prinzip, wodurch die Tiefenbilder exaktere Werte enthalten. Vor allem feine Tiefendifferenzen werden genauer abgebildet, als bei den anderen Kameras. Andererseits versagt die Kinect2 häufiger bei schwarzen Objekten und leicht reflektierenden Oberflächen. Des Weiteren hat die Kinect2 den mit höchsten Stromverbrauch.

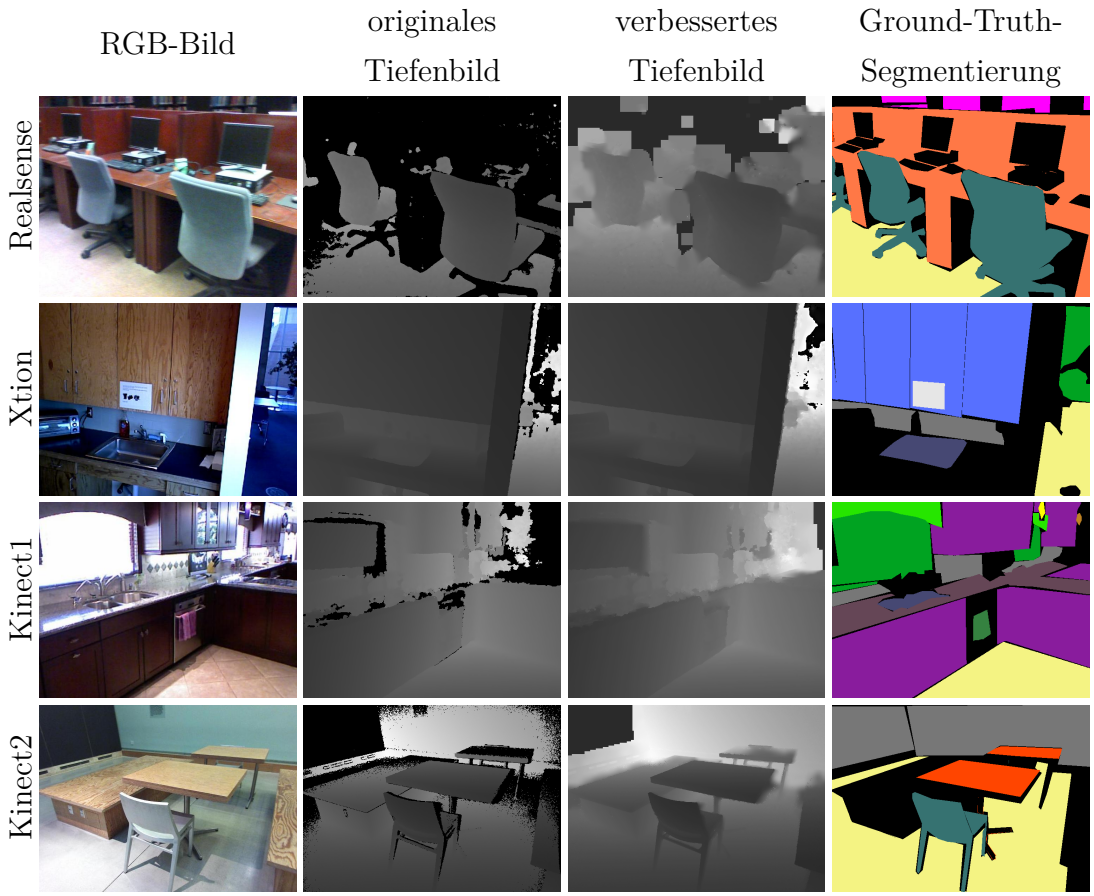


Abbildung 5.3: Beispielbilder des SUN RGB-D Datensatzes
Jede Kamera nimmt ein RGB- und ein Tiefenbild auf. Mithilfe zeitlich benachbarter Frames können Lücken in den originalen Tiefenbildern geschlossen werden. Die schlechte Qualität der Realsense-Tiefenbilder ist deutlich erkennbar. Schwarze Bereich in der Ground-Truth-Segmentierung gehören der Void-Klasse an. In Abbildung 6.29 (Seite 129) sind die Farben der Segmentierung den einzelnen Klassen zugeordnet.

5.2.3 Verbesserung der Tiefenbilder

Um die Qualität der Tiefenbilder zu verbessern, wurden kurze Videos aufgenommen. Mithilfe zeitlich benachbarter Frames können Lücken in den Tiefenbildern geschlossen werden: Dafür werden die Frames zuerst in den 3D-Raum projiziert. Anschließend werden jeweils die 3D-Rotation und -Translation zwischen dem aktuellen Frame und dem Ziel-Frame berechnet. Die Frames werden an das Ziel-Frame ausgerichtet und zurück in den 2D-Raum projiziert. Für jedes Pixel wird nun der Median und die 25% und 75% Perzentile des Tiefenwerts berechnet. Wenn der Tiefenwert des Ziel-Frames fehlt oder außerhalb des 25% - 75% Bereichs liegt, wird der Median aus mindestens 10 benachbarten Frames übernommen. Ansonsten wird der originale Tiefenwert beibehalten. Für diese Art der Tiefenbildverbesserung ist eine genaue Schätzung der 3D-Transformation essentiell. Hierfür nutzen die Autoren eine Kombination der Algorithmen SIFT[9], RANSAC[10] und ICP[11]. Beispiele für das originale Tiefenbild und das verbesserte Tiefenbild sind in Abbildung 5.3 in Spalte 2 und 3 zu sehen. Dabei fällt auf, dass die „Verbesserung“ des Tiefenbilds nicht immer zum Vorteil ist. Vor allem bei den Realsense-Bildern leidet die Qualität des Tiefenbilds oftmals unter dieser „Verbesserung“.

Wie im RedNet [Jiang et al., 2018] und im ACNet [Hu et al., 2019] werden in dieser Masterarbeit für das Training und zum Testen des Netzwerks zunächst die verbesserten Tiefenbilder verwendet. In Abschnitt 6.6.5 wird untersucht, welche Segmentierungsleistung bei Verwendung der originalen Tiefenbilder erzielt werden kann. Unabhängig davon stehen in der Anwendung lediglich die originalen Tiefenbilder zur Verfügung, weshalb hierfür ein Training mit den originalen Tiefenbildern sinnvoller sein kann.

5.2.4 Zusammensetzung des Datensatzes

Der SUN RGB-D Datensatz enthält alle 1449 Bilder aus dem NYUv2-Datensatz und 554 manuell ausgewählte Bilder aus dem Berkeley B3DO-Datensatz mit realistischen Szenenlayout.[12] Beide Datensätze wurden mit einer Microsoft Kinect1 aufgenommen.

[9] SIFT (Scale Invariant Feature Transformation) ist ein Feature-Detektor und -Deskriptor. [Lowe, 1999]

[10] Der RANSAC-Algorithmus (Random Sample Consensus) ermöglicht es, ein Modell zu schätzen innerhalb von Messwerten mit Ausreißern. [Fischler und Bolles, 1981]

[11] Im Iterative Closest Point Algorithmus werden Punktwolken aneinander angepasst. [Rusinkiewicz und Levoy, 2001]

[12] Auch für die Bilder aus dem NYUv2- und dem B3DO-Datensatz sind verbesserte Tiefenbilder vorhanden. Aus dem SUN RGB-D Paper [Song et al., 2015] geht allerdings nicht hervor, wie diese berechnet wurden und ob oder woher die Autoren die notwendigen Videosequenzen haben.

Zusätzlich wurden 3389 manuell ausgewählte Bilder aus dem SUN3D-Datensatz – aufgenommen mit einer Asus Xtion – dem Datensatz hinzugefügt. Des Weiteren enthält der Datensatz neu aufgenommene Bilder: davon 3784 mit der Microsoft Kinect2 und 1159 mit der Intel Realsense. Insgesamt ergeben sich dadurch 10335 RGBD-Bilder aus dem Indoor-Bereich, davon 5285 für das Training und 5050 zum Testen. In Tabelle 5.2 ist die Anzahl an Trainings- und Testbildern pro Kamera aufgeführt. Ein Validierungsdatensatz ist nicht vorhanden.

Bis auf die Bilder des NYUv2-Datensatzes wurden alle Bilder neu annotiert. Für die semantische Segmentierung sind 37 Klassen und eine Void-Klasse vorhanden. Als Void werden dabei alle Pixel gelabelt, die keiner der anderen 37 Klassen zuzuordnen sind. Beispiele für eingefärbte Ground-Truth-Segmentierungen sind in Abbildung 5.3 in der letzten Spalte zu sehen. Wie in Abbildung 5.4 zu erkennen ist, ist die Klassenverteilung im SUN RGB-D Datensatz stark unbalanciert.

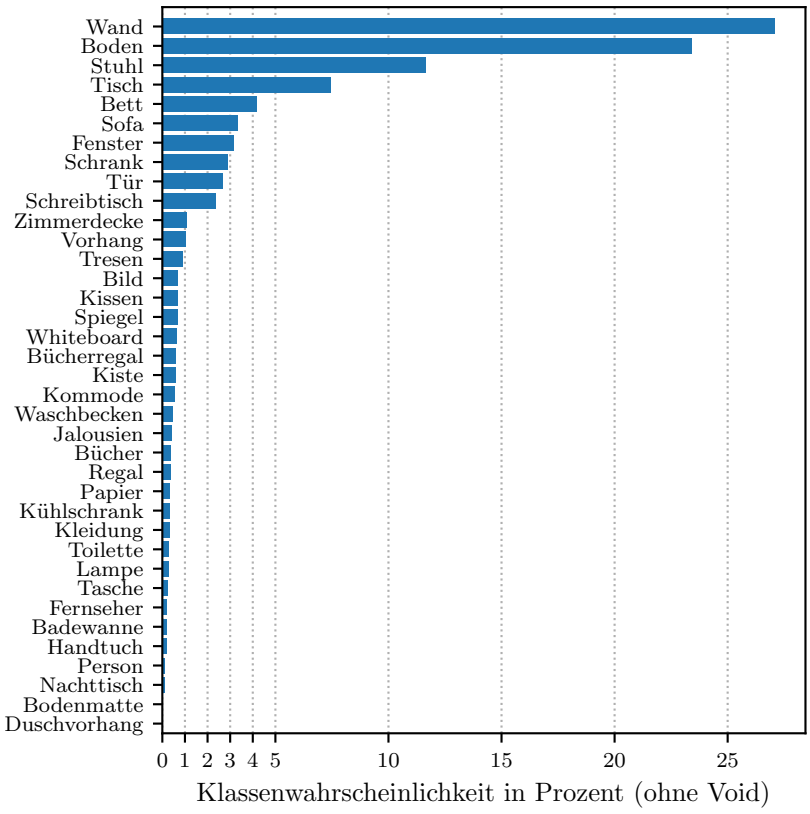


Abbildung 5.4: Klassenverteilung im SUN RGB-D Datensatz

Klassenverteilung im SUN RGB-D Trainingsdatensatz ohne die Void-Klasse. Die Klassenverteilung des Testdatensatzes pro Kamera ist in Abbildung A.22 dargestellt.

Die Void-Klasse macht 25% der Pixel aus. Von den verbleibenden Pixeln sind über 60% den drei Klassen Wand, Boden und Stuhl zuzuordnen. 25 Klassen ist jeweils weniger als 1% der Nicht-Void-Pixel zuzuordnen. Aufgrund der starken Unbalanciertheit ist eine Klassengewichtung während des Trainings essentiell.[13]

5.3 Datenvorverarbeitung

Die Datenvorverarbeitung für die Netzwerkeingabe entspricht – bis auf kleine Änderungen – der Datenvorverarbeitung des RedNets [Jiang et al., 2018]. Die darin enthaltene Augmentierung der Trainingsdaten wirkt einem Auswendiglernen entgegen. Für die Netzwerkeingabe müssen alle Bilder der Input-Auflösung $H \times W = 480 \times 640$ entsprechen. Nachfolgend wird die Datenvorverarbeitung der Trainings- und Testdaten erklärt.

Trainingsdaten. Die Trainingsdaten werden zuerst zufällig um einen Faktor aus dem Bereich $[1.0, 1.4]$ skaliert.[14] Dabei werden die RGB-Bilder bilinear skaliert. Für die Tiefenbilder und die Ground-Truth-Segmentierung wird die Nearest-Neighbor-Interpolation angewandt. Aus den skalierten Bildern wird anschließend ein zufälliger Bereich der Zielgröße $H \times W = 480 \times 640$ herausgeschnitten. Bei den Bildern der Asus Xtion und der Kinect1 kann es dazu kommen, dass die skalierten Bilder zu klein sind und kein entsprechend großer Bereich ausgeschnitten werden kann (siehe dazu Tabelle 5.2). In diesem Fall werden die skalierten Bilder direkt auf die Zielgröße skaliert. Da die Bilder der beiden Kameras nahezu ein Seitenverhältnis von $H{:}W = 3{:}4$ haben, kommt es durch diese direkte Skalierung auf die Zielgröße nur zu minimalen Verzerrungen.[15] Nach dem Skalieren und Zuschneiden werden die Bilder zufällig horizontal gespiegelt. Die RGB-Bilder werden zusätzlich über zufällige Veränderung des Farbwerts, der Helligkeit und der Sättigung augmentiert. Zuletzt werden die RGB- und Tiefenbilder mittels Mittelwert und Standardabweichung normalisiert.

[13] Die verwendete Klassengewichtung wird in Abschnitt 5.4 erklärt.

[14] Ein kleinerer Faktor von beispielsweise 0.9 würde dazu führen, dass die resultierende Auflösung kleiner als die Zielgröße $H \times W = 480 \times 640$ wird (siehe dazu Tabelle 5.2).

[15] Im Gegensatz zum eben beschriebenen Skalieren und Zuschneiden werden die Bilder im RedNet zuerst auf die gewünschte Zielgröße $H \times W = 480 \times 640$ skaliert und erst anschließend um einen Faktor $[1.0, 1.4]$ hochskaliert und danach zugeschnitten. Das hat den Nachteil, dass bei den Realsense- und Kinect2-Bildern durch das Herunterskalieren Pixelinformationen verlorengehen, die beim anschließenden Hochskalieren nicht wieder hergestellt werden können.

Testdaten. Die Testdaten werden direkt auf die Zielgröße $H \times W = 480 \times 640$ skaliert. Anschließend werden RGB- und Tiefenbilder mit denselben Mittelwerten und Standardabweichungen der Trainingsdaten normalisiert.

Wie in Abschnitt 4.4 begründet, verzichtet diese Masterarbeit auf die HHA-Codierung der Tiefenbilder.

5.4 Klassengewichtung

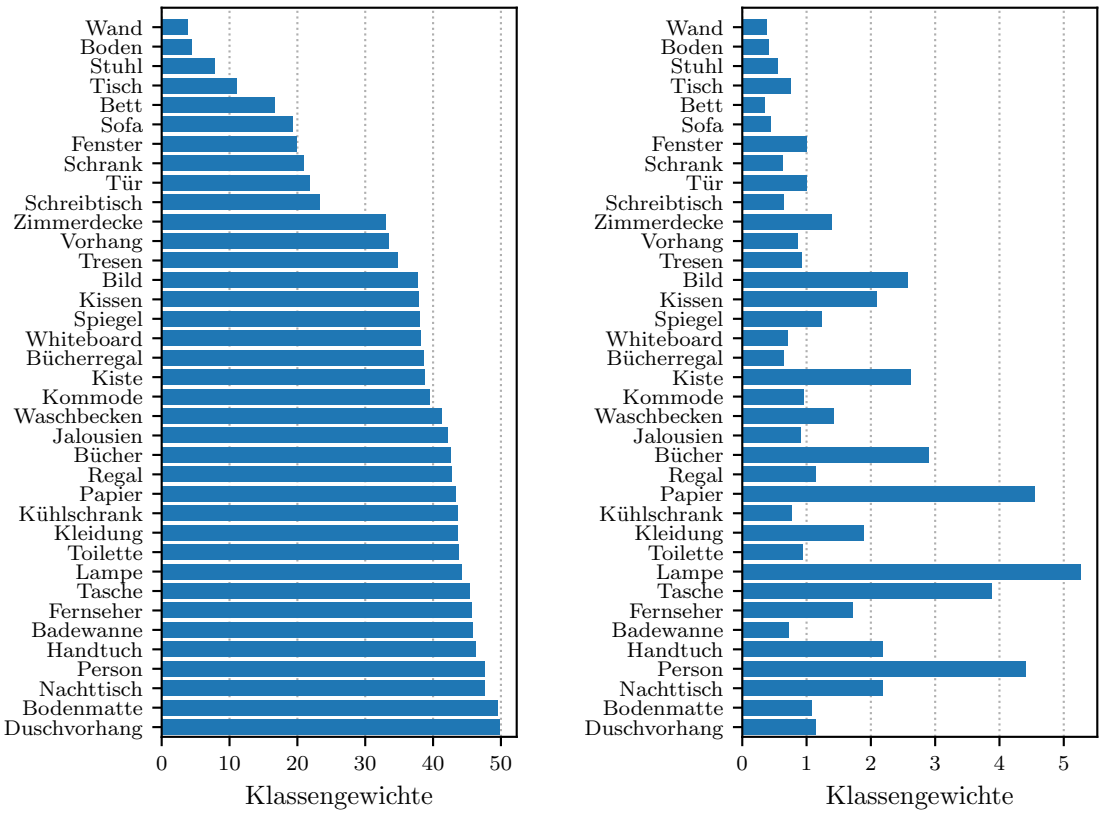
Die Klassenverteilung des SUN RGB-D Datensatze ist stark unbalanciert. Ohne eine Klassengewichtung würde das Deep-Learning-Netzwerk hauptsächlich häufig vorkommende Klassen prädizieren und unterrepräsentierte Klassen vernachlässigen. Aus diesem Grund werden in der Lossfunktion seltene Klassen stärker gewichtet als häufig vorkommende Klassen.
Bekannte Klassengewichtungen sind die *logarithmische Klassengewichtung* nach [Paszke et al., 2016] und das *Median Frequency Balancing* aus [Eigen und Fergus, 2015]. Die logarithmische Klassengewichtung $\omega_{c,log}$ ist für die Klasse c folgendermaßen definiert:

$$\omega_{c,log} = \frac{1}{ln\,(1.02 + p_c)} \tag{5.1}$$

Dabei gibt p_c die Klassenwahrscheinlichkeit der Klasse c an. Durch die Addition von 1.02 werden die Klassengewichte nach oben begrenzt. Diese Beschränkung ist vor allem für sehr seltene Klassen sinnvoll.
Beim Median Frequency Balancing ergibt sich das Klassengewicht $\omega_{c,mfb}$ nach folgender Berechnung:

$$\omega_{c,mfb} = \frac{median\_freq}{freq_c} \tag{5.2}$$

Hierbei steht $freq_c$ für die Pixelanzahl der Klasse c geteilt durch die Gesamtanzahl an Bildern, in denen die Klasse vorkommt. *median_freq* ist der Median dieser Frequenzen über alle Klassen.
Die resultierenden Klassengewichte für den SUN RGB-D Trainingsdatensatz sind in Abbildung 5.5 dargestellt. Bei der logarithmischen Klassengewichtung sind die Klassengewichte höher, je geringer die Klassenwahrscheinlichkeit ist.



(a) logarithmische Klassengewichtung **(b)** Median Frequency Balancing

Abbildung 5.5: Klassengewichtung

Klassengewichte sortiert nach Klassenwahrscheinlichkeit. Je häufiger eine Klasse vorkommt, desto weiter oben befindet sie sich im Diagramm. (Die Klassenverteilung ist in Abbildung 5.4 dargestellt.)

Beim Median Frequency Balancing spielt außerdem die Anzahl an Bildern mit dieser Klasse eine Rolle. Die Klasse Duschvorhang hat beispielsweise eine geringere Klassenwahrscheinlichkeit als die Klasse Lampe, kommt allerdings in weniger Bildern vor. Daher ist das Klassengewicht für die Klasse Duschvorhang geringer.

In Vorexperimenten konnten mit beiden Klassengewichtungen ähnliche mIoU-Werte erreicht werden. Da in den Implementierungen des RedNets [Jiang et al., 2018] und des ACNets [Hu et al., 2019] Median Frequency Balancing zur Klassengewichtung verwendet wird, wird auch in den Experimenten dieser Masterarbeit das Median Frequency Balancing eingesetzt.

5.5 Loss-Funktion

Als Loss-Funktion wird die Kreuzentropie verwendet. Pixel, die in der Ground-Truth-Segmentierung als Void gekennzeichnet sind, werden dabei ausmaskiert. Der Loss aller anderen Pixel wird mit den berechneten Klassengewichten multipliziert und aufsummiert. In der Implementierung des RedNets [Jiang et al., 2018] und ACNets [Hu et al., 2019] wird der aufsummierte Loss schließlich durch die Anzahl der Nicht-Void-Pixel dividiert. Wie in Abbildung 5.5 zu erkennen ist, sind die Klassengewichte bei der logarithmischen Klassengewichtung allerdings deutlich höher als beim Median Frequency Balancing. Dies würde in einem insgesamt höheren Loss resultieren und dadurch das Training beeinflussen. Um beide Klassengewichtungen gegeneinander testen zu können, wird der aufsummierte Loss durch die, mit den Klassengewichten, gewichtete Anzahl der Nicht-Void-Pixel dividiert.

5.6 Multi Scale Supervision

In Anlehnung an das RedNet werden mehrere Loss-Werte unterschiedlicher Auflösungsstufen berechnet. Hierfür enthält der Decoder zusätzliche Neben-Outputs, welche nur während des Trainings berechnet werden. Diese Neben-Outputs sind in Abbildung 5.1 dargestellt. Vor jedem bilinearen Upsampling im Decoder wird als Nebenzweig eine 1×1 Convolution angewandt, welche eine Segmentierung geringerer Auflösung berechnen soll. Insgesamt enthält das Netzwerk drei Neben-Outputs unterschiedlicher Auflösungen. Der erste Neben-Output ist um $\frac{1}{32}$ kleiner als der Input und hat somit eine Auflösung von $H \times W = 15 \times 20$. Der zweite Neben-Output ist um $\frac{1}{16}$ kleiner und hat die Auflösung 30×40. Der dritte Neben-Output entspricht $\frac{1}{8}$ der Input-Auflösung mit 60×80. Da das Netzwerk ab einer Auflösung von $\frac{1}{4}$ nur noch bilinear hochskaliert, existieren keine weiteren Neben-Outputs. Über Nearest-Neighbor-Interpolation wird die Ground-Truth-Segmentierung auf die Größe der Neben-Outputs herunterskaliert. Zusätzlich zum Loss der Netzwerkausgabe wird ein Loss für jeden Neben-Output bestimmt. Der gesamte Loss ist die Addition aller vier Losse. Hierbei ist zu beachten, dass die vier Losse zuerst einzeln berechnet und erst danach addiert werden. Das hat zur Folge, dass der Loss eines Pixels bei einem Neben-Output geringer Auflösung implizit stärker gewichtet wird als bei Outputs größerer Auflösung.

Laut [Jiang et al., 2018] werden dadurch bessere Ergebnisse erzielt als bei einer gleichmäßigen Gewichtung.
Durch diese Multi Scale Supervision wird das Netzwerk dazu gezwungen bereits am Anfang des Decoders eine gute Segmentierung zu berechnen, welche nun lediglich mit Encoder Feature Maps höherer Auflösung verfeinert wird. Des Weiteren erhalten somit die beiden Encoder und die ersten Layer des Decoders den vierfachen Gradienten, wodurch das Training beschleunigt wird.
In Vorexperimenten zeigte sich, dass mithilfe der Multi Scale Supervision höhere mIoU-Werte erreicht werden können.

5.7 Hyperparameter und Implementierungsdetails

Das Netzwerk und das Training wird in PyTorch [Paszke et al., 2019] mit Python[16] implementiert. Die Implementierung dieser Masterarbeit baut auf der Implementierung des RedNets[17] auf.
Ingesamt wird das Netzwerk für maximal 500 Epochen trainiert. Nach jeder Epoche wird die mIoU auf den Testdaten bestimmt. Für ein faires Training wäre ein Validierungsdatensatz notwendig. Allerdings besteht der SUN RGB-D Datensatz lediglich aus Trainings- und Testdaten. In Vorexperimenten wurden 10% der Trainingsdaten abgespalten und zur Validierung verwendet. Jedoch verschlechterten sich dadurch die erzielten Ergebnisse auf den Testdaten. Des Weiteren wurde in keiner der recherchierten Publikationen ein Validierungsdatensatz erwähnt. Für einen fairen Vergleich mit dem State of the Art wird daher auch in dieser Masterarbeit auf einen Validierungsdatensatz verzichtet.

Optimierer. Als Optimierer wird SGD mit einem Nesterov-Momentum von 0.9 verwendet. [Sutskever et al., 2013] In Vorexperimenten erzielte SGD mit normalem Momentum etwas schlechtere Ergebnisse. Mit den Optimierern RAdam [Liu et al., 2019] und RMSprop [Hinton, 2012] schwankten Loss und mIoU sehr stark und das Training wurde instabil. Erst mit sehr kleinen Lernraten war überhaupt ein sinnvolles Training möglich. Die erzielten Ergebnisse waren dennoch schlechter als mit SGD und Nesterov-Momentum.

[16] `https://www.python.org/` – Abgerufen am 12.02.2020
[17] `https://github.com/JindongJiang/RedNet` – Abgerufen am 12.02.2020

Lernraten. Die mittels Vorexperimenten festgelegte Basislernrate von 0.0025 wird einmal halbiert und einmal verdoppelt. Da einige Netzwerke bei einer Lernrate von 0.005 die besten Ergebnisse erzielen, wird die Lernrate nochmals verdoppelt. Insgesamt ergeben sich somit vier Lernraten: 0.00125, 0.0025, 0.005 und 0.01. Eine weitere Verdopplung der Lernrate wird ausgeschlossen, da mIoU und Loss bereits bei einer Lernrate von 0.01 stark schwanken.

Lernraten-Scheduling. Für das Training wird der Lernraten-Scheduler OneCyle aus PyTorch verwendet. Dabei steigt die Lernrate für 50 Epochen an und erreicht dann ihr Maximum. Die Startlernrate beträgt $\frac{1}{25}$ der maximalen Lernrate. Anschließend fällt die Lernrate für 450 Epochen auf $\frac{1}{1000}$ der maximalen Lernrate ab. Der Verlauf der Lernrate ist in Abbildung 5.6 dargestellt.

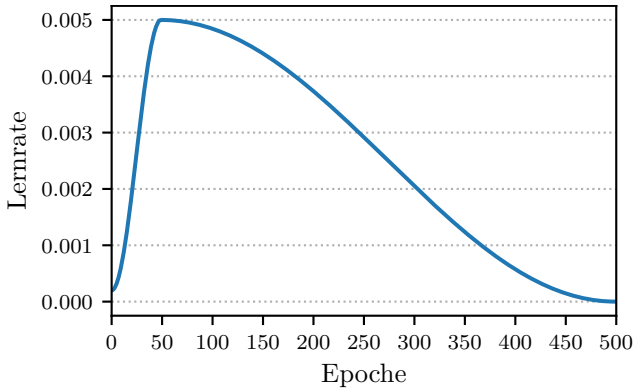


Abbildung 5.6: Lernraten-Scheduler
OneCycle-Lernraten-Scheduler mit 0.005 als maximale Lernrate.

Dieses Lernraten-Scheduling hat folgende Vorteile: Das Training wird mit einer niedrigen Lernrate begonnen. Dadurch divergieren die Gewichte der auf ImageNet vortrainierten Encoder nicht so schnell von ihrem erzielten Minimum. Die Gewichte des Decoders werden sich zunächst in der Nähe der zufälligen Initialisierung an ein lokales Minimum annähern. Anschließend wird die Lernrate erhöht, wodurch kleinere Minima gefunden werden können. Die hohe Lernrate in der Mitte des Trainings wirkt regularisierend und verhindert ein Overfitting auf den Trainingsdaten. Gegen Ende des Trainings können mit der sehr kleinen Lernrate steilere lokale Minima erreicht werden.[18]

[18] Weitere Informationen zum OneCycle-Lernraten-Scheduling sind in [Smith, 2018] und `https://sgugger.github.io/the-1cycle-policy.html` – Abgerufen am 12.02.2020 zu finden.

Batchsize. Um Experimente mit verschieden komplexen Netzwerkstrukturen mit derselben Batchsize durchführen zu können, wurde eine Batchsize von 8 gewählt. Während des Training werden die Batches zufällig zusammengestellt. Ein Batch kann also Bilder verschiedener Kameras enthalten. Das letzte, kleinere Batch jeder Epoche wird verworfen und nicht mit trainiert.

mIoU als Bewertungsmaß. Zur Bewertung der Segmentierungsleistung wird die *mean Intersection over Union (mIoU)* verwendet. Die prädizierte Segmentierung des Netzwerks wird hierfür zuerst auf die Auflösung der Ground-Truth-Segmentierung bilinear skaliert. Diese Skalierung erfolgt vor der Berechnung des Maximums entlang der Kanalachse. Dadurch können unerwünschte Blockartefakte in der Segmentierung vermieden werden. Aus der prädizierten Segmentierung und der Ground-Truth-Segmentierung wird anschließend eine Konfusionsmatrix berechnet. Die als Void markierten Pixel werden hierbei ignoriert. Danach wird die Intersection over Union (IoU) jeder Klasse berechnet. Die IoU ist in [Hazirbas et al., 2016] definiert als $\mathrm{IoU} = \mathrm{TP}/\left(\mathrm{TP} + \mathrm{FP} + \mathrm{FN}\right)$. TP, FP und FN steht dabei entsprechend für die Pixelanzahl an True Positives, False Positives und False Negatives. Die mIoU ist die Mittlung der IoUs aller Klassen.
Die Konfusionsmatrizen und die mIoU werden zunächst für jede Kamera einzeln bestimmt. Dadurch wird es möglich die prädizierte Segmentierung batchweise auf die Auflösung der Ground-Truth-Segmentierung zu skalieren. Des Weiteren können so die erzielten Ergebnisse auf den einzelnen Kameras miteinander verglichen werden. Im Anschluss werden die Konfusionsmatrizen der vier Kameras addiert und die mIoU auf dem gesamten Testdatensatz bestimmt.

Early Stopping. Das Training wird vorzeitig beendet, sobald sich über 100 Epochen keine weitere Verbesserung der mIoU ergibt.

Trainingsdurchläufe. Alle Konfigurationen werden – soweit nicht anders angegeben – zweimal trainiert. Dadurch kann eine bessere Aussage getroffen werden, ob ein besonders hoher oder geringer mIoU-Wert nur ein Ausreiser ist, oder tatsächlich auf die Netzwerkarchitektur zurückzuführen ist. Weitere Trainingsdurchläufe würden diese Aussage nochmals verbessern. Da jedoch ein Training – je nach Komplexität der Netzwerkarchitektur – zwei bis drei Tage in Anspruch nimmt, wird jede Konfiguration lediglich zweimal trainiert.

5.8 Inferenzzeitmessung

Für den Einsatz auf einem mobilen Roboter ist eine geringe Inferenzzeit notwendig. Daher werden im Rahmen dieser Masterarbeit Inferenzzeiten für die entwickelten Architekturen gemessen. Als Hardware kommt hierfür ein NVIDIA Jetson AGX Xavier[19] zum Einsatz. Das trainierte PyTorch-Netzwerk wird mithilfe von TensorRT[20] für die Inferenz optimiert. Hierfür muss das PyTorch-Modell zuerst in das offene Format ONNX[21] umgewandelt werden. Vom ONNX-Format kann anschließend eine TensorRT-Engine gebaut werden. Eine noch geringere Inferenzzeit kann mit Float16 erreicht werden. Der Einfluss auf die mIoU ist hierfür allerdings noch zu untersuchen. Sofern nicht anders deklariert, sind die in Kapitel 6 angegebenen Inferenzzeiten mit TensorRT Float32 gemessen.

[19] `https://www.nvidia.com/de-de/autonomous-machines/embedded-systems/jetson-agx-xavier/` – Abgerufen am 13.02.2020
[20] `https://developer.nvidia.com/tensorrt` – Abgerufen am 13.02.2020
[21] `https://onnx.ai/` – Abgerufen am 13.02.2020

Kapitel 6

Experimentelle Untersuchungen

In diesem Kapitel soll die, in Abschnitt 5.1, vorgestellte Architektur (siehe Abbildung 5.1) experimentell untersucht und weiterentwickelt werden. Der Fokus liegt hier auf einer möglichst hohen Segmentierungsleistung und einer geringen Inferenzzeit. Zuerst wird in Abschnitt 6.1 der Einfluss der Netzwerktiefe und der Modalität untersucht. Anschließend wird in Abschnitt 6.2 der ResNet-Non-Bottleneck-Block durch andere, effiziente Encoder-Blöcke ersetzt. Nach den Experimenten zum Encoder wird in Abschnitt 6.3 der Decoder variiert. Danach werden in Abschnitt 6.4 verschiedene Möglichkeiten zur Fusion von RGB- und Tiefen-Features getestet. Anschließend wird in Abschnitt 6.5 der Einfluss des Kontextmoduls untersucht. Zuletzt wird in Abschnitt 6.6 das beste Netzwerk näher betrachtet.

6.1 Netzwerktiefe vs. Modalität

Zuerst soll der Einfluss der Netzwerktiefe auf die mean Intersection over Union (mIoU) und die Inferenzzeit untersucht werden. Die vorgestellte Netzwerkarchitektur in Abschnitt 5.1 verwendet für die beiden Encoder jeweils ein ResNet18. Das eher flache ResNet18 (●) wird hier durch das tiefere ResNet34 (●) bzw. ResNet50 (●) ersetzt. Das ResNet34 verwendet wie das ResNet18 den ResNet-Non-Bottleneck-Block. Im ResNet50 kommt der ResNet-Bottleneck-Block zum Einsatz.[1] Anstelle von jeweils 2 Blöcken pro Auflösungsstufe werden im ResNet34 und im ResNet50 für die vier Auflösungsstufen entsprechend 3, 4, 6 und 3 Blöcke verwendet.

[1] Beide ResNet-Blöcke sind in Abschnitt 2.1 (Seite 6) erklärt.

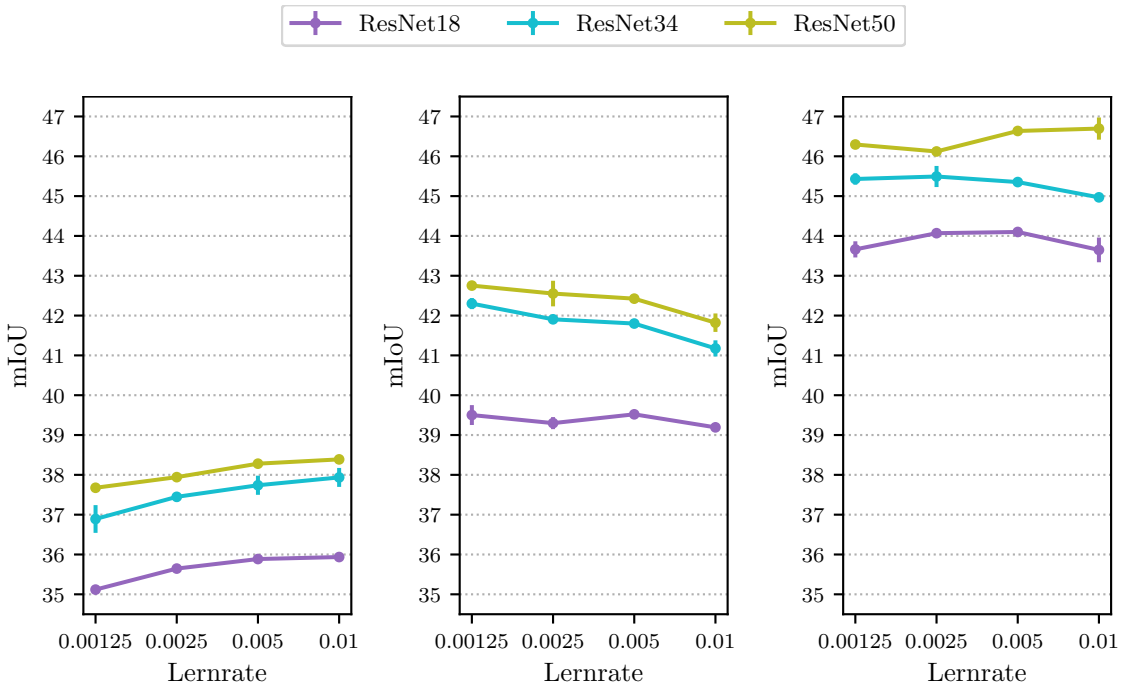
Ein Netzwerk mit zwei Encodern für RGB- und Tiefenbilder ist prinzipiell rechenaufwendiger als ein Netzwerk mit nur einem einzigen Encoder für RGB-Bilder. Daher soll zusätzlich der Einfluss verschiedener Input-Modalitäten untersucht werden. Aus der vorgestellten Netzwerkarchitektur wird hierfür eine unimodale Netzwerkarchitektur mit nur einem Encoder abgeleitet: Alle Module zur Fusion von RGB- und Tiefen-Features entfallen. Um eine möglichst ähnliche Architektur zu erhalten, werden die Encoder Features auch hier in den Decoder konkateniert. Dafür wird die Kanalanzahl der Encoder Features mit einer 1×1 Convolution auf $C_{out} = 24$ verringert. Vor der Konkatenation mit dem Decoder wird nun noch Batch Normalization angewandt. Die restliche Netzwerkarchitektur ist identisch zur multimodalen RGBD-Netzwerkarchitektur, welche in Abbildung 5.1 dargestellt ist. Diese unimodale Netzwerkarchitektur wird einmal auf lediglich RGB-Bildern und einmal auf lediglich Tiefenbildern trainiert. Insgesamt ergeben sich somit drei Modalitäten: Tiefe, RGB und RGBD.

6.1.1 Baseline-Experimente

Jede Modalitäts-Konfiguration wird mit allen drei Encodern und den vier verschiedenen Lernraten trainiert.[2] Alle Konfigurationen werden zweimal trainiert. Insgesamt ergeben sich dadurch $3 \cdot 3 \cdot 4 \cdot 2 = 72$ Trainingsdurchläufe.

In Abbildung 6.1 ist die erzielte mIoU für die verschiedenen Encoder und Modalitäten nach der Lernrate dargestellt. In allen drei Modalitäts-Konfigurationen werden mit tieferen Netzwerken höhere mIoU-Werte erzielt. Bei den unimodalen Netzwerken ist der Unterschied zwischen einem ResNet18- (●) und einem ResNet34-Encoder (●) höher als zwischen einem ResNet34- und einem ResNet50-Encoder (●). Es fällt auf, dass Tiefen-Netzwerke bei höheren Lernraten besser abschneiden. Das deutet darauf hin, dass der auf ImageNet-vortrainierte Encoder stärker umlernen muss. Da ImageNet aus RGB-Bildern besteht, müssen die Gewicht der RGB-Netzwerke nicht so stark adaptiert werden und erzielen bei niedrigeren Lernraten bessere Ergebnisse. Bei den multimodalen RGBD-Netzwerken werden mit mittleren Lernraten die höchsten mIoU-Werte erzielt.

[2] Die Hyperparameter zum Trainieren der Netzwerke sind in Abschnitt 5.7 erklärt.



(a) Tiefe **(b)** RGB **(c)** RGBD

Abbildung 6.1: Modalitäten und Netzwerktiefe nach Lernrate – mIoU
Darstellung der mittleren mIoU und der Standardabweichung je nach Lernrate für verschiedene Encoder (ResNet18, ResNet34 und ResNet50) und verschiedene Modalitäten (nur Tiefe, nur RGB und RGBD). In Tabelle A.1 (Seite 157) sind die exakten Werte enthalten.

Abbildung 6.2 zeigt für alle Modalitäten und Encoder die mIoU der jeweils besten Lernrate. Jede Modalität wird mit einem tieferen Netzwerk besser. Insgesamt sind die RGBD-Netzwerke (●) deutlich besser als die beiden unimodalen Netzwerke. Wie erwartet, erreichen die unimodalen RGB-Netzwerke (●) eine höhere mIoU als die Tiefen-Netzwerke (●). Daraus lässt sich schließen, dass für die semantische Segmentierung RGB-Bilder deskriptivere Informationen enthalten als Tiefenbilder. Dennoch können die Tiefenbilder zusätzliche Informationen liefern, mit denen sich die Segmentierungsleistung nochmals deutlich verbessert.
Es fällt zudem auf, dass mit den multimodalen RGBD-ResNet18-Netzwerken bessere Ergebnisse erzielt werden können als mit den unimodalen RGB-ResNet50-Netzwerken. Bei Betrachtung der Verarbeitungsfrequenzen in Abbildung 6.3 lässt sich feststellen, dass sich der zusätzliche Rechenaufwand für zwei Encoder lohnt.

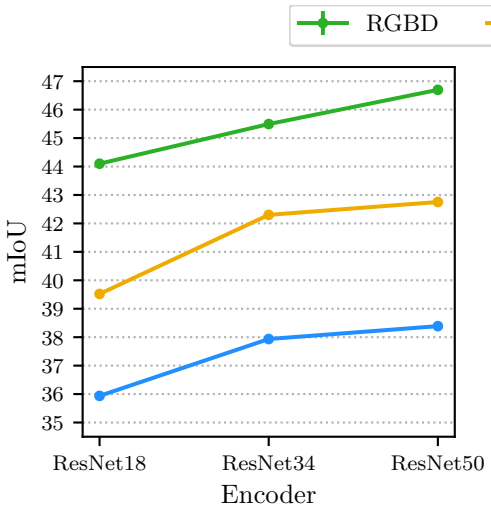


Abbildung 6.2:
Netzwerktiefe vs. Modalität - mIoU
Dargestellt ist die Verbesserung jeder Modalität mit tieferem Encoder.

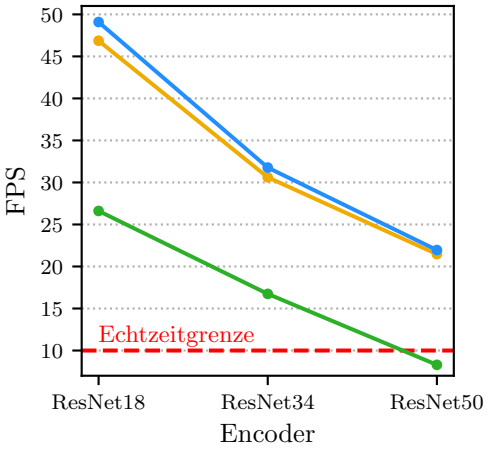


Abbildung 6.3:
Netzwerktiefe vs. Modalität - FPS
Verarbeitungsfrequenzen auf einem Jetson Xavier. Das Modell wurde für die Inferenzzeitmessung in eine TensorRT-Engine mit Float32 umgewandelt. Verarbeitungsfrequenzen für PyTorch und TensorRT Float16 sind in Tabelle A.2 enthalten.

Während die unimodale RGB-ResNet50-Netzwerkarchitektur nur 21.5 FPS erreicht, erreicht die multimodale RGBD-ResNet18-Netzwerkarchitektur bereits 26.6 FPS. Dennoch erzielt das multimodale RGBD-ResNet18-Netzwerk bessere Ergebnisse als das unimodale RGB-ResNet50-Netzwerk.

Die Tiefen-Netzwerkarchitekturen erreichen jeweils eine minimal höhere Verarbeitungsfrequenz als die RGB-Netzwerkarchitekturen. Dies ist ausschließlich auf die erste 7×7 Convolution mit nur einem – anstelle von drei – Input-Kanälen zurückzuführen. Bei tieferen Netzwerkarchitekturen ist der Einfluss dieser ersten Convolution geringer, wodurch der Unterschied zwischen den Verarbeitungsfrequenzen abnimmt.

Da die RGBD-ResNet50-Netzwerkarchitektur trotz Optimierung mit TensorRT unter der, in dieser Masterarbeit definierten, Echtzeitgrenze von 10 FPS liegt, wird diese Netzwerkarchitektur im weiteren Verlauf der Masterarbeit nicht weiter betrachtet und dient lediglich zum Vergleich.

6.1.2 Verschiedene Lernraten für verschiedene Netzwerkteile

Wie in Abbildung 6.1 zu erkennen ist, erzielt das unimodale Tiefen-Netzwerk mit einer hohen Lernrate von 0.01 die besten Ergebnisse. Dahingegen erreicht das RGB-Netzwerk mit der deutlich niedrigeren Lernrate von 0.00125 die höchste mIoU. Aus diesem Grund soll untersucht werden, ob sich für das multimodale RGBD-Netzwerk eine weitere Verbesserung ergibt, wenn der Tiefen-Encoder mit einer höheren Lernrate als der RGB-Encoder trainiert wird. Hierfür wird als Lernrate für den Tiefen- und den RGB-Encoder entsprechend 0.01 und 0.00125 gewählt. Die RGB-Netzwerke erzielen im Allgemeinen bessere Ergebnisse als die Tiefen-Netzwerke. Deswegen wird für die restlichen Netzwerk-Bestandteile – ab jetzt vereinfacht nur noch *Decoder* genannt – zunächst die niedrige Lernrate des RGB-Encoders gewählt. In einer zweiten Konfiguration wird der Decoder mit der höheren Lernrate des Tiefen-Encoders trainiert. Dies lässt sich folgendermaßen motivieren: Die Gewichte des Decoders sind zufällig initialisiert. Daher sind sie noch weit von einem guten Minimum entfernt und müssen stärker lernen. Die erzielten Ergebnisse mit ResNet18- und ResNet34-Encoder sind in Tabelle 6.1 zu sehen.

	ResNet18 mIoU		**ResNet34 mIoU**	
	$\varnothing$	σ	$\varnothing$	σ
feste Lernrate (siehe Abbildung 6.1c)	44.10	0.01	45.49	0.26
RGB-Encoder und Decoder: lr=0.00125 **Tiefen-Encoder: lr=0.01**	44.36	0.10	45.22	0.01
RGB-Encoder: lr=0.00125 **Tiefen-Encoder und Decoder: lr=0.01**	44.09	0.10	45.53	0.03

Tabelle 6.1: Unterschiedliche Lernraten für verschiedene Netzwerkteile
Für die mIoU ist jeweils der Mittelwert $\varnothing$ und die Standardabweichung σ angegeben. Bei fester Lernrate sind die genannten Ergebnisse jeweils für die beste Lernrate: ResNet18: lr=0.005 und ResNet34: lr=0.0025. Tabelle A.3 (Seite 158) enthält die Werte für die einzelnen Trainings.

In der ersten Zeile sind die bisherigen Ergebnisse für die jeweils beste, feste Lernrate dargestellt. Darunter sind die beiden, eben vorgestellten, Konfigurationen mit verschiedenen Lernraten zu sehen. Insgesamt lässt sich sagen, dass die Auswirkung auf die mIoU in beiden Konfigurationen nur minimal sind. Wird im Decoder die geringere Lernrate verändert, so ist nur mit ResNet18-Encoder eine leichte Verbesserung zu erkennen. Das schlechtere Ergebnis beim ResNet34-Encoders liegt allerdings noch beinahe in der Standardabweichung bei Verwendung einer festen Lernrate. Auch die leichte Verbesserung bei höherer Lernrate im Decoder und ResNet34-Encoder liegt innerhalb der Standardabweichung und ist deshalb zu vernachlässigen. Aus diesem Grund wird im weiteren Verlauf eine feste Lernrate für das gesamte Netzwerk beibehalten.

6.1.3 Pre-Training auf unimodalen Segmentierungsnetzwerken

Die unimodalen Netzwerke haben bereits die Aufgabe der Segmentierung erlernt. Daher ist eine weitere Idee das multimodale RGBD-Netzwerk mit Gewichten aus den unimodalen Segmentierungsnetzwerken zu initialisieren. In einer ersten Variante (▪▪) werden lediglich für die beiden Encoder die Gewichte aus den jeweils besten unimodalen Netzwerken übernommen. In einer zweiten Variante (•••) wird zusätzlich das restliche Netzwerk – soweit möglich – mit den Gewichten des unimodalen RGB-Netzwerks initialisiert.[3] In dieser zweiten Variante werden lediglich die Gewichte für die RGBD-Fusion zufällig initialisiert. Die erzielten Ergebnisse mit ResNet18-Encoder sind in Abbildung 6.4 dargestellt.

Insgesamt kann mit keiner der beiden Varianten ein besseres Ergebnis als mit der Baseline erreicht werden. Vor allem bei kleinen Lernraten liegt die erzielte mIoU deutlich unter der Baseline. Der Trainingsverlauf in Abbildung 6.5a zeigt, dass bei Verwendung einer geringen Lernrate die mIoU beim Netzwerk mit auf Segmentierung vortrainierten Encodern (▪▪) deutlich schneller ansteigt als bei der Baseline (●). Jedoch stagniert die mIoU nach wenigen Epochen. Wahrscheinlich hängen die Gewichte in lokalen Minima fest, weshalb keine besseren Ergebnisse erzielt werden können.

Bei höheren Lernraten nähert sich die maximal erzielte mIoU der mIoU der Baseline an. Im Trainingsverlauf in Abbildung 6.5b ist ersichtlich, dass die mIoU zunächst schnell

[3] Aufgrund der besseren Ergebnisse des RGB-Netzwerks werden für den Decoder nicht die Gewichte des Tiefen-Netzwerks verwendet.

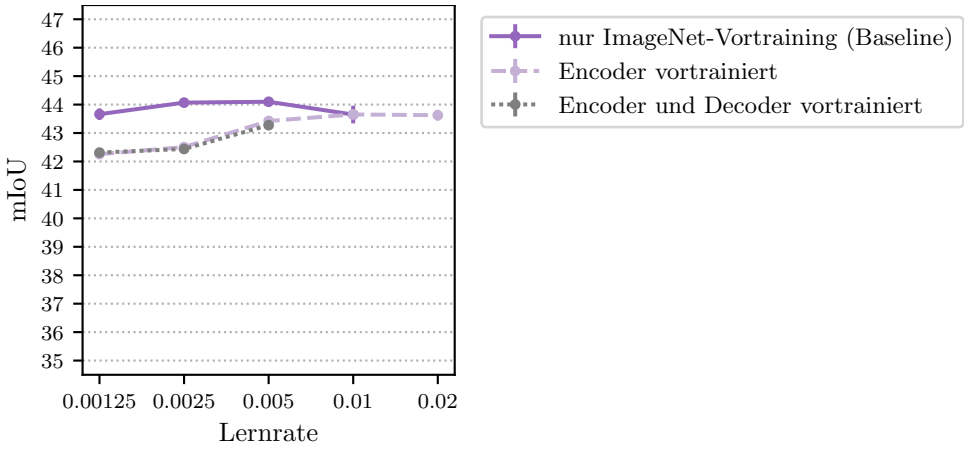
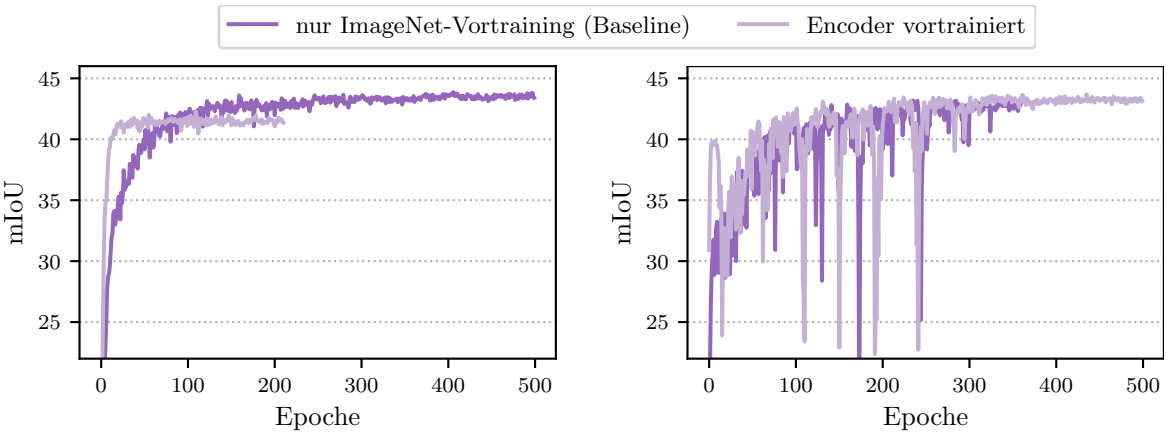


Abbildung 6.4: Pre-Training auf unimodalen Segmentierungsnetzwerken – mIoU
RGBD-ResNet18-Netzwerk unterschiedlich vortrainiert. (●): Beide Encoder wurden lediglich auf ImageNet vortraniert (Baseline). (▪▪): Initialisierung beider Encoder mit Gewichten aus den unimodalen Netzwerken. (•••): Zusätzliche Initialisierung des Decoder soweit möglich mit Gewichten aus dem RGB-Netzwerk. Exakte Werte sind in Tabelle A.4 (Seite 158) enthalten.

ansteigt. Sobald allerdings die Lernrate höher wird[4], bricht die mIoU ein. Erst im späteren Trainingsverlauf werden wieder ähnlich gute Werte wie bei der Baseline erreicht. Daraus lässt sich schließen, dass bei höheren Lernraten der Vorteil der Initialisierung mit Segmentierungs-Gewichten verloren geht. Wahrscheinlich springen die Gewichte aus ihren lokalen Minima heraus und durchlaufen einige Erhöhungen im Fehlergebirge, bevor sie in anderen lokalen Minima landen. Auch bei einer weiteren Verdopplung der Lernrate auf 0.02 konnten keine besseren Ergebnisse als mit der Baseline erreicht werden. Insgesamt sind die erlernten Features der, auf Segmentierung vortrainierten, Encoder wahrscheinlich bereits zu spezifisch um in dem sehr leichtgewichtigen Decoder noch kombiniert werden zu können. Die erzielte mIoU bei zusätzlich vortrainiertem Decoder (•••) ähnelt stark der mIoU bei lediglich vortrainierten Encodern (▪▪). Da keine weitere Verbesserung erwartet wird, wurde das Experiment vorzeitig abgebrochen und auch keine Experimente mit tieferen Encodern durchgeführt. In allen nachfolgenden Experimenten werden für die Encoder lediglich auf ImageNet-vortrainierte Gewichte verwendet.

[4] Der verwendete Lernratenscheduler OneCycle beginnt bei einer niedrigen Lernrate und erreicht erst nach 50 Epochen die maximale Lernrate. Siehe hierzu auch Abschnitt 5.7.



(a) lr=0.00125 **(b)** lr=0.01

Abbildung 6.5: Trainingsverlauf mit und ohne Vortrainieren der Encoder auf unimodalen Segmentierungsnetzwerken
(a) Bei niedriger Lernrate steigt die mIoU beim Modell mit auf Segmentierung vortrainierten Encodern deutlich schneller an als bei auf ImageNet vortrainierten Encoder. Allerdings stagniert die mIoU nach wenigen Epochen. Nach 211 Epochen greift das Early Stopping.
(b) Bei hoher Lernrate steigt die mIoU beim Modell mit auf Segmentierung vortrainierten Encodern bei Trainingsbeginn noch rasanter an. Allerdings stürzt die mIoU nach bereits 10 Epochen ein. Erst nach 48 Epochen werden wieder höhere Werte erreicht. Die Einbrüche in beiden Trainings sind auf die generell hohe Lernrate zurückzuführen.

6.1.4 Unterschiedlich tiefe Encoder

Wie in Abbildung 6.2 zu erkennen ist, können mit den unimodalen RGB-Netzwerken (●) deutlich bessere Ergebnisse erreicht werden als mit den unimodalen Tiefen-Netzwerken (●). Daraus lässt sich schließen, dass RGB-Bilder mehr semantische Informationen enthalten als Tiefenbilder. Aus diesem Grund soll untersucht werden, wie ein multimodales RGBD-Netzwerk mit unterschiedlich tiefen Encodern abschneidet. Da aus den RGB-Bildern mehr semantische Informationen extrahiert werden können, wird für die RGB-Bilder ein ResNet34 als tieferer Encoder verwendet. Die Tiefenbilder sollen nur zusätzliche komplementäre Informationen liefern, weswegen hier ein flacheres ResNet18 als Encoder zum Einsatz kommt. In Abbildung 6.6 sind die erzielten Ergebnisse dargestellt.

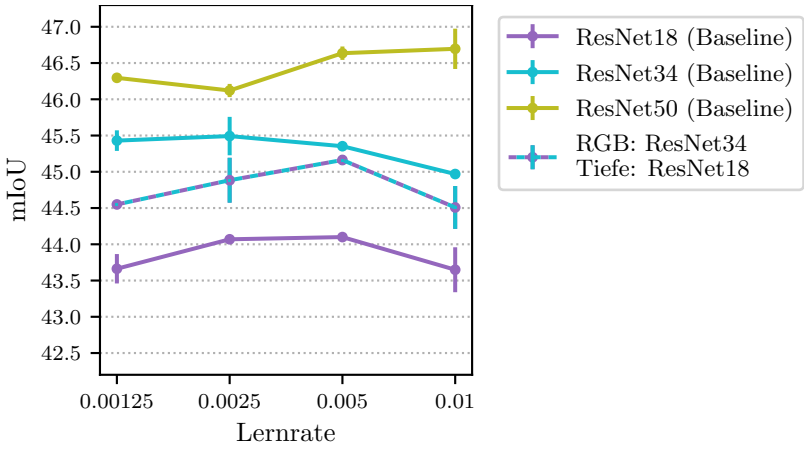


Abbildung 6.6: Unterschiedlich tiefe Encoder – mIoU
Einordnung der Segmentierungsleistung eines Netzwerks mit ResNet34-Encoder für die RGB-Bilder und ResNet18-Encoder für die Tiefenbilder im Vergleich zu den bisherigen Baselines. Um die Unterschiede deutlicher zu erkennen wird von nun an eine feiner aufgelöste Achseneinteilung verwendet. Exakte Werte sind in Tabelle A.5 (Seite 158) enthalten.

Wie zu erwarten liegen die Segmentierungsleistungen der Netzwerke mit unterschiedlich tiefen Encoder (◐) zwischen den ResNet18- (●) und den ResNet34-Netzwerken (●). Bei einer Lernrate von 0.005 können mit einer mittleren mIoU von 45.36 beinahe die Ergebnisse des ResNet34-Netzwerk mit einer mittleren mIoU von 45.50 bei Lernrate 0.0025 erreicht werden. Mit 20.6 FPS ist die Netzwerkarchitektur mit unterschiedlich tiefen Encoder zudem schneller als die ResNet34-Netzwerkarchitektur mit 16.7 FPS, wenn auch langsamer als die ResNet18-Netzwerkarchitektur mit 26.6 FPS.
Dennoch liefert das ResNet34-Netzwerk mit einer maximal erreichten mIoU von 45.76 etwas bessere Ergebnisse. Daher wird im weiteren Verlauf der Masterarbeit für beide Encoder ein ResNet34 verwendet. Wenn für die Anwendung eine schnellere Inferenzzeit gewünscht ist, kann immer noch auf einen flacheren Tiefen-Encoder zurückgegriffen werden.

6.2 Variation der Encoder-Blöcke

Die beiden ResNet34-Encoder machen den Hauptteil der Berechnungen im gesamten Netzwerk aus. Daher soll als Nächstes untersucht werden, inwiefern die Effizienz der

beiden Encoder verbessert werden kann – möglichst ohne Einbußen in der Segmentierungsleistung hinzunehmen. Die insgesamt 16 ResNet-Non-Bottleneck-Blöcke im ResNet34 werden dafür durch effizientere Blöcke ersetzt. Das so modifizierte ResNet wird anschließend auf ImageNet [Russakovsky et al., 2015] vortrainiert. Dadurch kann ein fairer Vergleich zum vortrainierten ResNet34 gezogen werden. Da ein ImageNet-Training etwa eine Woche Zeit in Anspruch nimmt, können nur wenige, ausgewählte Blöcke getestet werden. In Abschnitt 6.2.1 wird zuerst eine Auswahl für bestimmte Encoder-Blöcke getroffen. Danach wird in Abschnitt 6.2.2 das ImageNet-Training und die erzielten Ergebnisse auf ImageNet beschrieben. Zuletzt werden in Abschnitt 6.2.3 die Ergebnisse auf dem SUN RGB-D Datensatz dargestellt und interpretiert.

6.2.1 Auswahl der Encoder-Blöcke

Für die Auswahl der Encoder-Blöcke wird erneut ein Blick auf Tabelle 3.1 (Seite 43) geworfen. Da das ESPNet [Mehta et al., 2018] und das ESPNetv2 [Mehta et al., 2019b] im Vergleich zu anderen Architekturen schlechtere mIoU-Werte erreichen, werden das ESP-Modul und das EESP-Modul nicht näher betrachtet. Für die anderen, in Abschnitt 3.2 vorgestellten, Blöcke werden Inferenzzeiten gemessen. Alle ResNet-Non-Bottleneck-Blöcke im RGBD-Segmentierungsnetzwerk werden für die Inferenzzeitmessung durch einen von insgesamt vier effizienten Blöcke ersetzt:

- Non-Bottleneck-1D-Block
  aus dem ERFNet [Romera et al., 2018] (erklärt in Abschnitt 3.2.1)
- Split-Shuffle-Non-Bottleneck-Block
  aus dem LEDNet [Wang et al., 2019a] (erklärt in Abschnitt 3.2.1)
- Parallel Factorized Convolution Unit (PFCU)
  aus dem ESNet [Wang et al., 2019b] (erklärt in Abschnitt 3.2.1)
- Depthwise-Asymmetric-Bottleneck-Block (DAB-Block)
  aus dem DABNet [Li et al., 2019a] (erklärt in Abschnitt 3.2.2)

Die gemessenen Inferenzzeiten sollen dabei helfen eine Auswahl an Encoder-Blöcken zu treffen, mit welchen ein Training stattfinden soll. Zusätzlich soll ein eigens entwickelter Block getestet werden. Der Inverted-Residual-Bottleneck-Block aus dem MobileNetv2 [Sandler et al., 2018] fehlt in dieser Auflistung. Anstatt die Blöcke im

ResNet durch Inverted-Residual-Bottleneck-Blöcke auszutauschen, soll das EfficientNet[5] den gesamten Encoder ersetzen. Nachfolgend wird der Aufbau des EfficientNets beschrieben. Danach wird auf die gemessenen Inferenzzeiten und den eigenen Encoder-Block eingegangen.

EfficientNet. Das EfficientNet besteht zum Großteil aus Inverted-Residual-Bottleneck-Blöcken mit Squeeze-and-Excitation.[6] Die Netzwerkkonfiguration des kleinsten EfficientNet-B0 ist in Tabelle 6.2 dargestellt.

Stufe i	Operator $\hat{\mathcal{F}}_i$	Auflösung $H_{in} \times W_{in}$	#Kanäle C_{out}	#Layer L_i
1	3×3 Conv	224×224	32	1
2	IRB - 1, 3×3	112×112	16	1
3	IRB - 6, 3×3	112×112	24	2
4	IRB - 6, 5×5	56×56	40	2
5	IRB - 6, 3×3	28×28	80	3
6	IRB - 6, 5×5	14×14	112	3
7	IRB - 6, 5×5	14×14	192	4
8	IRB - 6, 3×3	7×7	320	1
9	1×1 Conv & Pooling & FC	7×7	1280	1

Tabelle 6.2: EfficientNet-B0 Konfiguration
Jede Zeile beschreibt eine Stufe i mit L_i Layern bzw. Blöcken, der Input-Auflösung $H_{in} \times W_{in}$ und der Anzahl an Output-Kanälen C_{out}. IRB steht für Inverted-Residual-Bottleneck-Block. Die Inverted-Residual-Bottleneck-Blöcke sind mit Squeeze-and-Excitation-Modul. Die Zahl dahinter gibt den Expansionsfaktor an. Nach dem Komma steht die Kernelgröße der Depthwise Convolution. Die Input-Auflösung entspricht der Auflösung für ein ImageNet-Training. Tabelle angelehnt an [Tan und Le, 2019].

[5] Das EfficientNet [Tan und Le, 2019] erzielte kürzlich herausragende Ergebnisse in Bezug auf Effizienz und Accuracy. Das EfficientNet gibt es – wie das ResNet – in unterschiedlichen Komplexitäten, wobei das EfficientNet-B0 das kleinste Netzwerk ist. Das EfficientNet-B0 enthält nur 0.39 GFLOPs im Gegensatz zu den 3.6 GFLOPs des ResNet34. Dennoch erzielt es mit einer Top-1-Accuracy auf ImageNet von 76.3 bessere Ergebnisse als das ResNet34 mit einer Top-1-Accuracy von 73.27.

[6] Der Inverted-Residual-Bottleneck-Block und das Squeeze-and-Excitation-Modul werden in Abschnitt 3.2.3 erklärt.

Um das EfficientNet-B0 als Basisnetzwerk verwenden zu können, werden die letzten Layer zur Klassifikation (Stufe 9) entfernt. Somit besteht der Output des EfficientNets aus 320 Feature Maps. Die letzte 1×1 Convolution erhöht die Kanalanzahl im EfficientNet-B0 von 320 auf 1280 Kanäle. Da der Decoder des RGBD-Segmentierungsnetzwerks (siehe Abbildung 5.1d) aus nur 128 Kanälen besteht, und der SUN RGB-D Datensatz mit 37 Klassen deutlich weniger Klassen enthält als der ImageNet-Datensatz mit 1000 Klassen, wurde diese 1×1 Convolution im Encoder weggelassen. Wie im ResNet werden auf ImageNet vortrainierte Gewichte verwendet.[7] Das RGBD-Segmentierungsnetzwerk erreicht mit EfficientNet-B0-Encoder (●) eine höhere Verarbeitungsfrequenz als mit ResNet34-Encoder (●), wie in Tabelle 6.3 zu erkennen ist. Dennoch ist der Unterschied nicht so groß, wie die Anzahl an FLOPs vermuten lassen (0.39 GFOPs vs. 3.6 GFLOPs). Dies liegt zum Einen an der bereits erwähnten weniger effizienten Implementierung der Depthwise Convolution. Zum Anderen basiert die verwendete PyTorch-Implementierung auf der TensorFlow-Implementierung. Um die TensorFlow-Gewichte übernehmen zu können, sind einige Anpassungen, wie manuelles Padding, notwendig, die das Netzwerk insgesamt verlangsamen. Mit dem nächstgrößeren EfficientNet-B1 sinkt die Verarbeitungsfrequenz bereits auf 14.4 FPS. Da selbst das EfficientNet-B0 mit einer Batchsize von 8 lediglich auf einer TitanRTX trainiert werden konnte, wurden keine größeren EfficientNets als Encoder getestet.

Non-Bottleneck-1D-Block. Obwohl der Non-Bottleneck-1D-Block (●) mit den Factorized Convolutions (siehe Abbildung 6.7a) nur etwa ⅔ der Berechnungen des Baseline-Blocks benötigt, ist die Verarbeitungsfrequenz mit dem Block im gesamten RGBD-Segmentierungsnetzwerk geringer, wie in Tabelle 6.3 zu erkennen ist. Dies ist auf den verdoppelten Aufrufen an Convolution-Operationen und der weniger effizienten Implementierung der Factorized Convolutions zurückzuführen. Da jedoch alle effizienten Encoder-Blöcke der Liste Factorized Convolutions verwenden, soll dennoch der Einfluss der Faktorisierung auf die mIoU untersucht werden. Aufgrund der hohen Downsamplingrate von 32 im ResNet und dem bereits vorhandenen Kontextmodul im RGBD-Segmentierungsnetzwerk wird auf die Dilation im Non-Bottleneck-1D-Block verzichtet.

[7] Die Implementierung des EfficientNets und die vortrainierten Gewichte stammen von `https://github.com/lukemelas/EfficientNet-PyTorch` – Abgerufen am 24.02.2020

Encoder		FPS
ResNet34	Non-Bottleneck-Block (Baseline)	16.7
	Non-Bottleneck-1D-Block	14.9
	Split-Shuffle-Non-Bottleneck-Block	18.6
	Parallel Factorized Convolutions Unit	8.3
	Depthwise-Asymmetric-Bottleneck-Block	16.6
	eigener Block	16.2
EfficientNet-B0		19.4
EfficientNet-B1		14.4

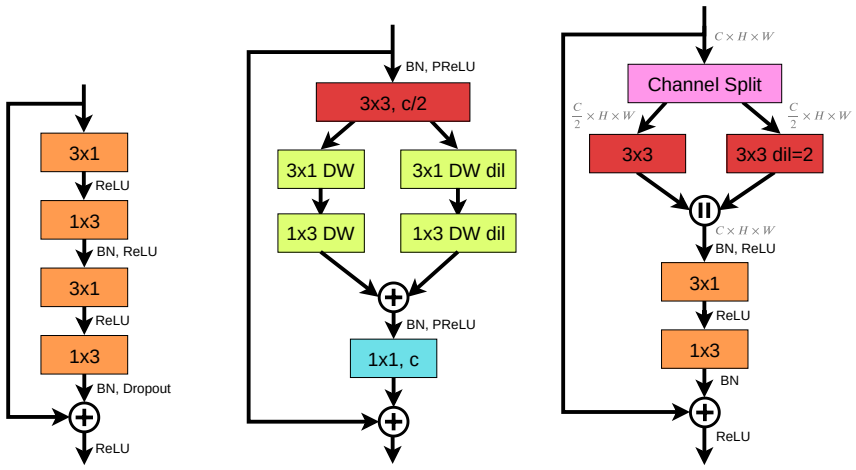
Tabelle 6.3: Verarbeitungsfrequenzen bei Verwendung unterschiedlicher Encoder
Die Verarbeitungsfrequenzen sind jeweils für das gesamte RGBD-Segmentierungsnetzwerk angegeben. Für die Inferenzzeitmessung wurden im ResNet34 die Non-Bottleneck-Blöcke durch andere effizientere Blöcke ersetzt. Zusätzlich wurden Inferenzzeiten mit EfficientNet-B0-Encoder gemessen. Das Netzwerk wurde für die Inferenzzeitmessung in eine TensorRT-Float32-Engine umgewandelt. Die verwendeten Encoder-Blöcke sind mit farbigen Punkten markiert. Diese Farben entsprechen den Farben in Abbildung 6.8. Verarbeitungsfrequenzen für PyTorch und TensorRT Float16 sind in Tabelle A.6 (Seite 159) zu finden.

Split-Shuffle-Non-Bottleneck-Block. Durch die Aufteilung in zwei Gruppen (siehe Abbildung 3.3b auf Seite 25) kann mit dem Split-Shuffle-Non-Bottleneck-Block eine höhere Verarbeitungsfrequenz von 18.6 FPS erreicht werden. Jedoch reduziert sich durch dadurch wahrscheinlich auch die Repräsentationskraft des Blocks. Aufgrund der ansonsten großen Ähnlichkeit zum Non-Bottleneck-1D-Block und der langen ImageNet-Trainingszeit wird den anderen Blöcken der Vorrang gelassen.

Parallel Factorized Convolutions Unit. Das RGBD-Segmentierungsnetzwerk mit der Parallel Factorized Convolutions Unit erreicht nur etwa die Hälfte der Verarbeitungsfrequenz wie mit dem Baseline-Blocks. Durch die drei parallelen Zweigen (siehe Abbildung 3.3c auf Seite 25) wird der Block deutlich rechenaufwendiger. Allerdings wird der Block im ESNet [Wang et al., 2019b] auch erst in späteren Layern eingesetzt, und nicht wie hier im gesamten Netzwerk. Aufgrund der geringen Verarbeitungsfrequenz wird auch hier auf ein Training verzichtet.

Depthwise-Asymmetric-Bottleneck-Block. Mit dem DAB-Block (●) wird in etwa die selbe Verarbeitungsfrequenz wie mit dem Baseline-Block erreicht. Das Bottleneck-Design und die Depthwise Factorized Convolutions (siehe Abbildung 6.7b) machen diesen Block sehr effizient. Mithilfe der zwei parallelen Zweige mit unterschiedlichen Dilationraten können sowohl lokale Informationen als auch Kontextinformationen extrahiert werden. Durch die hohe Downsamplingrate von 32 im Encoder und dem Kontextmodul im RGBD-Netzwerk werden allerdings bereits Kontextinformationen extrahiert. Daher soll untersucht werden, inwiefern zusätzliche Dilated Convolutions im Encoder Vorteile bringen. Aus diesem Grund wird der DAB-Block in zwei verschiedenen Varianten verwendet: In der ersten Variante (●) wird die Dilationrate in beiden Zweigen auf $d = 1$ gesetzt und entspricht somit einer Convolution ohne Dilation. In der zweiten Variante (●) wird im zweiten Zweig eine Dilationrate von $d = 2$ verwendet.

Eigener Block. Zusätzlich soll ein eigens entwickelter Non-Bottleneck-Block (●) getestet werden. Dieser Block kombiniert mehrere Konzepte aus anderen Encoder-Blöcken und ist in Abbildung 6.7c dargestellt.



(a) N-Bt-1D-Block **(b)** DAB-Block **(c)** eigener Block

Abbildung 6.7: Getestete Encoder-Blöcke

(a) Non-Bottleneck-1D-Block (●) aus dem ERFNet von [Romera et al., 2018].

(b) Depthwise-Asymmetric-Bottleneck-Block (●) des DABNets von [Li et al., 2019a].

(c) Selbst entwickelter Block mit Grouped Dilated und Factorized Convolutions (●).

Zuerst werden wie im Split-Shuffle-Non-Bottleneck-Block die Feature Maps in zwei gleich große Gruppen eingeteilt. Auf jede Gruppe wird eine 3×3 Convolution angewandt. Im zweiten Zweig wird eine Dilationrate von $d = 2$ verwendet. Dadurch sollen – ähnlich wie im DAB-Block – sowohl lokale Informationen als auch Kontextinformationen extrahiert werden. Die Feature Maps beider Gruppen werden anschließend konkateniert. Danach wird – wie in den anderen effizienten Non-Bottleneck-Blöcken – ein Factorized-Convolutions-Paar eingesetzt. Die Factorized Convolutions verarbeiten alle Kanäle des Tensors. Dadurch ist ein Austausch zwischen den Feature Maps beider Gruppen aus dem ersten Teil des Blocks gewährleistet und ein Channel Shuffle wird überflüssig. Zuletzt wird der Output des Blocks mit dem Input elementweise addiert. Im Gegensatz zu den anderen effizienten Non-Bottleneck-Blöcken enthält dieser Block noch 3×3 Convolutions. Diese können räumliche Features extrahieren, welche durch die eindimensionalen Convolutions nicht approximiert werden können. Da die 3×3 Convolutions jeweils nur die Hälfte der Feature Maps verarbeiten, benötigen sie dennoch weniger Rechenoperationen als ein Factorized-Convolutions-Paar.[8] Im zweiten Teil des Blocks werden alle Feature Maps verarbeitet. Dadurch können mehr Abhängigkeiten zwischen den Kanälen erlernt werden. Vor allem die erste 3×1 Convolution erhält die Aufgabe die lokalen Informationen und Kontextinformationen aus dem ersten Teil zu kombinieren.
Die Verarbeitungsfrequenz des RGBD-Segmentierungsnetzwerks mit dem eigenen Block liegt bei 16.2 FPS und damit nur knapp unter der Baseline mit 16.7 FPS. Wie bereits beim Non-Bottleneck-1D-Block ist die Verarbeitungsfrequenz trotz weniger Rechenoperationen im Vergleich zur Baseline etwas geringer.

Insgesamt werden drei verschiedene Encoder-Blöcke und das EfficientNet-B0 getestet. Die drei Blöcke sind zum Vergleich in Abbildung 6.7 dargestellt.

[8] Die Aufteilung in zwei Gruppen reduziert den Rechenaufwand um die Hälfte. Dahingegen wird der Rechenaufwand bei einem Factorized-Convolutions-Paar mit einer Kernelgröße von 3 nur um 1/3 reduziert.

6.2.2 Pre-Training auf ImageNet

Die drei modifizierten ResNets werden zunächst auf ImageNet vortrainiert. Hierfür wird die Standard Trainings-Konfiguration[9] zum Trainieren auf ImageNet verwendet: Insgesamt wird das Netzwerk 90 Epochen lang trainiert. Die Anfangslernrate beträgt normalerweise 0.1 bei einer Batchsize von 256. Da auf einer GTX 1080 Ti nicht mit einer Batchsize von 256 trainiert werden konnte, wurde sowohl die Batchsize als auch die Startlernrate halbiert. Nach jeweils 30 Epochen wird die Lernrate mit $1/10$ multipliziert. Als Optimierer wird SGD mit Momentum verwendet. Nachfolgend wird die Vorverarbeitung der Trainings- und Validierungsdaten beschrieben.

Vorverarbeitung der Trainingsdaten. Aus den Trainingsbildern wird ein zufälliger Bereich herausgeschnitten. Die Größe des Patches liegt im Bereich [0.08, 1.0] des Originalbilds und das Seitenverhältnis im Bereich $\left[\frac{3}{4}, \frac{4}{3}\right]$. Der Patch wird anschließend auf die Größe 224×224 skaliert. Danach wird das Bild mit den Mittelwerten und Standardabweichungen der drei Farbkanäle normalisiert. Zuletzt wird das Bild zufällig horizontal gespiegelt.

Vorverarbeitung der Validierungsdaten. Die Validierungsbilder werden zuerst auf die Größe 256×256 skaliert. Anschließend wird ein Patch der Größe 224×224 aus dem mittleren Bereich herausgeschnitten (Center-Crop). Die Patches werden mit den Mittelwerten und Standardabweichungen der Trainingsdaten normalisiert. Im Gegensatz zu anderen Verfahren wird hier nur der Center-Crop validiert.[10] Um die somit erzielte Accuracy mit der Accuracy des ResNet34 vergleichen zu könnnen, wird auch für das ResNet34 und für das EfficientNet-B0 nochmals die Validierungs-Accuracy mit lediglichem Center-Crop bestimmt.

[9] Die Trainings-Konfiguration zum Trainieren auf ImageNet ist beispielsweise zu finden unter `https://github.com/pytorch/examples/blob/master/imagenet/main.py` – Abgerufen am 25.02.2020

[10] Andere Verfahren verwenden zum Benchmarking einen 5-Crop oder sogar einen 10-Crop. Beim 5-Crop werden zusätzlich zum Center-Crop die vier Eck-Bereiche durch das Netzwerk propagiert. Beim 10-Crop werden zudem die horizontalen Spiegelungen des 5-Crops verwendet. Anschließend werden die Softmaxausgaben der 10 Patches gemittelt. Eine solche Validierung würde allerdings die Validierungszeit verzehnfachen. Da das ImageNet-Training nur als Pre-Training für das Transfer Learning auf Segmentierung dient, wird hier die einfachere Validierung mit lediglichem Center-Crop verwendet.

Besonderheiten beim Trainieren des DAB-Blocks. Der DAB-Block wird in zwei verschieden Varianten getestet, die sich lediglich in der Dilationrate des zweiten Zweigs unterscheiden. Für ein schnelleres Training wird nur die Variante mit einer Dilationrate von $d = 1$ für die vollen 90 Epochen auf ImageNet trainiert. Für die Variante mit einer Dilationrate von $d = 2$ im zweiten Zweig werden die Gewichte mit den Gewichten des DAB-Blocks der ersten Variante initialisiert. Es erfolgt hierfür lediglich ein Nachtrainieren mit der zuletzt verwendeten Lernrate von 0.0005. Da sich nach 10 Epochen keine weitere Verbesserung auf den ImageNet-Validierungsdaten ergab, wurde das Training an dieser Stelle abgebrochen.

Ergebnisse auf ImageNet. Die erzielten Top-1- und Top-5-Accuracy-Werte auf den ImageNet-Validierungsdaten sind in Tabelle 6.4 dargestellt.

	Encoder	Top-1-Acc.	Top-5-Acc.
ResNet34	• Non-Bottleneck-Block (Baseline)	71.94	90.81
	• Non-Bottleneck-1D-Block	71.36	90.57
	• DAB-Block (Dilation 1)	65.51	86.98
	• DAB-Block (Dilation 2)	64.80	86.36
	• eigener Block	71.00	90.57
• EfficientNet-B0		76.92	93.51

Tabelle 6.4: Accuracy auf den ImageNet-Validierungsdaten
Die Top-1- und Top-5-Accuracy des ResNet34 (Baseline) und des EfficientNet-B0 wurden nochmals mit lediglichem Center-Crop bestimmt und weichen daher von den genannten Werten im jeweiligen Paper ab.

Die Accuracy des ResNet mit Non-Bottleneck-1D-Block liegt nur knapp unter der Accuracy des ResNet34 (Baseline). Direkt im Anschluss folgt das ResNet mit dem eigenen Block. Beide Varianten mit DAB-Block erzielen eine deutlich schlechtere Accuracy. Die zweite Variante mit einer Dilationrate von $d = 2$ im zweiten Zweig erzielt etwas schlechtere Ergebnisse als die erste Variante mit einer Dilationrate von $d = 1$ in beiden Zweigen. Dies kann allerdings auch auf das ledigliche Nachtrainieren zurückzuführen sein. Würde die zweite Variante auch auf den vollen 90 Epochen

trainiert werden, könnten unter Umständen bessere Ergebnisse erzielt werden. Das EfficientNet-B0 erreicht mit Abstand die höchste Accuracy.
Als erste Prognose lässt sich sagen, dass das RGBD-Segmentierungsnetzwerk mit Non-Bottleneck-1D-Block und dem eigenen Block wahrscheinlich ähnliche Ergebnisse in der Segmentierung erzielen wie die Baseline. Die erzielten mIoU-Werte beider Varianten des DAB-Blocks liegen voraussichtlich deutlich darunter. Mit EfficientNet-B0-Encoder werden wahrscheinlich die besten Ergebnisse erzielt.

6.2.3 Erzielte Ergebnisse

Mit den auf ImageNet vortrainierten Encodern wird anschließend das RGBD-Segmentierungsnetzwerk auf dem SUN RGB-D Datensatz trainiert. Die erzielten mIoU-Werte sind in Abbildung 6.8 dargestellt.

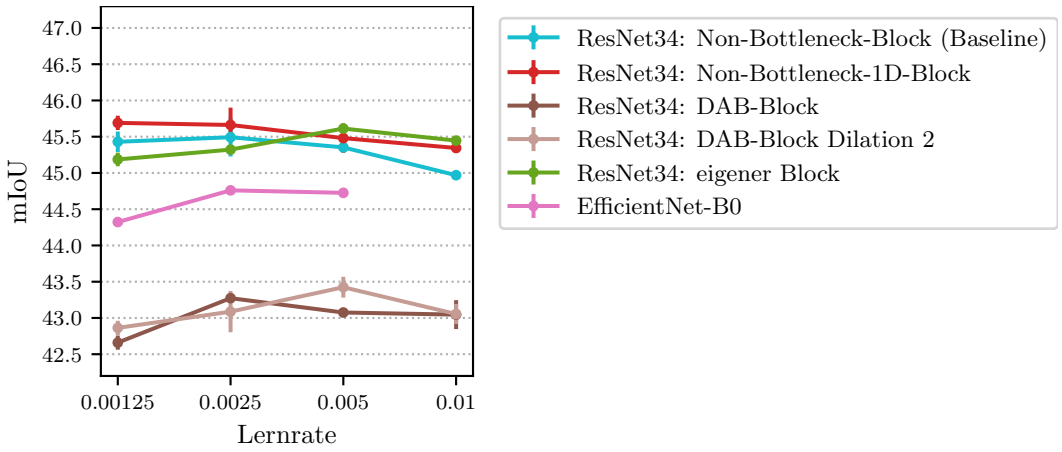


Abbildung 6.8: Variation der Encoder-Blöcke – mIoU
Im RGBD-Segmentierungsnetzwerk werden die Blöcke im ResNet34 variiert. Zusätzlich wird das EfficientNet-B0 als Encoder verwendet.
Der ResNet-Non-Bottleneck-Block (●) entspricht dem Standard ResNet34 (Baseline).
Exakte Ergebnisse sind in Tabelle A.7 (Seite 159) zu finden.

Zunächst ist zu erkennen, dass wie erwartet die Netzwerke mit Standard-ResNet34-Encoder (Baseline) (●) und die beiden Varianten mit Non-Bottleneck-1D-Block (●) und dem eigenen Block (●) ähnliche mIoU-Werte erzielen.
Obwohl die Variante mit Non-Bottleneck-1D-Block (●) eine geringere Parameteranzahl hat und weniger FLOPs benötigt als das Netzwerk mit Standard-ResNet34 (●), liegt

die mIoU bei allen Lernraten über der mIoU der Baseline. Die Variante mit dem eigenen Block (●) liegt für die geringeren Lernraten unterhalb und für die höheren Lernraten oberhalb der mIoU der anderen beiden Varianten. Dieser Anstieg der Segmentierungsleistung könnte auf die Dilation im zweiten Zweig zurückzuführen sein: Für die Aufgabe der Klassifikation sind Kontextinformationen nicht so wichtig wie für die semantische Segmentierung. Daher muss die Dilated Convolution sich stärker auf die neue Segmentierungsaufgabe umstellen und erlernen sinnvolle Kontextinformationen zu extrahieren. Dieser erhöhten Anforderung der Gewichtsanpassung könnte mit höheren Lernraten besser nachgekommen werden.

Erstaunlicherweise erzielen die Netzwerke mit EfficientNet-Encoder (●) deutlich schlechtere Ergebnisse. Aufgrund des eindeutigen Trends wurde hier jede Konfiguration nur einmal trainiert und auch das Training mit der Lernrate 0.01 weggelassen. Die schlechten Ergebnisse könnten auch auf die fehlende 1×1 Convolution nach dem letzten Inverted-Residual-Bottleneck-Block zurückzuführen sein. Diese 1×1 Convolution erhöht die Kanalanzahl im EfficientNet-B0 von 320 auf 1280 Kanäle, wie in Tabelle 6.2 dargestellt ist. Aufgrund dieser Überlegung wurden zusätzliche Trainings mit dieser 1×1 Convolution durchgeführt. Jedoch konnte auch hier keine bessere mIoU erreicht werden, wie in Tabelle A.7 (Seite 159) zu erkennen ist.

Beide Varianten mit DAB-Block (● ●) erzielen die mit Abstand schlechtesten Ergebnisse. Durch das Bottleneck-Design und die Depthwise Convolutions ist wahrscheinlich die Repräsentationskraft des Blocks zu gering. Allerdings kommen im DABNet [Li et al., 2019a] zuerst drei Standard 3×3 Convolutions zum Einsatz. Erst danach werden die DAB-Blöcke verwendet. Die, in dieser Masterarbeit verwendete, Netzwerkarchitektur enthält lediglich eine 7×7 Convolution und ein Max-Pooling vor den DAB-Blöcken. Jedoch ist zu erkennen, dass die Netzwerke mit einer Dilationrate von $d = 2$ im zweiten Zweig (●) etwas bessere Ergebnisse erzielen als die Netzwerke mit einer Dilationrate von $d = 1$ in beiden Zweigen (●). Diese Verbesserung wird erreicht, obwohl die Variante mit einer Dilationrate von $d = 1$ auf ImageNet eine höhere Accuracy erzielt hat, wie in Tabelle 6.4 zu erkennen ist. Daraus kann geschlossen werden, dass Dilated Convolutions im Encoder trotz Verwendung eines Kontextmoduls Vorteile für die Segmentierung bringen können.

Insgesamt wird mit dem Non-Bottleneck-1D-Block (●) die höchste mIoU erreicht. Mit Float16 erzielt dieser Block zudem die höchste Verarbeitungsfrequenz, wie in

Tabelle A.6 (Seite 159) zu erkennen ist. Daher wird dieser Block von nun an in allen weiteren Experimenten eingesetzt. Da auch die Variante mit dem eigenen Block (●) gute Ergebnisse erzielt und eine hohe Verarbeitungsfrequenz erreicht, wird dieser Block nochmals in einem späteren Experiment aufgegriffen (siehe Abschnitt 6.3.3).
Des Weiteren erzielt keine Netzwerkarchitektur bei einer Lernrate von 0.01 die besten Ergebnisse. Aus diesem Grund, und aufgrund der hohen Schwankungen des Loss und der mIoU bei dieser hohen Lernrate[11], werden in den weiteren Experimenten lediglich die drei kleineren Lernraten verwendet.

6.3 Variation im Decoder

Nach den Experimenten zu den Encodern der RGBD-Segmentierungsarchitektur wird nun der Fokus auf den Decoder gelegt. Der bisherige Decoder besteht aus drei Decoder-Modulen mit 128 Kanälen und einer 3×3 Convolution mit 37 Output-Kanälen für die 37 Klassen des SUN RGB-D Datensatzes. Das verwendete Decoder-Modul ist in Abbildung 5.1d (Seite 61) dargestellt. Zunächst soll in Abschnitt 6.3.1 die Fusion der Encoder Feature Maps in den Decoder untersucht werden. Anschließend wird in Abschnitt 6.3.2 die Breite und in Abschnitt 6.3.3 die Tiefe des Decoders erhöht, um eine möglicherweise bessere Segmentierungsleistung zu erzielen.

6.3.1 Untersuchung der Encoder-Decoder-Fusion

Zuerst soll die Fusion der Encoder Feature Maps in den Decoder untersucht werden. Bisher wurden die RGBD Feature Maps des Encoders zunächst mit einem Kanaldeskriptor des Decoders nach SSMA [Valada et al., 2019] gewichtet und erst danach in den Decoder konkateniert, wie in Abbildung 5.1e dargestellt ist. In der ersten, hier untersuchten, Variante (▪▪) wird diese Gewichtung gestrichen und die RGBD-Features direkt in den Decoder konkateniert. Durch dieses Experiment soll untersucht werden, ob die Gewichtung mit dem Decoder-Deskriptor überhaupt einen Vorteil bringt.
In einer zweiten Variante (▪▪) wird die Fusion der Encoder Features in den Decoder komplett entfernt. Die RGB- und Tiefen-Features werden in dieser zweiten Variante

[11] Die hohen Schwankungen der mIoU während des Trainings bei einer Lernrate von 0.01 sind in Abbildung 6.5b zu erkennen.

lediglich am Ende der beiden Encoder fusioniert. Hierdurch wird sowohl die Eignung der Fusion von RGB und Tiefe im Decoder (siehe Abschnitt 4.1.6), als auch der Einfluss von Skip Connections zwischen Encoder und Decoder (siehe Abschnitt 3.1.3) auf die mIoU untersucht.
Die erzielten Ergebnisse dieser beiden Varianten sind in Abbildung 6.9 dargestellt.

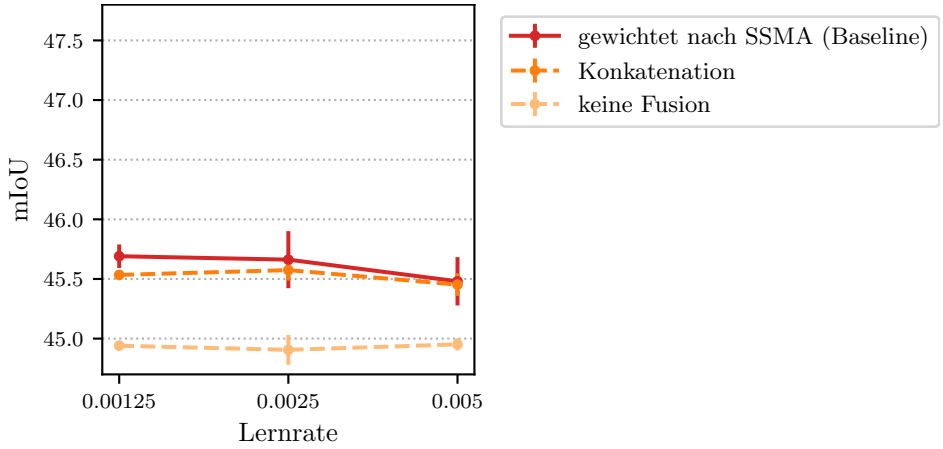


Abbildung 6.9: Verschiedene Encoder-Decoder-Fusionierungen – mIoU
Die Gewichtung der Encoder-Features mit dem Decoder-Deskriptor nach SSMA (●) entspricht der Baseline. (▪▪): einfache Konkatenation der Encoder-Features in den Decoder. (▪▪): zweite Variante völlig ohne Fusion in den Decoder. Bei allen drei Varianten werden als Encoder zwei ResNet34 mit Non-Bottleneck-1D-Blöcken verwendet. Im Vergleich zu den vorherigen Experimenten wird von nun an eine neue Achseneinteilung verwendet. Exakte Werte sind in Tabelle A.8 (Seite 160) enthalten. Zugehörige Verarbeitungsfrequenzen sind in Tabelle A.9 (Seite 160) zu finden.

Es ist zu erkennen, dass die reine Konkatenation der RGBD-Features in den Decoder (▪▪) nur minimal schlechter ist als mit zusätzlicher Gewichtung (●). Allerdings ist auch zu erkennen, dass das komplette Fehlen der Fusion in den Decoder (▪▪) die mIoU nur um 0.6 Prozentpunkte verschlechtert. Die bloße Verschlechterung zeigt, dass mit Skip Connections zwischen Encoder und Decoder und weiteren Fusionierungen von RGB und Tiefe eine verbesserte Segmentierungsleistung erreicht werden kann. Die geringe Differenz in der mIoU deutet allerdings darauf hin, dass der Decoder zu leichtgewichtig und damit nicht repräsentativ genug sein könnte: Im Vergleich zu den rechenaufwendigeren Encodern besteht der Decoder aus nur wenigen Convolutions und nur 128 Kanälen. Dem Decoder könnte daher die notwendige Repräsentationskraft

fehlen, um aus den fusionierten RGBD-Features der beiden Encoder genügend Informationen extrahieren zu können. Des Weiteren werden jeweils nur 24 RGBD Feature Maps aus den Encodern in den Decoder fusioniert. Auch hier könnte der Decoder von mehr Feature Maps profitieren. Aus diesem Grund wird zunächst die Kanalanzahl im Decoder und die Kanalanzahl der fusionierten RGBD-Features erhöht; der Decoder wird also breiter. In Abschnitt 6.3.3 wird anschließend zusätzlich die Tiefe des Decoders erhöht. In beiden Experimenten wird die Gewichtung der Encoder-Features mit dem Decoder-Deskriptor beibehalten, da der nur marginale Vorteil mit der Gewichtung auch auf den zu leichtgewichtigen Decoder zurückzuführen sein kann. Mit einem stärkeren Decoder könnte dieser Unterschied demnach deutlicher ausfallen.

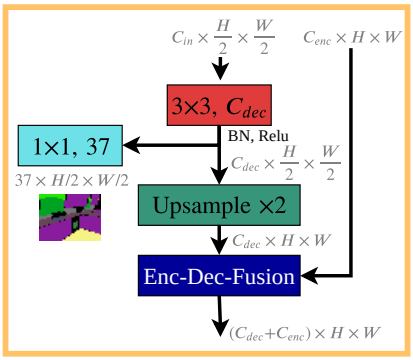
6.3.2 Breiterer Decoder

Um die Repräsentationskraft des Decoders zu erhöhen, wird die Breite des Decoders erhöht. Hierfür wird die Kanalanzahl im Decoder C_{dec} von 128 auf 256 verdoppelt (●). Außerdem wird die Kanalanzahl der fusionierten RGBD Feature Maps aus den Encodern C_{enc} von 24 auf 48 Feature Maps verdoppelt (●). Beide Erhöhungen der Kanalanzahl werden sowohl unabhängig voneinander als auch kombiniert (●) getestet, wodurch sich drei verschiedene Konfigurationen ergeben. Dadurch soll festgestellt werden, ob eine wahrscheinliche Verbesserung der mIoU hauptsächlich auf den breiteren Decoder – oder auf mehr Feature Maps aus den Encodern zurückzuführen ist.

In einer vierten Konfiguration (●) nimmt die Kanalanzahl im Decoder gegenläufig zum Encoder ab. Dies lässt sich folgendermaßen motivieren: Im Encoder wird bei Halbierung der räumlichen Auflösung die Kanalanzahl verdoppelt. Dadurch soll der räumliche Informationsverlust verringert, und die Information in mehr Kanälen gespeichert werden. Dieser Gedankengang kann auch rückwärts gegangen werden: Wird im Decoder die räumliche Auflösung verdoppelt, so kann die Kanalanzahl halbiert werden. Auch im RedNet [Jiang et al., 2018] und im ACNet [Hu et al., 2019] wird die Kanalanzahl im Decoder bei Verdoppelung der räumlichen Auflösung halbiert.

Bei dieser vierten Konfiguration nimmt die Kanalanzahl der Feature Maps, die aus den Encoder fusioniert werden, dazu gegenläufig zu, was sich wie folgt begründen lässt: Je größer die räumliche Auflösung im Decoder, desto mehr ist der Decoder auf zusätzliche Features aus dem Encoder für genauere räumliche Details angewiesen. Daher werden

bei größerer räumlicher Auflösung im Decoder mehr Feature Maps aus den Encodern in den Decoder konkateniert. Das so modifizierte Decoder-Modul ist in Abbildung 6.10a dargestellt. Die genaue Konfiguration der drei modifizierten Decoder-Module ist in Tabelle 6.10b zu erkennen.



(a) modifiziertes Decoder-Modul

$H \times W$	C_{dec}	C_{enc}
30×40	512	12
60×80	256	24
120×160	128	48

(b) Konfiguration der drei Decoder-Module

Abbildung 6.10: Modifizierte Decoder-Module bei abnehmender Kanalanzahl
C_{dec} steht für die Kanalanzahl im Decoder und C_{enc} für die Kanalanzahl der fusionierten RGBD-Features aus den Encodern. Die angegebene räumliche Auflösung in der Tabelle entspricht der räumlichen Auflösung der Feature Maps aus den Encodern. Die 3×3 Convolution im Decoder-Modul erfolgt jeweils mit der Hälfte der angegebenen räumlichen Auflösung. Nach der Konkatenation der Encoder Feature Maps in den Decoder enthält der Decoder entsprechend $C_{dec} + C_{enc}$ Kanäle.

Die erzielten Ergebnisse der unterschiedlichen Konfigurationen sind in Abbildung 6.11 dargestellt. Prinzipiell ist zu erkennen, dass mit mehr Kanälen bessere Ergebnisse erreicht werden. Beide Konfigurationen mit 256 Kanälen im Decoder (● ●) erzielen bessere Ergebnisse als beide Konfigurationen mit nur 128 Kanälen im Decoder (● ●). Daraus lässt sich schließen, dass ein breiterer Decoder wichtiger ist, als eine erhöhte Anzahl von Feature Maps aus den Encodern. Beide Konfigurationen mit $C_{enc} = 48$ (● ●) überschreiten nur bei höheren Lernraten die Segmentierungsleistungen der Konfigurationen mit $C_{enc} = 24$ (● ●). Dies könnte sich wie folgt erklären lassen: Wenn der Decoder mehr Kanäle aus den Encodern erhält, steigt das Verhältnis von Feature Maps aus den Encodern zu den Decoder Feature Maps. Es müssen mehr sinnvolle RGBD Feature Maps berechnet werden, und der Decoder muss diese zielführend einsetzen, um die Segmentierung zu verbessern. Dieser erhöhten Lernanforderung könnte mit einer höheren Lernrate besser nachgekommen werden.

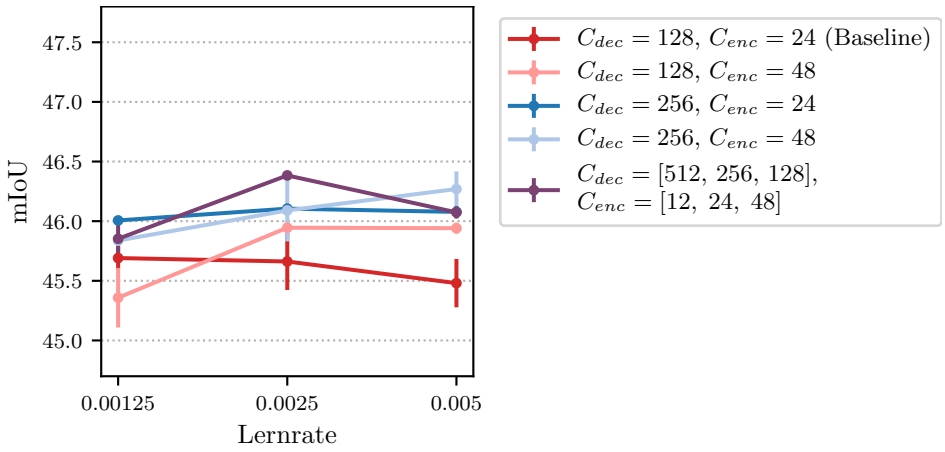


Abbildung 6.11: Breiterer Decoder – mIoU

Die Kanalanzahl der Decoder Feature Maps C_{dec} und die Kanalanzahl der fusionierten RGBD Feature Maps aus den Encodern C_{enc} wurde für dieses Experiment erhöht. Die Variante mit $C_{dec} = 128$ und $C_{enc} = 24$ entspricht der Baseline (●). Als Encoder werden in allen Konfigurationen zwei ResNet34 mit Non-Bottleneck-1D-Blöcken verwendet. Exakte Ergebnisse sind in Tabelle A.10 (Seite 160) enthalten. Zugehörige Verarbeitungsfrequenzen befinden sich in Tabelle A.11 (Seite 161).

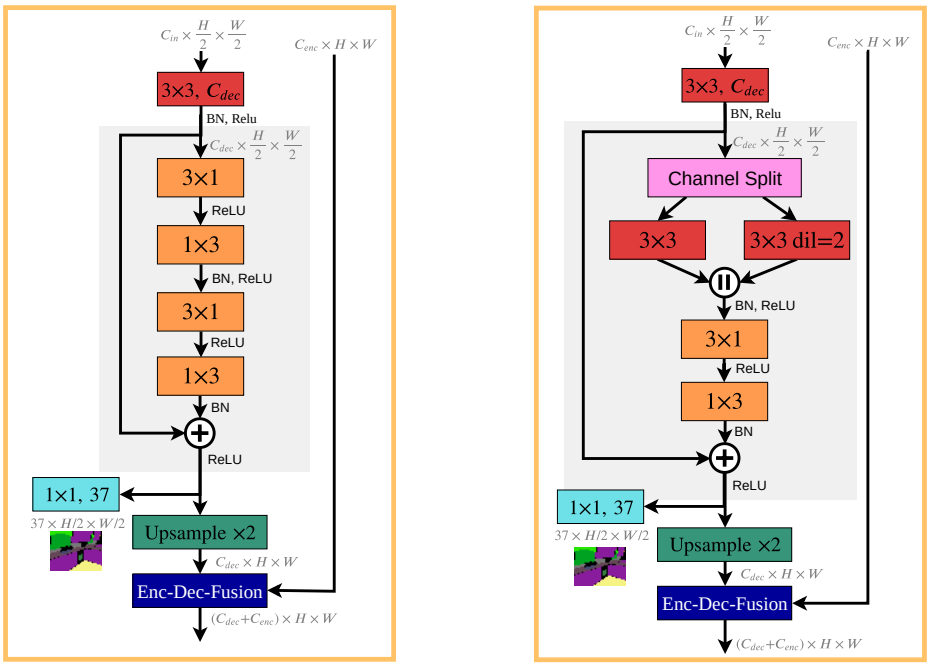
Die Konfiguration mit abnehmender Kanalanzahl (●) erreicht bei einer Lernrate von 0.0025 die bisher höchste erzielte mIoU von 46.38. Ansonsten erzielt diese Konfiguration ähnliche Ergebnisse wie die beiden Konfigurationen mit 256 Kanälen im Decoder (● ●). Dennoch kann mit dieser Konfiguration eine leicht höhere Verarbeitungsfrequenz erreicht werden, wie in Tabelle A.11 (Seite 161) zu erkennen ist. Für die nachfolgenden Experimente wird daher die Konfiguration mit abnehmender Kanalanzahl verwendet.

6.3.3 Tieferer Decoder

Nachdem die Breite im Decoder erhöht wurde, soll nun untersucht werden, ob die mIoU mit einem tieferen Decoder weiter verbessert werden kann. Hierfür wird im Decoder-Modul nach der 3×3 Convolution ein zusätzlicher Block eingesetzt.

Da der Non-Bottleneck-1D-Block im Encoder gute Ergebnisse erzielt, wird dieser Block auch im Decoder eingesetzt. Aufgrund der guten Ergebnisse des eigenen Blocks (siehe Abbildung 6.8) soll außerdem dieser Block getestet werden. Die so modifizierten Decoder-Module sind in Abbildung 6.12 dargestellt.

Der Shortcut vom Input zum Output des Blocks garantiert, dass in diesem zusätzli-



(a) mit Non-Bottleneck-1D-Block **(b)** mit eigenem Block

Abbildung 6.12: Tiefere Decoder-Module
Im Decoder-Modul wird nach der 3×3 *Convolution ein zusätzlicher Block eingefügt. (Vergleiche hierzu Abbildung 6.10a.)*

chen Block lediglich ein Residuum berechnet wird. Der zusätzliche Block sollte sich also nicht nachteilig auf die Segmentierungsleistung auswirken, sondern lediglich die bereits vorhandenen Feature Maps verbessern. Sollte sich die mIoU nicht erhöhen, so könnte daraus geschlossen werden, dass der zusätzliche Block im Decoder-Modul nicht notwendig ist.

Für die Konsistenz werden im Encoder und im Decoder die gleichen Blöcke verwendet. Es werden also zwei verschiedene Konfigurationen untersucht: In der ersten Konfiguration (▪▪) wird der Non-Bottleneck-1D-Block sowohl im Encoder als auch im Decoder verwendet. In der zweiten Konfiguration (▪▪) kommt der eigene Block sowohl im Encoder als auch im Decoder zum Einsatz.

Die zusätzlichen Blöcke in den drei Decoder-Modulen verarbeiten lediglich Feature Maps der Größe $1/32$, $1/16$ und $1/8$ der Input-Auflösung; der Einfluss auf die Inferenzzeit ist dadurch nur gering. In Tabelle A.13 (Seite 161) ist zu erkennen, dass sich die

Verarbeitungsfrequenz mit dem zusätzlichen Block um weniger als 1 FPS verringert.[12] Die erzielten Segmentierungsleistungen sind in Abbildung 6.13 dargestellt.

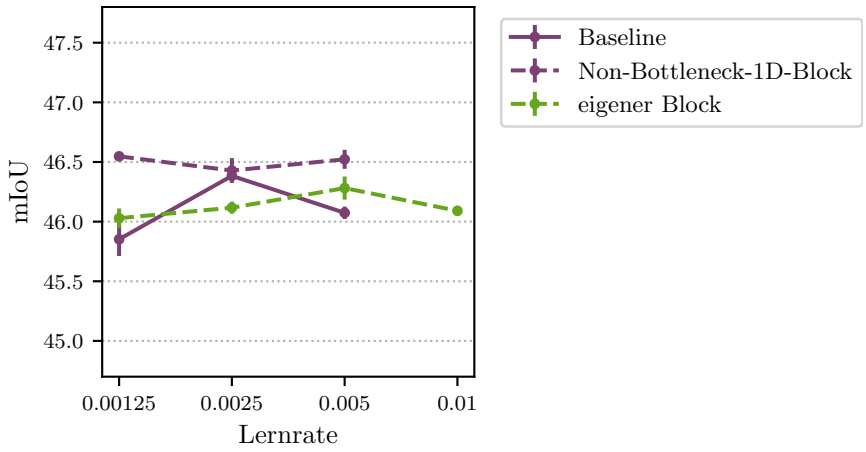


Abbildung 6.13: Zusätzliche Blöcke im Decoder – mIoU

Als Encoder kommt jeweils ein ResNet34 zum Einsatz. Für die Baseline (●) und die Variante mit dem Non-Bottleneck-1D-Block im Decoder (▪▪) wird als Encoder-Block der Non-Bottleneck-1D-Block verwendet. Für die Variante mit dem eigenen Block im Decoder (▪▪) wird auch der eigene Block als Encoder-Block eingesetzt. In allen drei Varianten ist die Kanalanzahl im Decoder abnehmend. Exakte Werte sind in Tabelle A.12 (Seite 161) enthalten. Zugehörige Verarbeitungsfrequenzen befinden sich in Tabelle A.13 (Seite 161).

Es ist zu erkennen, dass über alle Lernraten betrachtet mit dem Non-Bottleneck-1D-Block im Decoder (▪▪) bessere Ergebnisse als mit der Baseline (●) erreicht werden können. Mit dem eigenen Block im Encoder und im Decoder (▪▪) liegt die erzielte mIoU bei allen Lernraten darunter. Wie bereits bei den Experimenten zu den Encoder-Blöcken in Abschnitt 6.2.3 werden mit dem eigenen Block bei höheren Lernraten bessere Ergebnisse erzielt. Daher wurde hierfür ein weiteres Training mit der Lernrate 0.01 durchgeführt. Jedoch konnte auch mit dieser Lernrate kein höherer Wert erzielt werden. Dennoch wird mit allen Lernraten eine höhere mIoU erzielt als beim ersten Experiment zum eigenen Block (siehe Abbildung 6.8, Seite 96). Daraus kann geschlossen werden, dass unabhängig vom Encoder-Block mit einem breiteren und tieferen Decoder eine höhere Segmentierungsleistung erzielt werden kann.

[12] Die Verarbeitungsfrequenz mit dem eigenen Block (▪▪) ist höher, da der eigene Block bereits im Encoder eingesetzt wird.

Insgesamt konnte mit dem breiteren und tieferen Decoder eine Verbesserung der mIoU von 45.69 auf 46.55 erreicht werden. Für die weiteren Experimente wird daher an dem breiteren Decoder mit abnehmender Kanalanzahl und der erhöhten Tiefe durch den zusätzlichen Non-Bottleneck-1D-Block im Decoder-Modul festgehalten.

6.4 Variation der RGBD-Fusion

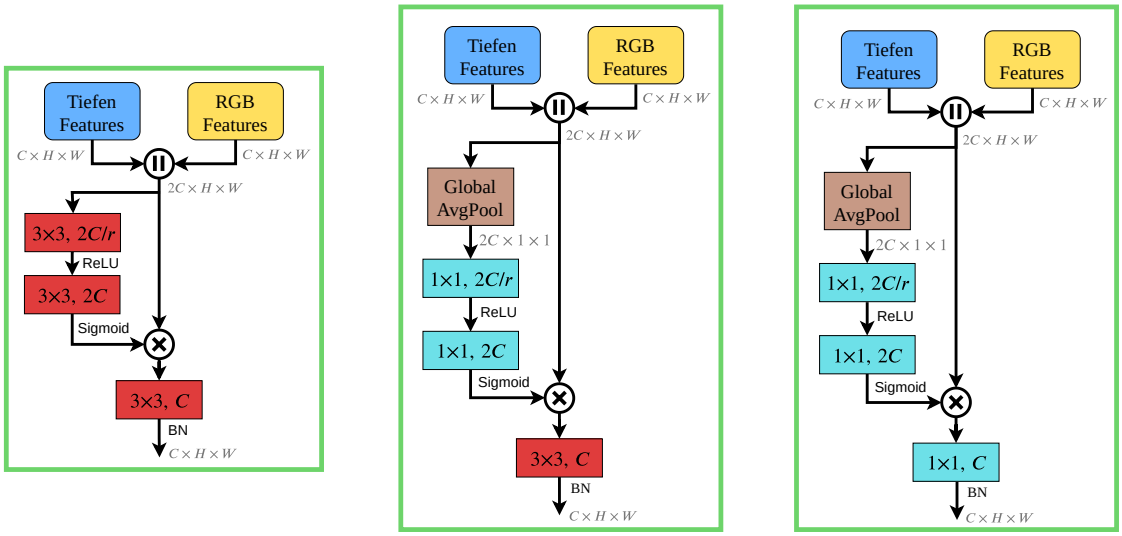
Nach den Experimenten zu den Encodern und dem Decoder der RGBD-Segmentierungsarchitektur wird nun der Fokus auf die Fusion von RGB und Tiefe gelegt. Bisher wurden RGB- und Tiefen-Features am Ende der beiden Encoder fusioniert. Zusätzlich wurden über Skip Connections Features aus den beiden Encodern in den Decoder fusioniert. In allen Fusionierungen von RGB und Tiefe wurden die Feature Maps der beiden Encoder zunächst konkateniert und anschließend pixelweise nach SSMA [Valada et al., 2019] gewichtet (siehe hierzu auch Abbildung 5.1b, Seite 61).
Zunächst wird die prinzipielle Fusion von RGB und Tiefe im Decoder beibehalten. Da die pixelweise Wichtung rechenaufwendig ist, soll in Abschnitt 6.4.1 untersucht werden, ob diese durch eine einfachere, kanalweise Wichtung ersetzt werden werden kann. Darüber hinaus wird untersucht, ob sich die Segmentierungsleistung über einen zusätzlichen, leichtgewichtigen dritten Encoder weiter verbessert. Aufgrund der guten Ergebnisse des RedNets [Jiang et al., 2018] (siehe Tabelle 4.1, Seite 57) soll in Abschnitt 6.4.2 außerdem eine Fusion der Tiefen-Features in den RGB-Encoder getestet werden. Hierbei wird zudem untersucht, ob es sinnvoll ist die Encoder-Features – wie bisher – in den Decoder zu konkatenieren, oder ob eine Addition der Encoder-Features mit den Decoder-Features bessere Ergebnisse erzielt.

6.4.1 Unterschiedliche Wichtungen und dritter Encoder

Pixelweise vs. kanalweise Wichtung. Laut [Valada et al., 2019] können mit der pixelweisen Wichtung in verschiedenen Bereichen des Bilds unterschiedliche Features hevorgehoben beziehungsweise unterdrückt werden. Über die pixelweise Wichtung nach SSMA soll somit gelernt werden, welche Features welcher Modalität in welchen Bildregionen relevant sind. Durch die Ersetzung der pixelweisen Wichtung mit einer kanalweisen Wichtung wird hier untersucht, ob die pixelweise Wichtung (▪▪) auf dem

verwendeten SUN RGB-D Datensatz einen Vorteil bringt. Hierfür werden drei verschiedene Konfigurationen getestet, welche zum Vergleich in Abbildung 6.14 dargestellt sind.

Für die kanalweise Wichtung werden die beiden 3×3 Convolutions im Wichtungszweig der RGBD-Fusion (siehe Abbildung 6.14a) durch ein weniger rechenaufwendiges Squeeze-and-Excitation-Modul (SE) (▪▪) ersetzt, wie in Abbildung 6.14b dargestellt ist. Zusätzlich wird untersucht, ob die letzte 3×3 Convolution auch durch eine einfachere 1×1 Convolution (▪▪) ersetzt werden kann (siehe Abbildung 6.14c).



(a) pixelweise Wichtung nach SSMA [Valada et al., 2019] (▪▪)

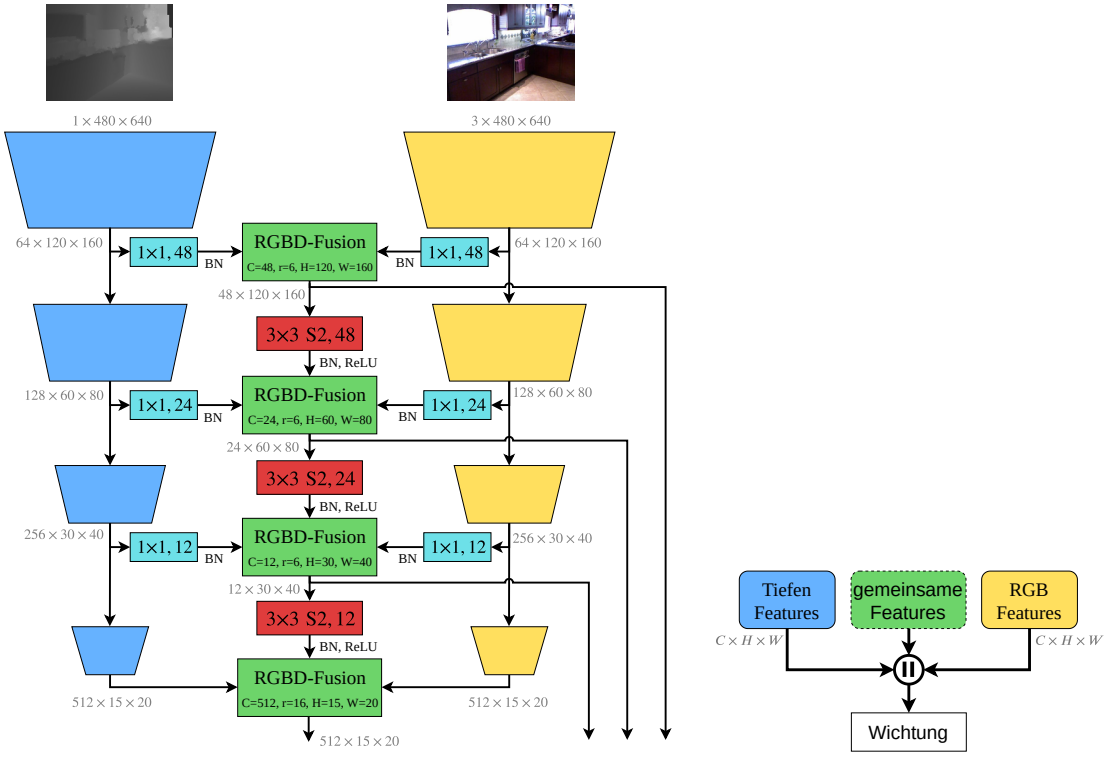
(b) kanalweise Wichtung mit 3×3 Convolution am Ende (▪▪)

(c) kanalweise Wichtung mit 1×1 Convolution am Ende (▪▪)

Abbildung 6.14: Unterschiedliche Wichtungen zur RGBD-Fusion
Die pixelweise Wichtung in (a) wird in (b) und (c) durch ein Squeeze-and-Excitation-Modul ausgetauscht. In (c) wird außerdem die letzte 3×3 *Convolution durch eine recheneffzientere* 1×1 *Convolution ersetzt. Der Rechenaufwand nimmt von links nach rechts ab.*

Dritter Encoder. Darüber hinaus soll untersucht werden, ob sich die Segmentierungsleistung verbessert, wenn die fusionierten RGBD-Features der unterschiedlichen Auflösungsstufen miteinander verbunden werden. Mit einer 3×3 Convolution mit Stride 2 werden die RGBD-Features weiter verarbeitet und in ihrer Auflösung halbiert.

Anschließend können sie mit den RGBD-Features der nächstkleineren Auflösungsstufe fusioniert werden. Dadurch entsteht ein leichtgewichtiger dritter Encoder (siehe Abbildung 6.15a).

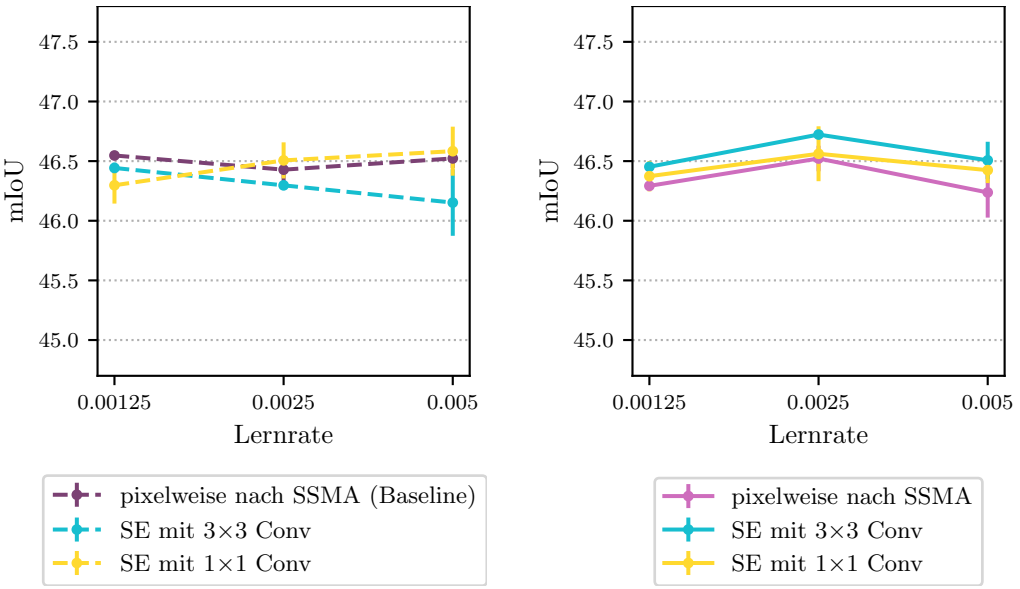


(a) Netzwerkarchitektur **(b)** Konkatenation vor der Wichtung

Abbildung 6.15: Netzwerkarchitektur mit drittem Encoder
(a) Die fusionierten RGBD-Features der beiden Encoder werden über 3×3 *Convolutions mit Stride 2 miteinander verbunden. Die angegebene Kanalanzahl der RGBD-Features entspricht der Konfiguration für die abnehmende Kanalanzahl im Decoder (siehe Tabelle 6.10b).*
(b) Ab der zweiten RGBD-Fusion werden vor der Wichtung die Feature Maps aus den beiden Encodern und zusätzlich die Feature Maps des dritten Encoders konkateniert. Unterschiedliche Wichtungen sind in Abbildung 6.14 dargestellt.
(Vergleiche hierzu die Basisarchitektur in Abbildung 5.1, Seite 61)

Ab der zweiten Fusion von RGB- und Tiefen-Features werden die Feature Maps aus allen drei Encodern konkateniert, wie in Abbildung 6.15b dargestellt ist. Danach erfolgt die Wichtung der Feature Maps. Hierfür werden alle drei Wichtungen aus

Abbildung 6.14 getestet. Da die RGBD Features aus nur wenigen Kanälen bestehen, erhöht sich der Rechenaufwand durch den dritten Encoder nur geringfügig, wie in Tabelle A.15 (Seite 162) zu erkennen ist.
In Abbildung 6.16 sind die erzielten Ergebnisse mit und ohne drittem Encoder dargestellt. Es ist zu erkennen, dass der Einfluss auf die mIoU in allen Varianten nur



(a) Reine Variation der Gewichtung **(b)** Mit drittem Encoder

Abbildung 6.16: Variation der Gewichtung mit und ohne drittem Encoder – mIoU
RGB- und Tiefen-Features werden zuerst konkateniert und anschließend unterschiedlich gewichtet. (b) zusätzlicher dritter Encoder
In allen Varianten kommen als Encoder zwei ResNet34 mit Non-Bottleneck-1D-Blöcken zum Einsatz. Die Kanalanzahl im Decoder ist abnehmend. Die Decoder-Module enthalten einen zusätzlichen Non-Bottleneck-1D-Block. Exakte Werte sind in Tabelle A.14 (Seite 162) enthalten. Zugehörige Verarbeitungsfrequenzen befinden sich in Tabelle A.15 (Seite 162)

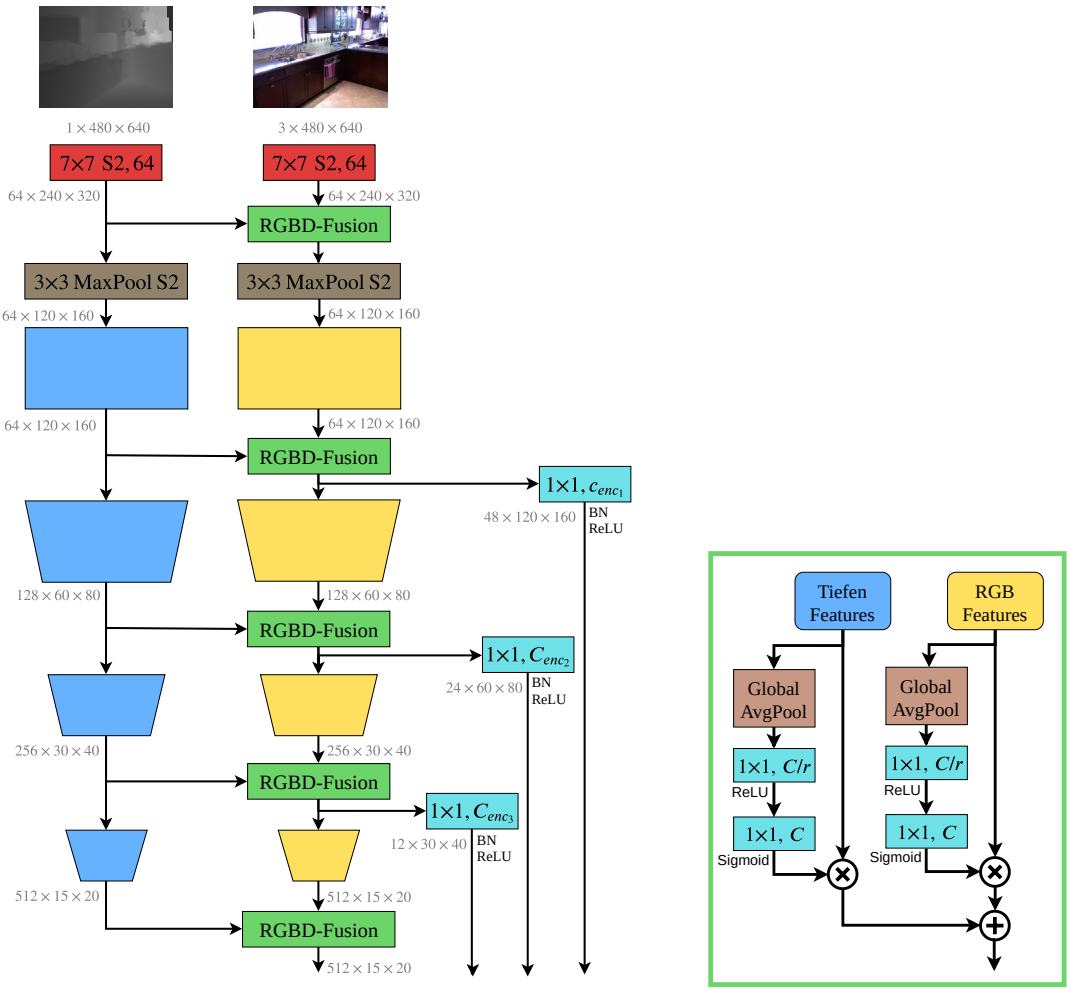
minimal ist. Daraus kann geschlossen werden, dass die pixelweise Wichtung (▪▪) keinen wirklichen Vorteil gegenüber der kanalweisen Wichtung (▪▪ ▪▪) bringt. Auch die aufwendigere 3×3 Convolution (▪▪) kann in Anbetracht der Ergebnisse durch eine einfachere 1×1 Convolution (▪▪) ersetzt werden.
Außerdem ist zu erkennen, dass die Segmentierungsleistung durch Verwendung des dritten Encoders (● ● ●) nicht deutlich besser wird. Dies kann allerdings auch auf die

letzte Fusion am Ende der Encoder zurückzuführen sein: Von den beiden modalitätsspezifischen Encodern kommen jeweils 512 Feature Maps. Dahingegen ist der Einfluss des dritten Encoders mit nur 12 Feature Maps sehr gering, wie in Abbildung 6.15a zu erkennen ist. Ein breiterer dritter Encoder würde allerdings die Inferenzzeit deutlich erhöhen.

Zusammenfassend lässt sich sagen, dass die einfachere kanalweise Wichtung (▪▪) auf dem SUN RGB-D Datensatz ähnliche Ergebnisse wie die aufwendigere pixelweise Wichtung (▪▪) nach SSMA [Valada et al., 2019] erzielt. Für das nächste Experiment dient diese Variante daher als Baseline. Bei Betrachtung der Tabelle 4.1 (Seite 57) fällt zudem auf, dass das SSMA nicht im oberen Bereich der erzielten mIoUs liegt. Daher stellt sich nun die Frage, ob auch die Fusion von RGB und Tiefe im Decoder durch eine andere – eventuell bessere – Fusionsart ersetzt werden sollte.

6.4.2 Fusion der Tiefen-Features in den RGB-Encoder.

Bisher wurden RGB- und Tiefen-Features erst für den Decoder fusioniert. Die beiden Encoder verarbeiteten ihre Features unabhängig voneinander. Erst am Ende der beiden Encoder und später über Skip Connections in den Decoder wurden RGB- und Tiefen-Features fusioniert. Aufgrund der guten Ergebnisse des RedNets [Jiang et al., 2018] soll nun stattdessen eine Fusion der Tiefen-Features in den RGB-Encoder getestet werden (siehe Abbildung 6.17a). Anstatt – wie im RedNet – die Tiefen-Features direkt in den RGB-Encoder zu addieren, sollen die Features vorher gewichtet werden. Dadurch kann gelernt werden, welche Features des Tiefen-Encoders hilfreich für den RGB-Encoder sind.
Die prinzipielle gewichtete Fusionsstrategie von RGB und Tiefe für den Decoder – Konkatenation, Wichten, Kanalreduktion – könnte theoretisch auch für die Fusion der Tiefen-Features in den RGB-Encoder angewandt werden. In Vorexperimenten zeigte sich allerdings, dass deutlich schlechtere Ergebnisse erzielt werden, wenn diese Fusionsstrategie auf die Fusion von Tiefe in den RGB-Encoder angewandt wird. Dies kann auf die letzte Convolution zur Kanalreduktion zurückzuführen sein: Die Grundidee der ResNets besteht in den durchgehenden Shortcuts vom ersten bis zum letzten Residual-Block. Diese durchgehenden Shortcuts werden bei Verwendung der bisheri-



(a) Aufbau der beiden Encoder mit Fusion **(b)** RGBD-Fusion

Abbildung 6.17: Fusion der Tiefen-Features in den RGB-Encoder – Architektur
Vom Tiefen-Encoder werden in jeder Auflösungsstufe Features in den RGB-Encoder fusioniert. Die RGB- und Tiefen-Features werden zunächst jeweils mit einem Squeeze-and-Excitation-Modul gewichtet und anschließend addiert. Die resultierenden Feature Maps werden vom RGB-Encoder weiter verarbeitet. Für die Skip Connections in den Decoder werden Feature Maps abgespalten. Anzahl an Output-Kanälen der 1×1 *Convolutions:*
Bei Konkatenation im Decoder (•••): $C_{enc_1} = 48$, $C_{enc_2} = 24$, $C_{enc_3} = 12$.
Bei Addition im Decoder (•••): $C_{enc_1} = 128$, $C_{enc_2} = 256$, $C_{enc_3} = 512$.
(Vergleiche hierzu die Basisarchitektur in Abbildung 5.1, Seite 61)

gen Fusionsstrategie mit der letzten Convolution zur Kanalreduktion aufgebrochen. Dadurch entstehen Nachteile für den Gradientenfluss und das Training wird erschwert. Daher werden stattdessen die RGB- und Tiefen-Features zunächst jeweils mit einem Squeeze-and-Excitation-Modul gewichtet und anschließend elementweise addiert, wie in Abbildung 6.17b dargestellt ist.

Wie im RedNet erfolgt die Fusion der Tiefen-Features in den RGB-Encoder vor jeder Auflösungsreduktion und am Ende der beiden Encoder, wie in Abbildung 6.17a zu erkennen ist. Im Gegensatz zur bisherigen Netzwerkarchitektur (siehe Abbildung 5.1, Seite 61) findet die erste Fusion daher bereits nach der ersten 7×7 Convolution statt. Insgesamt werden RGB und Tiefe damit fünf mal fusioniert. Bisher wurden RGB und Tiefe lediglich vier mal fusioniert. Einmal am Ende der Encoder und drei mal über Skip Connections in den Decoder. Um den Decoder – und die Fusion der Encoder-Features in den Decoder – zunächst weiterhin so beibehalten zu können, werden über 1×1 Convolutions die Feature Maps über Skip Connections an den entsprechenden Stellen auf 48, 24 bzw. 12 Kanäle reduziert (•••), wie in Abbildung 6.17a rechts zu erkennen ist.

In einer zweiten Variante (•••) sollen die Encoder Feature Maps mit den Decoder Feature Maps addiert werden. Aufgrund des aktuellen Decoder-Aufbaus[13], müssen die fusionierten RGBD Feature Maps des Encoder hierfür mit einer 1×1 Convolution auf 128, 256 bzw. 512 Kanäle verdoppelt werden.

Um einen Vergleich mit dem RedNet zu ziehen, sollen in einem weiteren Experiment (•••) sowohl die Fusion von RGB und Tiefe als auch die Fusion von Encoder Features in den Decoder aus einer reinen Addition bestehen. Das heißt, die vorherige Wichtung von RGB und Tiefe mit dem Squeeze-and-Excitation-Modul entfällt in dieser dritten Variante.

In Abbildung 6.18 sind die erzielten Ergebnisse dieser drei Varianten dargestellt. Bereits die erste Variante mit gewichteter Konkatenation im Decoder (•••) erzielt deutlich bessere Ergebnisse als die Baseline. Daraus lässt sich schließen, dass die Fusion von Tiefen-Features in den RGB-Encoder sinnvoller ist als, wenn RGB und Tiefe ausschließlich für die Verwendung im Decoder fusioniert werden. Werden die Encoder-Features in den Decoder addiert (•••) kann nochmals eine Verbesserung der mIoU erzielt werden.

[13] Der Decoder besteht aus drei Decoder-Modulen (siehe Tabelle 6.10b und Abbildung 6.12a). In jedem Decoder-Modul werden die Encoder Features Maps direkt nach dem Hochskalieren der Decoder Feature Maps in den Decoder fusioniert. Erst in der ersten Convolution des nächsten Decoder-Modul wird die Kanalanzahl halbiert.

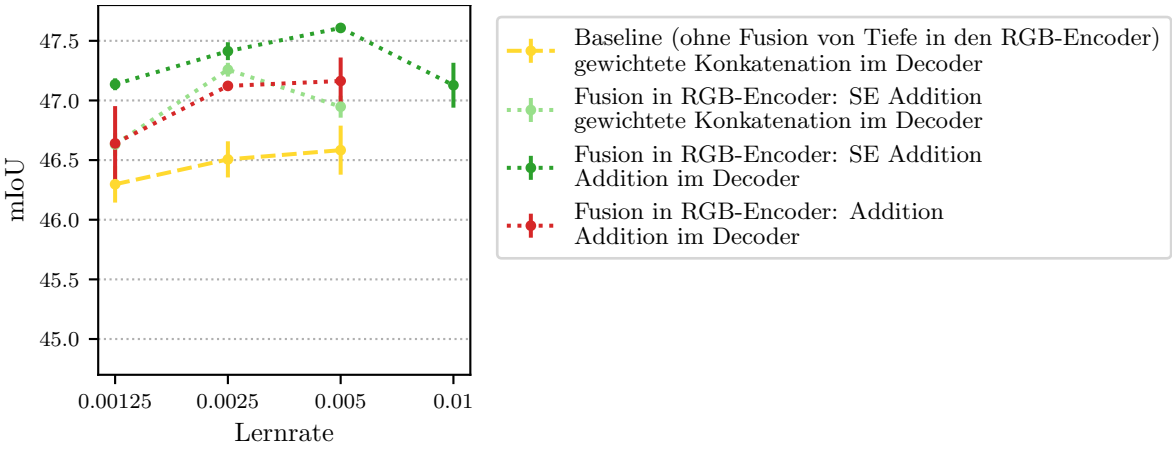


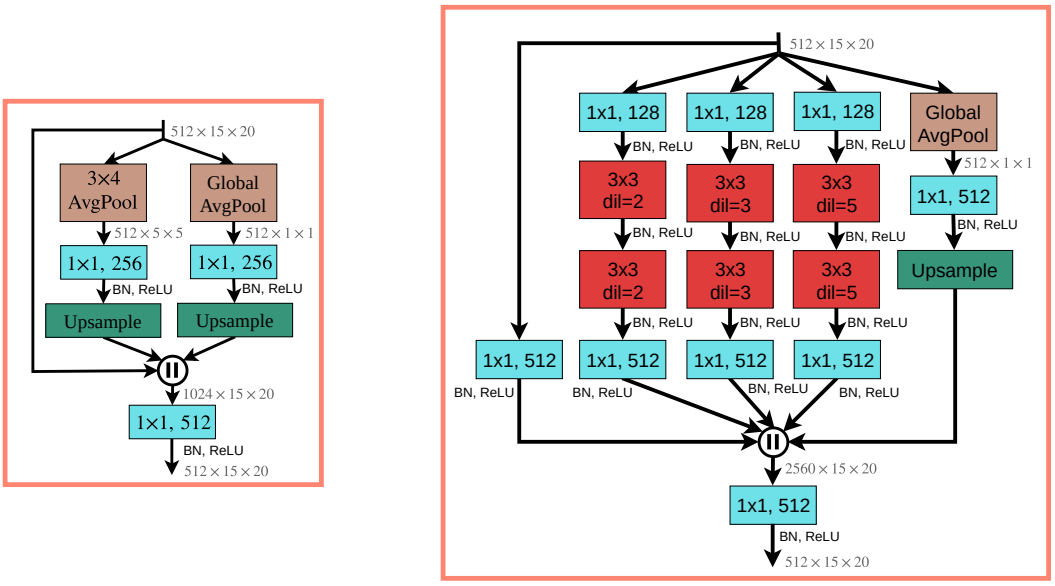
Abbildung 6.18: Fusion der Tiefen-Features in den RGB-Encoder – mIoU
In allen Varianten kommen als Encoder zwei ResNet34 mit Non-Bottleneck-1D-Blöcken zum Einsatz. Die Kanalanzahl im Decoder ist abnehmend. Die Decoder-Module enthalten einen zusätzlichen Non-Bottleneck-1D-Block. In Tabelle A.14 (Seite 162) sind exakte Werte enthalten. Zugehörige Verarbeitungsfrequenzen befinden sich in Tabelle A.15 (Seite 162).

Dies kann sich folgendermaßen begründen lassen: Bei der Addition ist der Informationsfluss von Encoder-Features in den Decoder durch die erhöhte Anzahl an Feature Maps deutlich größer. Außerdem wird bei der Addition der Gradient nur betragsmäßig geteilt. Dahingegen findet bei der Konkatenation eine räumliche Teilung des Gradienten statt. Die Addition im Decoder scheint somit gegenüber der Konkatenation von nur wenigen Feature Maps Vorteile für den Gradientenfluss zu haben und damit ein besseres Training zu ermöglichen.
Aufgrund des deutlichen Aufwärtstrends der Segmentierungsleistung bei höheren Lernraten wurde hierfür ein zusätzliches Training mit der Lernrate 0.01 durchgeführt. Jedoch konnte damit keine höhere mIoU erreicht werden.
Zudem ist zu erkennen, dass die reine Addition von RGB und Tiefe ohne vorherige Wichtung (•••) schlechtere Ergebnisse liefert, als wenn diese jeweils vorher mit einem Squeeze-and-Excitation-Modul gewichtet werden (•••). Dies bestätigt die Annahme, dass über die Wichtung gelernt werden kann, welche Features des Tiefen-Encoder hilfreich für den RGB-Encoder und den Decoder sind.

Für die letzten Experimente wird daher die Variante gewählt, in der Tiefe und RGB zunächst mit einem SE-Modul gewichtet und anschließend addiert werden (•••).

6.5 Einfluss des Kontextmoduls

Der letzte, verbleibende – noch nicht untersuchte – Bestandteil der RGBD-Segmentierungsarchitektur ist das Kontextmodul. Bisher wurde ein modifiziertes Pyramid-Pooling-Modul verwendet, welches nochmals in Abbildung 6.19a dargestellt ist. Nun soll untersucht werden, wie sich die Segmentierungsleistung ändert, wenn statt des Pyramid-Pooling-Moduls (•••) das efficient-Atrous-Spatial-Pyramid-Pooling-Modul (eASPP-Modul) (•••) aus [Valada et al., 2019] zum Einsatz kommt – oder wenn kein Kontextmodul (•••) verwendet wird. Das eASPP-Modul ist in Abschnitt 3.5 erklärt und nochmals in Abbildung 6.19b dargestellt.



(a) Pyramid-Pooling-Modul **(b)** eASPP-Modul

Abbildung 6.19: Getestete Kontextmodule
Die angegebenen Kanäle entsprechen der Kanalanzahl bei Verwendung der Konfiguration mit abnehmender Kanalanzahl im Decoder.

Das eASPP-Modul ist rechenaufwendiger, jedoch könnte sich damit die Segmentierungsleistung verbessern. Da die Input Feature Maps des Kontextmoduls nur eine geringe räumliche Auflösung von $H \times W = 15 \times 20$ haben, werden die Dilationraten des eASPP-Moduls angepasst: Anstelle der Dilationraten 6, 12 und 18 des originalen eASPP-Moduls für eine räumliche Auflöusng von $H \times W = 24 \times 48$ werden hier deutlich kleinere Dilationraten von 2, 3 und 5 verwendet. Dadurch wird ermöglicht, dass die

Kernelelemente noch genügend oft auf die Feature Maps aufgesetzt werden können.

In Abbildung 6.20 sind die Segmentierungsleistungen mit den zwei verschiedenen Kontextmodulen und ohne Kontextmodul dargestellt.

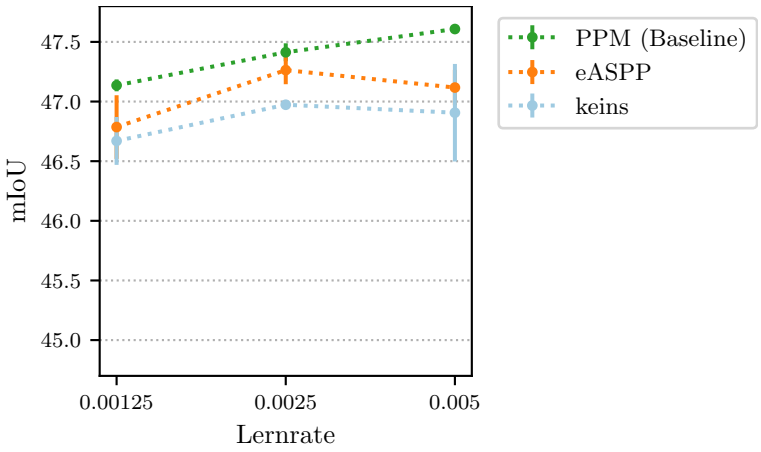


Abbildung 6.20: Variation des Kontextmoduls – mIoU
In allen Varianten kommen als Encoder zwei ResNet34 mit Non-Bottleneck-1D-Blöcken zum Einsatz. Die Kanalanzahl im Decoder ist abnehmend. Die Decoder-Module enthalten einen zusätzlichen Non-Bottleneck-1D-Block. Die Tiefen-Features werden in den RGB-Encoder fusioniert mittels vorheriger Wichtung und anschließender Addition. Die Encoder-Features werden über Skip Connections in den Decoder addiert. Exakte Werte sind in Tabelle A.16 (Seite 163) enthalten. Zugehörige Verarbeitungsfrequenzen befinden sich in Tabelle A.17 (Seite 163).

Es ist zu erkennen, dass ohne Kontextmodul (•••) die mittlere mIoU bei allen Lernraten knapp 0.5 Prozentpunkte schlechter ist als mit dem Pyramid-Pooling-Modul (•••). Daraus lässt sich schließen, dass Kontextmodule im Allgemeinen durchaus dabei helfen, eine bessere Segmentierungsleistung zu erzielen.

Es ist jedoch auch zu erkennen, dass mit dem eASPP-Modul (•••) schlechtere Ergebnisse als mit dem Pyramid-Pooling-Modul (•••) erzielt werden. Eine mögliche Begrüdung könnte sein, dass die verwendeten Dilationraten für die geringe räumliche Auflösung noch immer zu hoch sind. Außerdem enthält das eASPP-Modul im Vergleich zum Pyramid-Pooling-Modul keinen direkten Shortcut.

Eventuell könnten noch bessere Ergebnisse erzielt werden, wenn die beiden Kontextmodule zu einem neuen Kontextmodul kombiniert werden.

6.6 Betrachtung des besten Netzwerks

In diesem Abschnitt soll das Netzwerk, welches die beste mIoU erzielt hat, näher betrachtet werden. Zunächst werden in Abschnitt 6.6.1 die Netzwerkarchitektur des besten Netzwerks und die zugehörigen Trainingsparameter nochmals zusammengefasst.

Anschließend wird in Abschnitt 6.6.2 die Segmentierungsleistung auf den Bildern der einzelnen Kameras des SUN RGB-D Testdatensatzes untersucht und ein Blick auf die IoU-Werte der einzelnen Klassen geworfen. Daraus wird ersichtlich, welche Klassen bereits gut segmentiert werden können und bei welchen Klassen noch Probleme auftreten. Außerdem wird die Konfusionsmatrix näher untersucht, womit häufig verwechselte Klassen ausfindig gemacht werden können.

Nach dieser quantitativen Beurteilung erfolgt in Abschnitt 6.6.3 eine qualitative Einschätzung der Segmentierung anhand konkreter Beispielbilder aus dem SUN RGB-D Testdatensatz. Anschließend wird in Abschnitt 6.6.4 ein Vergleich mit dem State of the Art gezogen. Zuletzt wird in Abschnitt 6.6.5 die Segmentierung auf Aufnahmen der Kinect2 eines Roboters des Fachgebiets Neuroinformatik und Kognitive Robotik der TU Ilmenau betrachtet.

6.6.1 Netzwerkarchitektur und Trainingsparameter

Netzwerkarchitektur. Die beste Netzwerkarchitektur ist nochmal in Abbildung 6.21 dargestellt. Sie besteht aus zwei ResNet34-Encodern für die RGB- und Tiefenbilder, wobei die ResNet-Blöcke durch Non-Bottleneck-1D-Blöcke (siehe Abbildung 6.7a, Seite 92) ausgetauscht wurden. Die Features des Tiefen-Encoders werden in jeder Auflösungsstufe in den RGB-Encoder fusioniert. Dafür werden sowohl RGB- als auch Tiefen-Features zunächst mit einem Squeeze-and-Excitation-Modul gewichtet und anschließend addiert. Als Kontextmodul wird ein modifiziertes Pyramid-Pooling-Modul verwendet. Der Decoder besteht aus drei Decoder-Modulen, in denen auch jeweils ein Non-Bottleneck-1D-Block vorkommt. Die Kanalanzahl nimmt im Decoder Schritt für Schritt ab. Zuletzt werden die Features mit einer 3×3 Convolution mit 37 Output Feature Maps auf die Klassen des SUN RGB-D Datensatzes abgebildet.

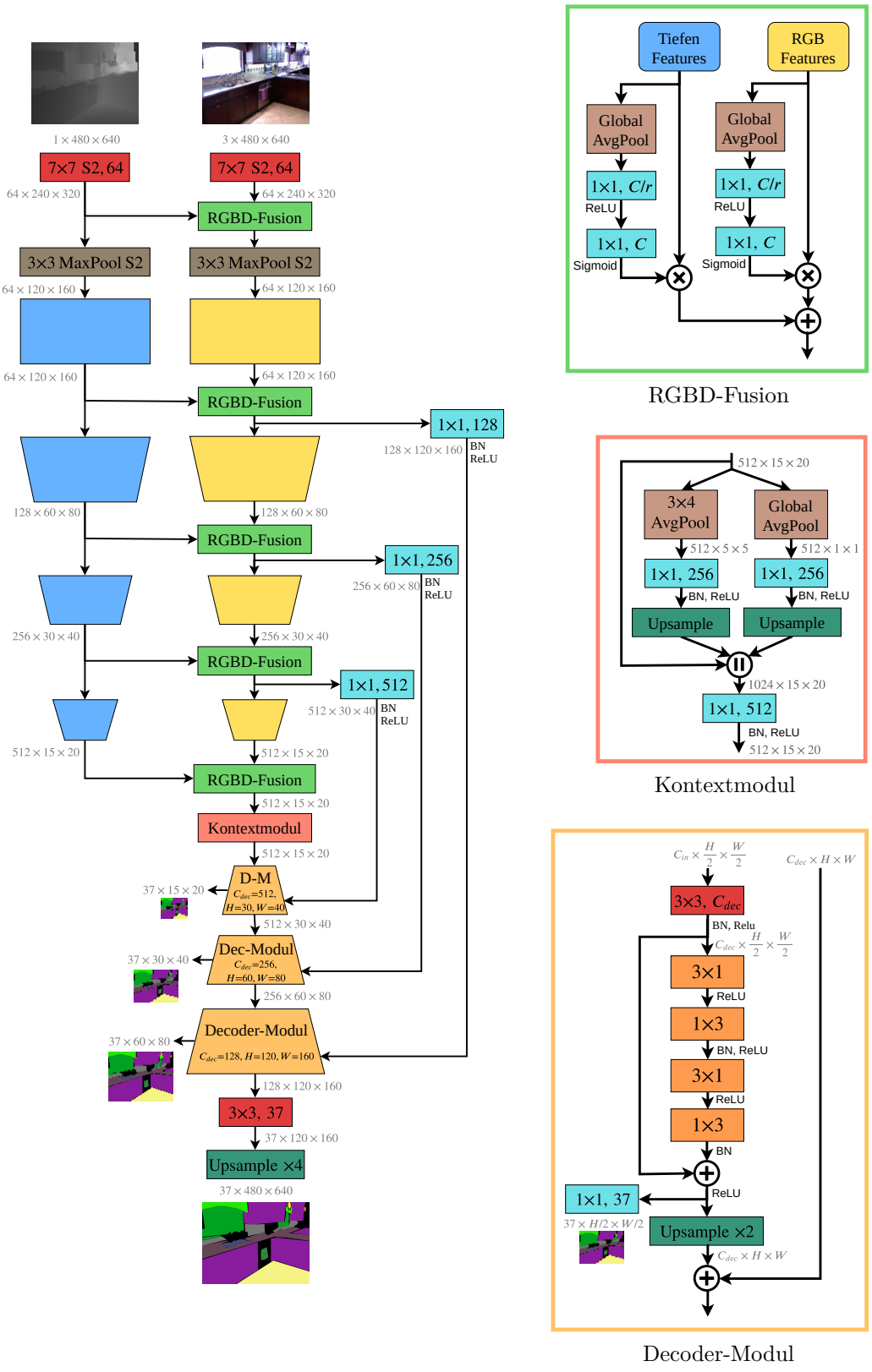


Abbildung 6.21: Finale Netzwerkarchitektur zur effizienten RGBD-Segmentierung

Trainigsparameter. Das Netzwerk wurde mit dem Optimierer SGD mit einem Nesterov-Momentum von 0.9 trainiert. Hierfür wurde der Lernraten-Scheduler OneCycle aus PyTorch verwendet. Dabei steigt die Lernrate für 50 Epochen auf die maximale Lernrate von 0.005 an und fällt daraufhin für 450 Epochen ab, wie in Abbildung 5.6 (Seite 75) zu erkennen ist. Die Batchsize wurde auf 8 gewählt.[14] Auf einer GTX 1080 Ti dauert eine Epoche inklusive Validierung etwa acht Minuten. Davon benötigt das Training etwa $^2/_3$ und die Validierung entsprechend $^1/_3$ der Zeit.

6.6.2 Quantitative Beurteilung

Mit der eben beschriebenen Netzwerkarchitektur erreicht das beste Netzwerk bei einer Lernrate von 0.005 in Epoche 303 eine maximale mIoU von 47.62 auf dem SUN RGB-D Testdatensatz. Zunächst soll nun betrachtet werden, wie sich die mIoU auf den Bildern der einzelnen Kameras unterscheidet. Diese Werte sind in Tabelle 6.5 enthalten.

	Epoche 303	Maximum
Kinect2	45.76 (303)	45.76 (303)
Kinect1	52.34 (303)	52.73 (258)
Xtion	42.06 (303)	42.12 (354)
Realsense	31.77 (303)	33.30 (371)

Tabelle 6.5: mIoU der einzelnen Kameras
In der mittleren Spalte ist die mIoU für die Epoche angegeben, in der insgesamt die höchste mIoU erzielt wurde. In der rechten Spalte ist die jeweils höchste erzielte mIoU mit entsprechender Epoche angegeben.

Dabei wird ersichtlich, dass auf den Bildern der einzelnen Kameras die maximal erzielte mIoU nicht immer in der gesamt besten Epoche erreicht wird. Es ist außerdem zu erkennen, dass auf den Bildern der Kinect1 die beste mIoU erzielt wird, und dass die Bilder der Realsense deutlich schlechter segmentiert werden. Um nachzuvollziehen, ob

[14] Weitere Details zu den Trainingsparametern sind in Abschnitt 5.7 zu finden.

sich dieser Trend über alle Klassen wiederspiegelt – oder ob einzelne Klassen in den unterschiedlichen Kameras besonders gut bzw. besonders schlecht segmentiert werden – sind in Abbildung 6.22 die IoU-Werte pro Klasse abgetragen.

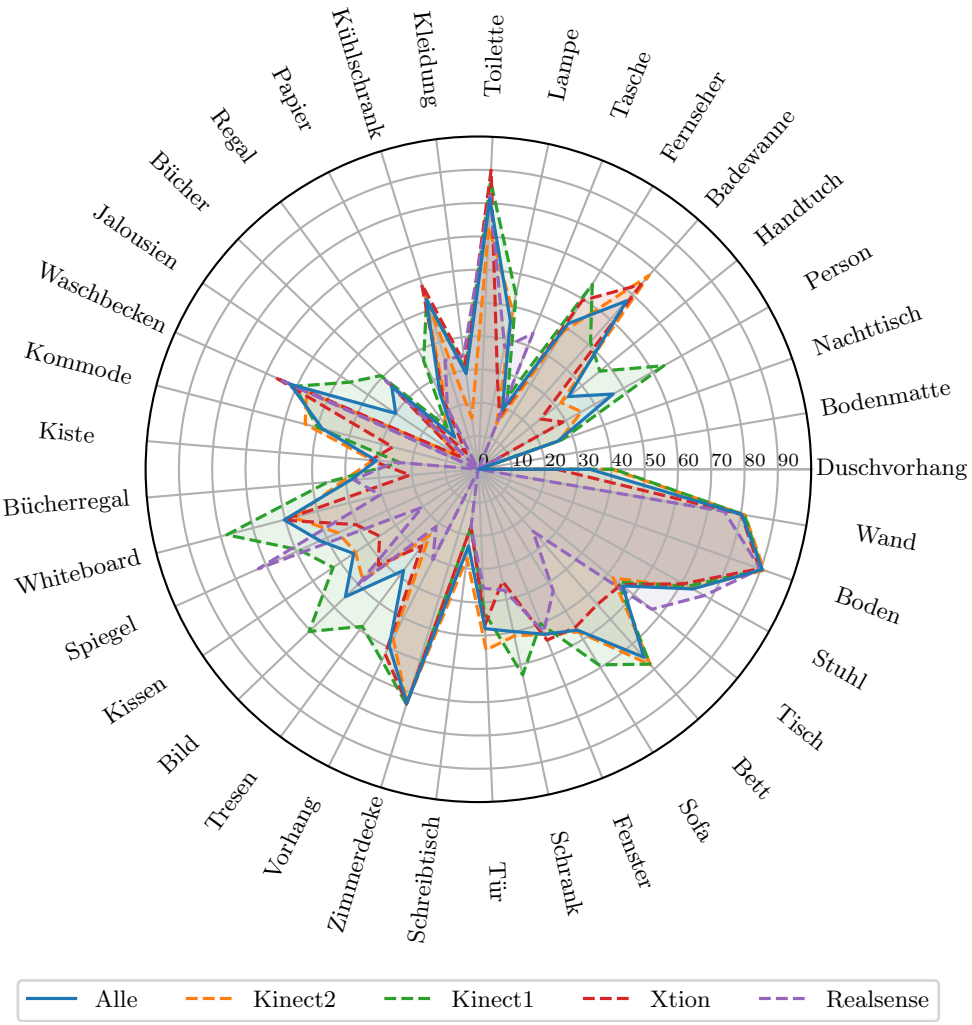


Abbildung 6.22: IoU pro Klasse nach Kamera
Angaben in Prozent. Die Klassen sind nach der Klassenwahrscheinlichkeit im Trainingsdatensatz absteigend im Uhrzeigersinn, beginnend bei Wand, angeordnet.

Insgesamt sind die IoUs bei den Bildern der Kinect1 über fast alle Klassen hinweg höher als bei der Kinect2 und diese höher als bei der Xtion. Bei den Bildern der Realsense liegt die IoU bei vielen Klassen deutlich näher an 0.
Dennoch folgen die IoUs der Kinect1, Kinect2 und Xtion einem ähnlichen Verlauf. Bei den Klassen, bei denen mit den Bildern der Kinect2 eine hohe (niedrige) IoU erzielt wird,

bei den Klassen ist im Allgemeinen auch die IoU auf den Bildern der anderen beiden Kameras vergleichsweise hoch (gering).

Bei näherer Betrachtung lässt sich erkennen, dass bei den Bildern der Kinect1 bei einigen Klassen wie Tresen, Bild, Jalousien oder Person eine erheblich höhere IoU erzielt wird als bei den Bildern der anderen Kameras. Wenn hierzu die Klassenverterteilung pro Kamera in Abbildung A.22 (Seite 164) hinzugezogen wird, ist zu erkennen, dass diese Klassen bei den Bildern der Kinect1 deutlich häufiger vorkommen; und damit wahrscheinlich besser segmentiert werden können. Im Allgemeinen ist die Klassenverteilung bei den Bildern der Kinect1 ausgewogener. Dies wirkt sich positiv auf die Segmentierungsleistung aus. Im Gegenteil dazu ist die Klassenverteilung bei den Bildern der Realsense sehr unausgewogen. Manche Klassen wie Fernseher, Nachttisch oder Duschvorhang kommen überhaupt nicht in den Bildern der Realsense-Kamera vor. Andere Klassen wie Bett, Zimmerdecke oder Jalousien sind im Verlgeich zu den Bildern anderer Kameras stark unterrepräsentiert. Dies drückt die IoU schnell gegen 0 und damit auch die mIoU als Mittelung über alle IoUs.

Zudem fällt auf, dass im Allgemeinen die Klassenhäufigkeit nicht direkt mit der Segmentierungsleistung korreliert. Die Klassen Toilette oder Badewanne werden beispielsweise zuverlässig segmentiert, obwohl sie nur sehr selten vorkommen. Dahingegen erzielen andere, häufigere Klassen wie Schreibtisch oder Regal eine deutlich schlechtere mIoU. Um zu untersuchen, woher diese schlechte schlechte Segmentierungsleistung stammt – und welche Klasse das Netzwerk stattdessen prädiziert –, wird die Konfusionsmatrix in Abbildung 6.23 näher betrachtet.

Zuerst ist die deutliche Hauptdiagonale zu erkennen; das Netzwerk segmentiert also die meisten Klassen richtig. Der größte Wert abseits der Hauptdiagonalen ist bei der Klasse Bodenmatte zu finden. In nahezu allen Fällen prädiziert das Netzwerk stattdessen die Klasse Boden. Allerdings kommt die Klasse Bodenmatte auch kaum im Datensatz vor; die Klasse Boden kommt dem daher noch am nähesten. Weiterhin befindet sich ein hoher Wert abseits der Hauptdiagonalen bei der Klasse Schreibtisch. Stattdessen prädiziert das Netzwerk häufig Tisch und auch die umgekehrte Verwechslung ist in der Konfusionsmatrix zu finden. Auch hier ist keine wirkliche Schwäche zu erkennen. Weitere hohe – aber verständliche – Missklassifikationen kommen bei der Klasse Duschvorhang (Vorhang) und Nachttisch (Tisch, Kommode) vor. Zudem werden nachvollziehbar die Klassen Bücher, Regal und Bücherregal miteinander verwechselt.

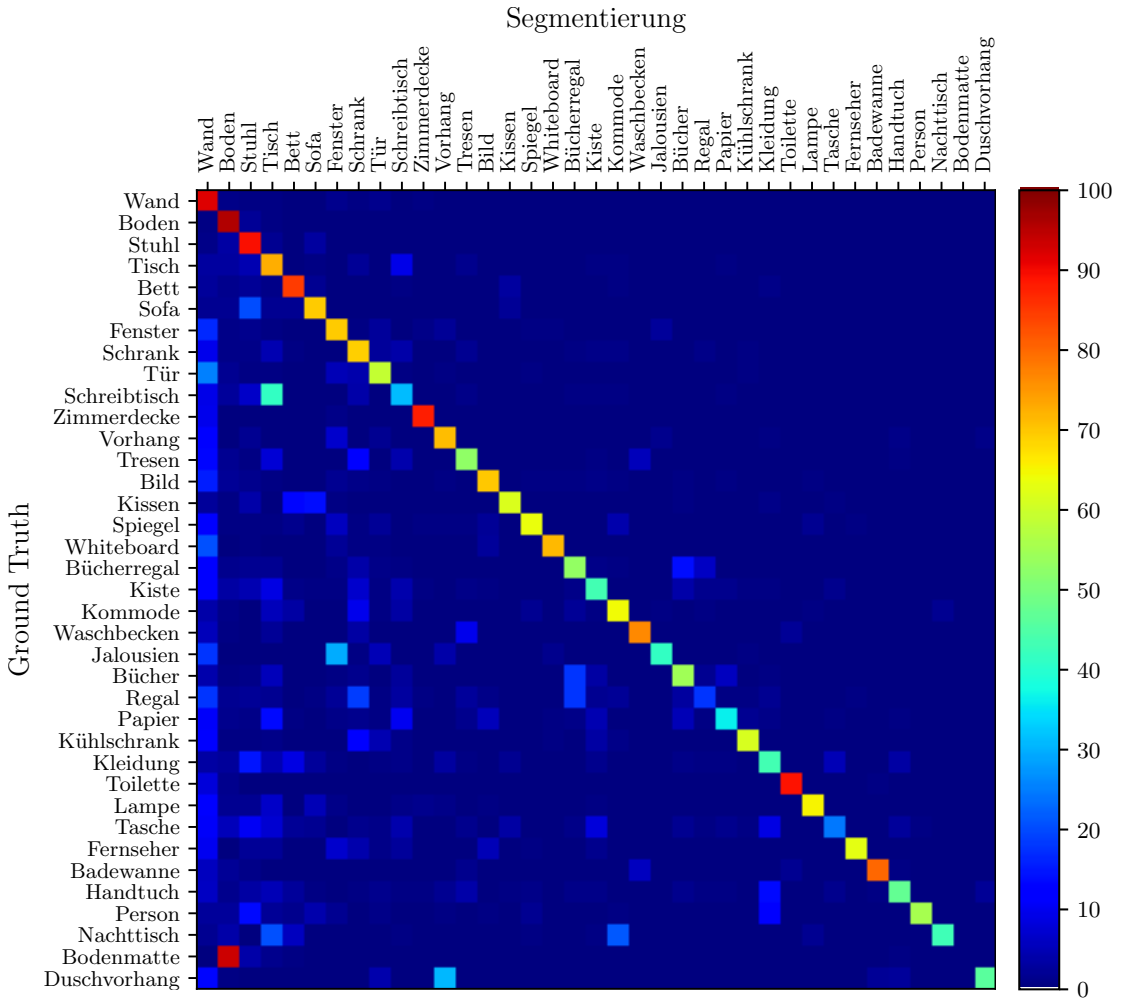


Abbildung 6.23: Konfusionsmatrix des besten Netzwerks

Die Konfusionsmatrix des besten Netzwerks ist in einer Heatmap dargestellt. Die Werte wurden anhand der Anzahl der Ground-Truth-Pixel pro Klasse normiert; das heißt die Zeilensummen ergeben 100%. Die Klassen sind nach der Klassenwahrscheinlichkeit sortiert. Häufige Klassen sind dabei links bzw. oben zu finden.

Die bisher erste, wirkliche Fehlklassifkation ist bei der Klasse Jalousien (Fenster) zu finden. Weiterhin werden Taschen, Handtücher oder Personen zum Teil als Kleidung klassifiziert. Auch hier lässt sich jedoch nicht ausschließen, dass es Labelfehler bei der Ground-Truth-Segmentierung gibt.

Außerdem ist zu erkennen, dass im Allgemeinen mehr Fehlklassifkationen im Bereich links neben der Hauptdiagonalen vorkommen. Im Zweifel werden also häufig vorkommende Klassen wie Wand, Stuhl, Tisch oder Schrank prädiziert.

6.6.3 Qualitative Beurteilung

Nach der quantitativen Beurteilung der Segmentierungsleistung des besten Netzwerk erfolgt nun eine qualitative Beurteilung anhand ausgewählter Segmentierungen auf den Testbildern des SUN RGBD-D Datensatzes.

Hierfür sind in Abbildung 6.24 besonders gute, und in Abbildung 6.25 eher schlechte Segmentierungen dargestellt. Hierbei sei erwähnt, dass die Auswahl an gut segmentierten Bildern deutlich höher war als die Auswahl an schlecht segmentierten Bildern. Schlechte Segmentierungen kommen vor allem in den Bereichen vor, die in der Ground-Truth-Segmentierung der Klasse Void zuzuordnen sind. Für die Berechnung der mIoU – oder auch anderen Bewertungsmaßen – spielen diese Bereiche allerdings keine Rolle.

Im Allgemeinen ist jedoch die gute Generalierungsfähigkeit des Netzwerks erkennbar. Bereiche, die in der Ground-Truth-Segmentierung der Klasse Void zugeordnet werden, werden vor allem in den Bildern in Abbildung 6.24 sinnvollen Klassen zugeordnet. Beispiele hierfür sind das Bett in Zeile 4, das Sofa in der vorletzten und die Bücherregale in der letzten Zeile.

Außerdem ist zu erkennen, dass selbst kleine Objekte, wie die Lampe in der zweiten Zeile oder der hintere Nachttisch in der vierten Zeile, sehr gut segmentiert werden können – und das, obwohl das Netzwerk ab einer räumlichen Auflösung von $H \times W = 120 \times 160$ nur noch bilinear hochskaliert. Obwohl die Qualität des Tiefenbilds in der vorletzten Zeile sehr schlecht ist, kann das Netzwerk dennoch genügend Information aus dem RGB-Bild ziehen und eine gute Segmentierung berechnen. In der vorletzten Zeile ist das vordere Sofa in der Ground-Truth-Segmentierung als Stuhl gelabelt. Dennoch ordnet das Netzwerk hier die eigentlich korrekte Klasse Sofa zu.

Bei den schlecht segmentierten Bildern in Abbildung 6.25 ist zu erkennen, dass das Netzwerk vor allem bei dunklen Bereichen (erste und zweite Zeile), ungewöhnlichen Aufnahmewinkeln (Zeile 4 und 6) oder bei schlechter Bildqualität (dritte, vorletzte und letzte Zeile) noch Probleme hat. Die Szenen in Zeile 5 und 6 sind außerdem deutlich unordentlicher, wodurch eine Segmentierung erschwert wird. Dennoch kann positiv hervorgehoben werden, dass in Zeile 5 die Wand hinter dem Regal noch richtig erkannt wird.

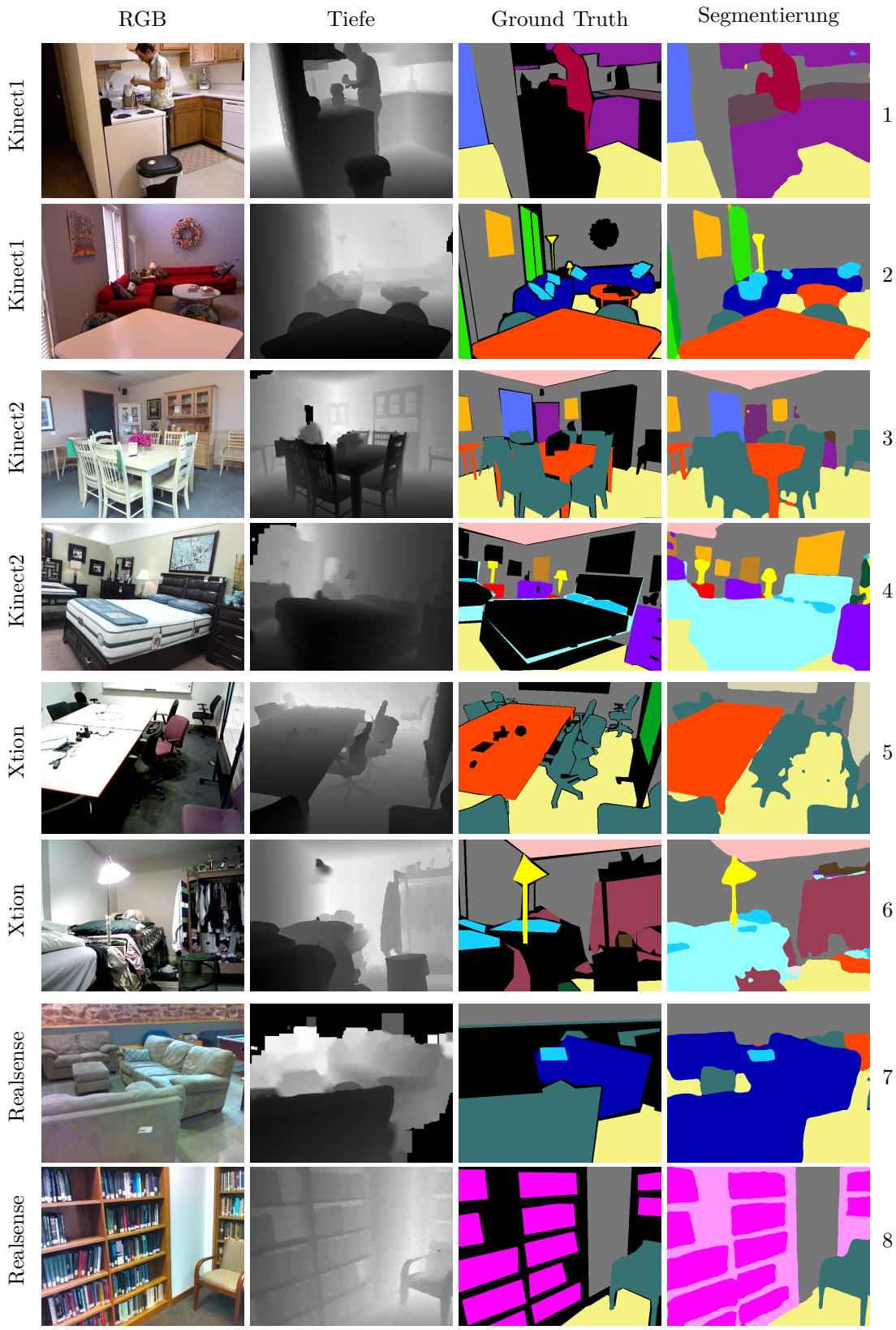


Abbildung 6.24: Gut segmentierte Bilder
Mit Kontrasterhöhung für Tiefenbilder. Colormap siehe Abbildung 6.29 (Seite 129).

Abbildung 6.25: Schlecht segmentierte Bilder
Mit Kontrasterhöhung für Tiefenbilder. Colormap siehe Abbildung 6.29 (Seite 129).

6.6.4 Vergleich mit dem State of the Art

Um die erzielte Segmentierungsleistung einordnen zu können, erfolgt nun ein Vergleich mit dem State of the Art. Da vom RedNet [Jiang et al., 2018] und vom ACNet [Hu et al., 2019] PyTorch-Implementierungen vorhanden sind, werden für diese beiden Netzwerkarchitekturen auch Inferenzzeiten bestimmt. Zum RedNet liegen zusätzlich trainierte Gewichte vor. Damit kann die, im Paper genannte, mIoU durch eine eigene Validierung überprüft werden. Außerdem können Segmentierungen visuell verglichen werden.
In Tabelle 6.6 sind die erzielte mIoU-Werte und die Verarbeitungsfrequenzen von ausgewählten Netzwerken angegeben. Für diese Tabelle wurden Verfahren ausgewählt, welche in dieser Masterarbeit häufiger thematisiert wurden. Alle drei Netzwerke verwenden ResNet50-Encoder. Bei anderen Netzwerken mit tieferen Encodern ist die Verabeitungsfrequenz voraussichtlich nochmals deutlich geringer. Netzwerke mit VGG16-Encoder erzielen schlechtere Segmentierungsleistungen. Die mIoUs von diesen anderen Netzwerken sind in Tabelle 4.1 (Seite 57) zu finden.

	mIoU		**FPS**		
	original	**480×640**	**PyTorch**	**TensorRT Float32**	**TensorRT Float16**
SSMA [Valada et al., 2019]	45.73	-	-	-	-
ACNet [Hu et al., 2019]	48.10	-	4.0	5.4	16.5
RedNet [Jiang et al., 2018]	47.38	47.75	5.2	7.3	23.6
eigenes Netzwerk	47.62	48.00	8.6	13.2	35.5

Tabelle 6.6: Vergleich mit dem State of the Art
Vegleich mit ausgewählten Verfahren zur RGBD-Segmentierung. Für jedes Netzwerk ist die mIoU und – bei vorhandener PyTorch-Implementierung – die Verarbeitungsfrequenz angegeben. Bei vorhandenen Gewichten wurde die mIoU einmal ermittelt, wenn die Netzwerkausgabe auf die Größe der Testbilder skaliert wird (original) und einmal, wenn die Ground-Truth-Segmentierung auf die Netzwerkausgabe skaliert wird (480×640).

Bei der Validierung des RedNets ist aufgefallen, dass mit der, in dieser Masterarbeit verwendeten, Validierungsstrategie nur eine mIoU von 47.38 im Gegensatz zu, einer im Paper genannten, mIoU von 47.8 erreicht wird. Wird stattdessen die Ground-Truth-

Segmentierung auf die Netzwerkausgabe (480×640) skaliert, wird eine mIoU von 47.75 erreicht, was gerundet dem Wert aus dem Paper entspricht.
Aus diesem Grund wurde auch für das eigene Netzwerk die mIoU mit dieser anderen Berechnungsmethode bestimmt. Es ist zu erkennen, dass mit beiden Berechnungsmethoden das, in dieser Masterarbeit entworfene, Netzwerk eine höhere mIoU erreicht als das RedNet, obwohl die beiden Encoder nur auf einem ResNet34 basieren. Dennoch ist die Verarbeitungsfrequenz des eigenen Netzwerks maßgeblich höher.
Beim SSMA und beim ACNet wird angenommen, dass die mIoU mit der Originalgröße der Testbilder ermittelt wurde. Es ist zu erkennen, dass das eigene Netzwerk eine deutlich höhere mIoU erreicht als das SSMA. Mit dem ACNet ist die erzielte mIoU höher, was sich allerdings in der geringen Verarbeitungsfrequenz niederschlägt.
Insgesamt kann daraus geschlossen werden, dass das eigene Netzwerk eine ähnliche Segmentierungsleistung zum State of the Art erzielt. Dennoch ist die Verarbeitungsfrequenz im Vergleich deutlich höher, was den Einsatz auf einem mobilen Roboter ermöglicht.
Mit den bereitgestellten Gewichten des RedNets wurden außerdem die Bilder des SUN RGB-D Testdatensatzes segmentiert und eingefärbt. Einige Segmentierungen des eigenen Netzwerks und des RedNets können in Abbildung 6.26 visuell miteinander verglichen werden. Es fällt auf, dass die Segmentierungen vom RedNet Gridding-Artefakte[15] enthalten. Dies ist auf die Transposed Convolutions zum Hochskalieren im Decoder zurückzuführen. Da die eigene Netzwerkarchitektur weder Transposed Convolutions noch Dilated Convolutions enthält, sind die Segmentierungen frei von Gridding-Artefakten.

[15] Gridding-Artefakte werden in Abschnitt A.2.1 erklärt

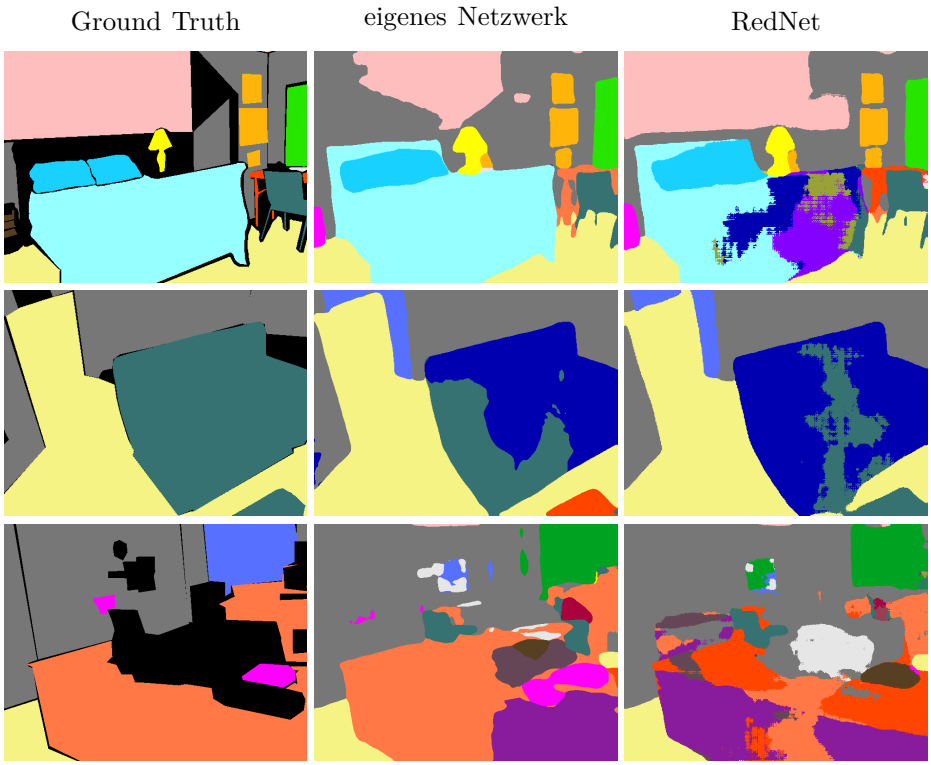


Abbildung 6.26: Vergleich der Segmentierung mit dem RedNet
Segmentierungen auf dem SUN RGB-D Testdatensatz. Im Gegensatz zum RedNet kommen bei der Segmentierung des eigenen Netzwerks keine Gridding-Artefakte vor. Colormap siehe Abbildung 6.29.

6.6.5 Anwendung auf eigenen Roboteraufnahmen

Zuletzt soll die Generalisierungsfähigkeit des Netzwerks auf neue Bilder anhand von Aufnahmen der Kinect2 eines Roboters des Fachgebiets NIKR beurteilt werden. Da hier keine Ground-Truth-Segmentierung vorhanden ist, kann lediglich eine visuelle Beurteilung der Segmentierungsleistung erfolgen.
Bisher wurden alle Netzwerke auf den Tiefenbildern mit ausgefüllten Lücken trainiert.[16] Wie bereits in Abschnitt 5.2.3 erwähnt, stehen in der Anwendung keine „verbesserten“ Tiefenbilder zur Verfügung. Daher wird die Netzwerkarchitektur, welche die besten Ergebnisse erzielt hat (siehe Abbildung 6.21), nochmals mit den originalen Tiefenbildern trainiert und validiert. Mit beiden Netzwerken werden anschließend die Roboteraufnahmen segmentiert. Dadurch kann eine Einschätzung erfolgen, was für die Anwendung besser geeignet ist.

[16] Die Entstehung dieser verbesserten Tiefenbilder wird in Abschnitt 5.2.3 erklärt.

Für die Datenvorverarbeitung wurden hierfür erneut Mittelwert und Standardabweichung der originalen Tiefenbilder bestimmt. Für die Bestimmung dieser Werte wurden die ungültigen Tiefenwerte (Wert = 0) ausmaskiert. Nach der Normalisierung werden diese zurück auf 0 gesetzt. Damit wird sichergestellt, dass das Netzwerk aus diesen ungültigen Werten keine Information ziehen kann.
Die erzielte mIoU mit den originalen Tiefenbildern ist in Abbildung 6.27 dargestellt.

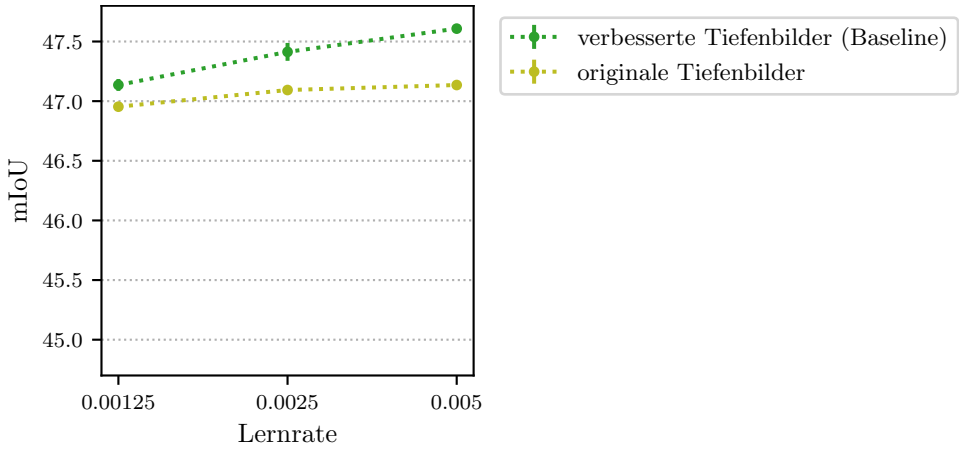


Abbildung 6.27: Verbesserte vs. originale Tiefenbilder – mIoU
Für die Anwendung stehen keine verbesserten Tiefenbilder zur Verfügung. Daher wurde die beste Netzwerkarchitektur nochmals mit den originalen Tiefenbildern trainiert. Exakte Werte befinden sich in Tabelle A.18 (Seite 163).

Es ist zu erkennen, dass mit den originalen Tiefenbildern (•••) auf dem SUN RGB-D Datensatz schlechtere Ergebnisse erzielt werden als mit den verbesserten Tiefenbildern (•••). Ob dies auch auf die Roboteraufnahmen zutrifft, soll visuell beurteilt werden.
Die Kinect2 des Roboters nimmt ein RGB-Bild mit der Auflösung $H \times W = 1080 \times 1920$ und ein Tiefenbild mit der Auflösung 424×512 auf. Diese beiden Bilder müssen zunächst zueinander registriert werden.[17] Hierfür kann entweder das Tiefenbild auf das RGB-Bild – oder das RGB-Bild auf das Tiefenbild – gemappt werden. Letzteres ist für die Anwendung deutlich effizienter. Allerdings enthält danach das RGB-Bild Lücken an den Stellen, an denen auch das Tiefenbild Lücken enthält. Um einzuschätzen womit die beste Segmentierungsleistung erzielt wird, werden daher vier verschiedene Varianten getestet, welche in Tabelle 6.7 enthalten sind. Die Segmentierungen dieser vier Varianten sind von links nach rechts in Abbildung 6.28 dargestellt.

[17] Die Registrierung von RGB- und Tiefenbild wird in Abschnitt 2.4.2 erklärt.

		Mapping	
		Tiefe → RGB	RGB → Tiefe
		$H \times W = 1080 \times 1920$	$H \times W = 424 \times 512$
trainiert auf	verbesserte Tiefenbilder	1	2
	originale Tiefenbilder	3	4

Tabelle 6.7: Varianten für die Segmentierung der Roboteraufnahmen
Die Segmentierungen der vier Varianten sind von links nach rechts in Abbildung 6.28 dargestellt.

Es ist zu erkennen, dass mit allen Varianten prinzipiell eine gute Segmentierung erzielt werden kann – wenn auch merklich schlechter als auf den Bildern des SUN RGB-D Datensatzes. Des Weiteren wird offensichtlich, dass das Netzwerk lediglich diejenigen Objekte gut segmentieren kann, welche auch im Datensatz vorkommen. Dennoch kann das Netzwerk generalisieren und Kontextinformationen einbeziehen: Die Klasse Fahrrad kommt beispielsweise nicht im Datensatz vor, weshalb das Netzwerk diese Klasse nicht kennen kann. In der zweiten Zeile werden den Fahrradpixeln daher Klassen der umliegenden Pixel zugeordnet.
Über alle Bilder hinweg ist die Segmentierung etwas besser, wenn das Netzwerk auf den originalen Tiefenbildern trainiert wurde. Es ist auch zu erkennen, dass die Segmentierung auf den kleinen Bildern (zweite und vierte Spalte) im Allgemeinen ähnlich gut ist, wie auf den großen Bildern. Dies ist für die Anwendung wichtig, da so die Tiefenbilder nicht erst hochskaliert werden müssen, um sie dann für die Netzwerkeingabe wieder herunterzuskalieren. Bei den kleinen Bildern in der zweiten und vierten Spalte entsteht oben und unten im Bild ein Bereich, an dem das RGB-Bild keine Werte enthält. Trotzdem kann das Netzwerk aus den Tiefenwerten in diesen Bereichen hilfreiche Informationen ziehen und eine sinnvolle Segmentierung berechnen. Allerdings wird die Segmentierung an Bereichen mit vielen Lücken deutlich schlechter, was vor allem in den Bildern der dritten Zeile zu erkennen ist.

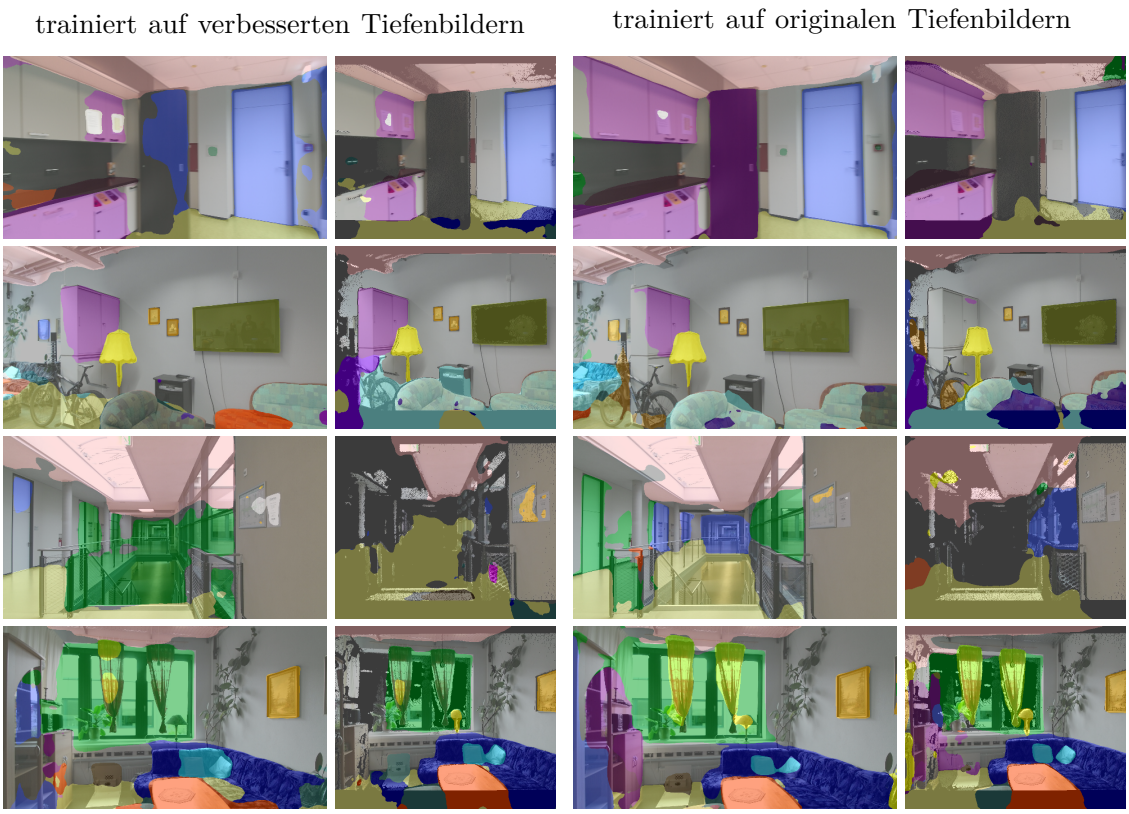


Abbildung 6.28: Anwendung auf Roboteraufnahmen
Erste und dritte Spalte: Die Tiefenbilder wurden auf die RGB-Bilder gemappt. Zweite und vierte Spalte: Die RGB-Bilder wurden auf die Tiefenbilder gemappt.

Abbildung 6.29: Colormap
Verwendete Colormap für den SUN RGB-D Datensatz. Ähnlichen Klassen wurden ähnliche Farben zugeordnet. Die Klassen sind nach der Klassenwahrscheinlichkeit im Trainingsdatensatz sortiert. Diese Colormap ist hilfreich für die Interpretation der Abbildungen 1.1c, 5.3, 6.24, 6.25 und 6.26.

Kapitel 7

Zusammenfassung und Ausblick

Abschließend soll diese Masterarbeit und die daraus gewonnen Erkenntnisse zusammengefasst werden. Außerdem wird ein Ausblick für weitere mögliche Experimente gegeben.

7.1 Zusammenfassung

Das Ziel dieser Masterarbeit war das Entwerfen und Trainieren eines Deep-Learning-basierten Verfahrens zur effizienten RGBD-Segmentierung von Indoor-Szenen für die Anwendung auf einem mobilen Roboter. Hierfür erfolgte zunächst eine breite Literaturrecherche zum State of the Art zur effizienten Segmentierung in Kapitel 3 und zur RGBD-Segmentierung in Kapitel 4. Aufbauend auf den Erkenntnissen zu beiden Themengebieten wurde daraufhin in Kapitel 5 eine eigene Netzwerkarchitektur entwickelt.
Die Bestandteile dieser Netzwerkarchitektur wurden in Kapitel 6 Schritt für Schritt experimentell untersucht und weiterentwickelt. Zuerst stellte sich in Abschnitt 6.1 heraus, dass mit einem RGBD-basiertem Netzwerk bestehend aus zwei ResNet18-Encodern für die RGB- und die Tiefenbilder eine bessere Segmentierungsleistung und eine höhere Verarbeitungsfrequenz erzielt werden konnte, als mit einem reinen RGB-basierten Netzwerk mit ResNet50-Encoder. Da das RGBD-ResNet34-Netzwerk eine noch höhere Segmentierungsleistung, bei ausreichend hoher Verarbeitungsfrequenz, erzielte, wurde dieses Netzwerk zur Baseline für nachfolgende Experimente.

Durch den Austausch der ResNet-Non-Bottleneck-Blöcke durch effizientere Non-Bottleneck-1D-Blöcke in Abschnitt 6.2 konnte die mean Intersection over Union (mIoU) weiter verbessert werden. Im Anschluss an die Experimente zum Encoder, wurde in Abschnitt 6.3 der Decoder näher untersucht. Mit einem breiteren und tieferen Decoder konnte die Segmentierungsleistung nochmals erhöht werden. Daraufhin wurde in Abschnitt 6.4 die prinzipielle Fusionsart von RGB und Tiefe untersucht. Es stellte sich heraus, dass die gewichtete Fusion der Tiefen-Features in den RGB-Encoder bessere Ergebnisse erzielte, als wenn RGB und Tiefe erst für den Decoder fusioniert werden. Danach wurde in Abschnitt 6.5 gezeigt, dass das verwendete Kontextmodul sinnvoll ist – und ohne Kontextmodule schlechtere Ergebnisse erzielt werden.
Zuletzt wurde in Abschnitt 6.6 das Netzwerk mit der höchsten mIoU näher untersucht und ein Vergleich mit dem State of the Art gezogen. Mit einer mIoU von 47.62 auf dem Indoor-RGBD-Datensatz SUN RGB-D erzielt das beste Netzwerk vergleichbare Ergebnisse zum State of the Art. Dennoch ist die Verarbeitungsfrequenz mit 13.2 Frames pro Sekunde auf einem NVIDIA Jetson AGX Xavier deutlich höher und ermöglicht somit den Einsatz auf einem mobilen Roboter.

7.2 Ausblick

Für diesen Abschnitt wurden im Verlauf dieser Masterarbeit Ideen gesammelt, die zu einem späteren Zeitpunkt noch untersucht werden können. Einige der Ideen beziehen sich auf die konkrete Netzwerkarchitektur, während andere auf das Training abzielen.

7.2.1 Weitere Ideen zur Netzwerkarchitektur

Austauch der Encoder-Blöcke. In den Experimenten zu den Encoder-Blöcken wurden alle Blöcke des ResNets ersetzt. Stattdessen könnte auch der jeweils erste Block einer Auflösungsstufe beibehalten und lediglich die nachfolgenden Blöcke ersetzt werden. In dem jeweils ersten Block wird die räumliche Auflösung mit einer Strided Convolution halbiert; das heißt, es geht nur jeder zweite Pixel in die Berechnung ein. Ist diese Strided Convolution zusätzlich faktorisiert wie beim Non-Bottleneck-1D-Block oder erhält nur die Hälfte der Kanäle wie beim eigenen Block, könnte sich dies nachteilig auf die Repräsentationskraft auswirken.

Modifikation des eigenen Encoder-Blocks. Der in dieser Masterarbeit entwickelte eigene Encoder-Block (siehe Abbildung 6.7c, Seite 92) teilt zunächst den Input-Tensor in zwei Gruppen. Jede Gruppe erhält die Hälfte der Feature Maps. Im ersten Zweig wird eine normale 3×3 Convolution angewandt. Im zweiten Zweig ist diese 3×3 Convolution zusätzlich Dilated. Anstatt beiden Gruppen die gleiche Anzahl an Feature Maps zu geben, könnte auch die Convolution ohne Dilation mehr Feature Maps erhalten. Damit würde der Fokus mehr auf der lokalen Feature-Extraktion legen. Die Dilated Convolution würde dadurch nur wenige, zusätzliche Kontextinformationen liefern.

Mehr Layer am Ende des Decoders. In der finalen Netzwerkarchitektur gibt es nach der letzten Fusion der Encoder Features in den Decoder nur noch eine einzige 3×3 Convolution, welche die Features auf die Klassen des SUN RGB-D Datensatzes abbildet (siehe Abbildung 6.21, Seite 116). Es könnte sein, dass eine höhere Segmentierungsleistung erzielt wird, wenn der Decoder stattdessen noch mehr Layer nach der letzten Fusion enthält. Außerdem könnten dadurch die Encoder Features etwas später in den Decoder fusioniert werden, wodurch die 1×1 Convolution zur Verdopplung der Kanalanzahl überflüssig wird. Des Weiteren kann untersucht werden, ob sich die Segmentierungsleistung weiter verbessert, wenn im Decoder noch eine Auflösungsstufe weiter hochskaliert wird. Hierfür könnte ein zusätzliches, viertes Decoder-Modul verwendet werden. Anstatt drei Skip Connections vom Encoder in den Decoder wären damit auch vier Skip Connections möglich. Allerdings ist hier zu untersuchen, wie sehr sich das auf die Inferenzzeit auswirkt.

Andere Encoder-Decoder-Fusion. In der finalen Netzwerkarchitektur werden die Encoder Features direkt mit den Decoder Features addiert. Stattdessen könnte auch eine gewichtete Fusion verwendet werden. Hierfür eignet sich beispielsweise das Mutual-Embedding-Upsample-Modul aus Abschnitt 3.4 (Seite 35).

7.2.2 Weitere Ideen für das Training

Pre-Training mit höherer Auflösung. Die Encoder des Netzwerks wurden zunächst auf ImageNet vortrainiert. Wie üblich wurde hierfür eine Inputauflösung von $H \times W = 224 \times 224$ verwendet. Das RGBD-Segmentierungsnetzwerk verwendet allerdings eine deutlich höhere Inputauflösung von $H \times W = 480 \times 640$.

Das führt dazu, dass die auf ImageNet-vortrainierten Encoder sich zunächst an die höhere Auflösung anpassen müssen. Es könnte daher sinnvoll sein zumindest für einige Epochen auf ImageNet mit der höheren Auflösung nachzutrainieren.

Pre-Training auf SceneNet. Anstatt auf ImageNet vorzutrainieren, könnte das Pre-Training auch auf dem synthetischen Indoor-RGBD-Datensatz Scene RGB-D erfolgen. Das hätte den Vorteil, dass bereits das gesamte Netzwerk – anstatt nur die Encoder – vortrainiert werden. Außerdem kann das Netzwerk hier bereits die Aufgabe der Segmentierung erlernen.

Focal Loss. Anstelle der Klassengewichtung könnte auch Focal Loss [Lin et al., 2020] verwendet werden. Beim Focal Loss gehen Pixel, die mit einer hohen Klassenzugehörigkeit richtig klassifiziert werden, weniger stark in die Berechnung ein. Das hat zur Folge, dass sich das Training mehr auf die schwierigen Pixel fokussiert. Somit gehen implizit häufig vorkommende – und damit einfach zu klassifizierende – Klassen weniger stark in die Loss-Berechnung ein.

Anhang A

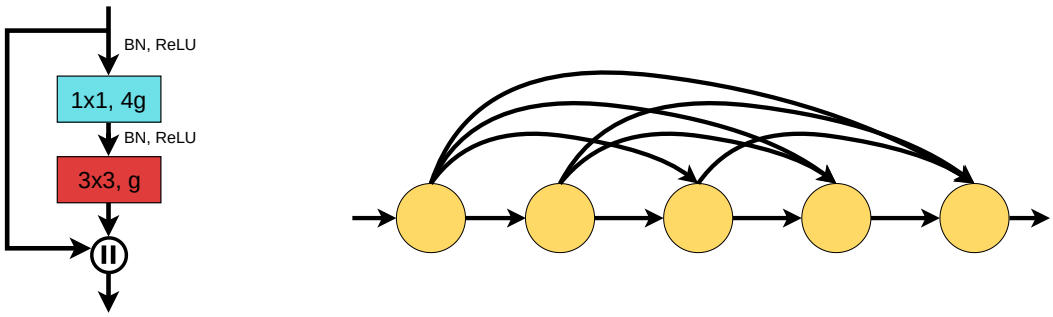
Ergänzende Unterlagen

A.1 Weiterführende Grundlagen

Dieser Abschnitt erweitert die, in Kapitel 2 vorgestellten, Grundlagen.

A.1.1 DenseNet

Das DenseNet [Huang et al., 2017] ist eine Erweiterung des ResNets. Der wesentliche Unterschied zum ResNet-Block besteht in der Konkatenation am Ende eines Layers anstelle der elementweisen Addition. Das hat zur Folge, dass der Output jedes Layers als Input für alle nachfolgenden Layer dient. In jedem Layer werden lediglich g zusätzliche Feature Maps berechnet. Der Parameter g gibt somit das Wachstum der Kanalanzahl an. Die Anzahl der Feature Maps steigt somit mit jedem Layer. Ein Layer bezeichnet hierbei – anders sonst üblich – die Abfolge: BN-ReLU-Conv-BN-ReLU-Conv. Dabei ist die erste Convolution eine 1×1 Convolution, um die Kanalanzahl auf $4g$ zu reduzieren. Die zweite Convolution mit einer Kernelgröße von 3×3 reduziert die Kanalanzahl weiter auf g Outputkanäle. Der entstehende Output-Tensor wird anschließend mit dem Input-Tensor des Layers konkateniert. Das Layer eines Dense-Blocks ist in Abbildung A.1a dargestellt. Ein Dense-Block besteht aus mehreren solcher Layer (siehe Abbildung A.1b). Zwischen zwei Dense-Blöcken werden sogenannte *Transition Layer* verwendet, um die räumliche Auflösung reduzieren. Des Weiteren findet im Transition Layer eine Kompression der Kanalanzahl statt.



(a) Layer eines Dense-Blocks **(b)** Dense-Block

Abbildung A.1: DenseNet

(a) Im sogenannten Layer eines Dense-Blocks werden beide Convolutions zur Kanalreduktion verwendet. Das Layer berechnet nur g zusätzliche Feature Maps. Im Gegensatz zum ResNet-Block wird der Input-Tensor mit dem Output-Tensor konkateniert.

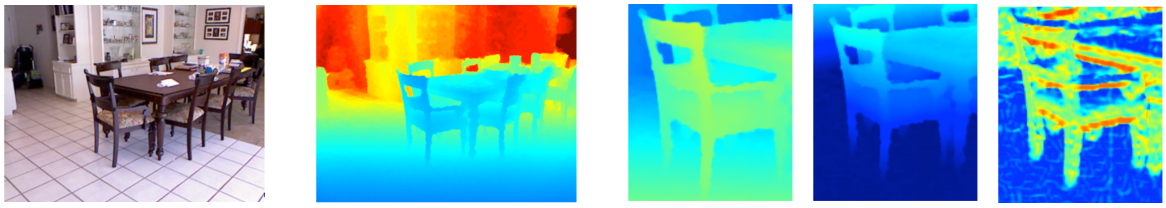
(b) Ein Dense-Block besteht aus mehreren solcher Layer. Durch die Konkatenation am Ende eines Layers dient jeder Output eines Layers als Input aller nachfolgenden Layer. (Grafik angelehnt an [Huang et al., 2017])

A.1.2 HHA-Codierung der Tiefenbilder

Im Gegensatz zu RGB-Bildern sind Tiefenbilder nur einkanalig. Die Tiefenbilder enthalten außerdem eine völlig andere Statistik als die RGB-Bilder. Dadurch wird es schwieriger vortrainierte Basisnetzwerke zu verwenden.[1] In [Gupta et al., 2014] wird daher eine sogenannte *HHA*-Codierung vorgeschlagen. Das einkanalige Tiefenbild wird dabei in eine dreikanalige geozentrische Repräsentation umgewandelt. Die drei Kanäle sind die horizontale Disparität (engl. Horizontal disparity), die Höhe über dem Boden (engl. Height above the ground) und der Winkel der Oberflächennormale zur angenommenen Gravitationsrichtung (engl. Angle of the pixel's local surface normal to the inferred gravity direction). Alle drei Kanäle werden linear auf den Wertebereich $[0, 255]$ skaliert, um der 8-Bit-Repräsentation von RGB-Bildern zu entsprechen. Die HHA-codierten Tiefenbilder sollen dadurch mehr den RGB-Bildern ähneln und damit

[1] In aktuellen Publikationenen zur semantischen Segmentierung wird der Encoder zumeist zuerst auf ImageNet [Russakovsky et al., 2015] vortrainiert. ImageNet ist ein Klassifikationsdatensatz mit 1.2 Millionen Trainingsbildern. Durch das Vortrainieren auf ImageNet erreichen Segmentierungsarchitekturen deutlich bessere Ergebnisse.

besser für vortrainierte Basisnetzwerke geeignet sein. Vor allem bei kleinen Trainingsdatensätzen können laut [Gupta et al., 2014] mithilfe der HHA-codierten Tiefenbilder bessere Ergebnisse erreicht werden. Ein Beispiel für ein HHA-codiertes Tiefenbild ist in Abbildung A.2 zu sehen.



(a) RGB **(b)** Tiefe **(c)** HHA

Abbildung A.2: HHA-Codierung
(a) RGB-Bild (b) zugehöriges Tiefenbild (c) HHA-Codierung eines Ausschnitts aus dem Tiefenbild. von links nach rechts: Disparität, Höhe, Winkel.
Die einkanaligen Bilder sind mithilfe einer Heatmap dargestellt. Dabei kennzeichnet blau niedrige und rot hohe Werte. Bilder aus [Gupta et al., 2014]

Die horizontale Disparität lässt sich direkt aus dem Tiefenbild und mithilfe der Kameraparameter ableiten. Sie ist jedoch invers proportional zur Tiefe. Je größer die horizontale Disparität desto näher befindet sich das Objekt an der Kamera und desto geringer ist der Tiefenwert. Der Algorithmus zur Berechnung der anderen beiden Kanäle stammt aus [Gupta et al., 2013]. Im Indoor-Bereich sind Boden- und Tischflächen horizontal. Schränke und Wände hingegen sind vertikal. Mithilfe dieser Annahmen kann eine Gravitationsrichtung abgeleitet werden. Als initiale Gravitationsrichtung wird die y-Achse angenommen. Diese Gravitationsrichtung wird iterativ verfeinert, indem die Anzahl der Punkte maximiert wird, an denen die lokal geschätzten Oberflächennormalen möglichst ausgerichtet oder orthogonal zur Gravitationsrichtung sind. Anschließend kann daraus für jeden Pixel die Höhe über dem Boden und der Winkel zur Gravitationsrichtung bestimmt werden. Zur Berechnung des Algorithmus sind die Kameramatrix bestehend aus den intrinsischen Kameraparametern, das originale Tiefenbild und das Tiefenbild mit ausgefüllten Lücken notwendig.[2]

[2] Original-Matlab-Quellcode zur Generierung von HHA-Bildern: `https://github.com/s-gupta/rcnn-depth/blob/master/rcnn/saveHHA.m`, zuletzt aufgerufen am 12.01.2020.
Entsprechender Python-Quellcode: `https://github.com/charlesCXK/Depth2HHA-python`, zuletzt aufgerufen am 12.01.2020

A.2 State of the Art: effiziente Segmentierung

A.2.1 Effiziente Encoder-Blöcke

Dieser Abschnitt ergänzt die in Abschnitt 3.2 (Seite 23) vorgestellten Encoder-Blöcke.

Exkurs: Gridding Artefakte Gridding-Artefakte entstehen bei der Transposed Convolution und bei der Dilated Convolution, da benachbarte Output-Pixel aus völlig verschiedenen Pixelsätzen des Inputs berechnet werden. Dies resultiert in einer Inkonsistenz an lokalen Informationen. Beispiele für auftretende Gridding-Artefakte in der Segmentierung sind in Abbildung A.3 zu erkennen.

Abbildung A.3: Gridding-Artefakte
*Beispiele für auftretende Gridding-Artefakte in der Segmentierung bei Verwendung eines mit Dilationraten modifiziertes ResNets. Bilder aus [*Wang *et al., 2018]*

Einige Segmentierungsarchitekturen verwenden als Basisnetzwerk ein modifiziertes ResNet: In späteren Layern wird die Strided Convolution zum Downsampling durch eine Dilated Convolution ausgetauscht.[3] Dadurch wird eine gleichbleibende Vergrößerung des rezeptiven Felds erreicht ohne die räumliche Auflösung zu verringern. Damit auch in den nachfolgenden Layern die rezeptive Feldgröße identisch bleibt, muss die Dilationrate in allen nachfolgenden Convolutions von d=1 auf d=2 verdoppelt werden. Wird eine weitere Strided Convolution durch eine Dilated Convolution ersetzt, so verdoppelt sich die Dilationrate nochmals auf d=4. Dies resultiert in einer Kaskadierung an Dilated Convolutions mit gemeinsamen Teiler als Dilationrate. Bei dieser Konfiguration werden die Gridding-Artefakte allerdings in späteren Layern immer stärker, wie in [Wang et al., 2018] und [Wang und Ji, 2018] erklärt wird. Um Gridding-Artefakte zu

[3] ResNets, in denen Strides durch Dilated Convolutions ersetzt werden kommen zum Beispiel im ICNet [Zhao et al., 2018], im PSPNet [Zhao et al., 2017] oder im DeepLabv3 [Chen et al., 2017] vor.

verringern, können teilerfremde Dilationraten[4] verwendet oder parallele Convolutions mit unterschiedlichen Dilationraten[5] fusioniert werden. Eine weitere Alternative ist es, die Kaskadierung an Dilated Convolutions aufzubrechen und dazwischen Layer mit einer Dilationrate von d=1 zu verwenden. Andere Ansätze – wie [Wang und Ji, 2018] und [Ziegler et al., 2019] – fokussieren sich darauf, das Grundproblem zu lösen, indem die Dilated Convolution selbst geglättet wird.

Efficient-Dense-Asymmetric-Modul (EDA-Modul). Im EDANet von [Lo et al., 2019] wird der Non-Bottleneck-1D-Block um eine 1×1 Convolution zur Kanalreduktion am Anfang erweitert. Die elementweise Addition wird durch die Konkatenation ersetzt. Damit kann das EDA-Modul (dargestellt in Abbildung A.4) als Dense-Unit verwendet werden.

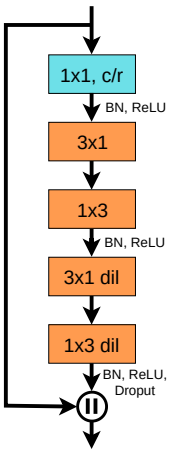


Abbildung A.4: EDA-Modul
Efficient-Dense-Asymmetric-Modul des EDANets [Lo et al., 2019]

Ein Vorteil der Dense-Verbindung ist die Gewinnung von Multiskalen-Informationen. Mit jedem Modul eines Dense-Blocks nimmt die rezeptive Feldgröße zu. Ein Dense-Block enthält jedoch die Outputs aller Module und damit Features mit unterschiedlicher rezeptiver Feldgröße. Es ist zu beachten, dass – im Vergleich zum Non-Bottleneck-1D-Block – das ReLU zwischen den Factorized Convolutions fehlt.

[4] Teilerfremde Dilationraten werden im LEDNet [Wang et al., 2019a] und im ESNet (An Efficient Symmetric Network) [Wang et al., 2019b] verwendet.

[5] Encoder-Blöcke mit parallelen Dilated Convolutions werden in den nachfolgenden Abschnitten vorgestellt.

Depthwise-Feature-Aggregation-Block. Auch der Depthwise-Feature-Aggregation-Block (dargestellt in Abbildung A.5) des MAVNets von [Nguyen et al., 2019] erweitert den Non-Bottleneck-1D-Block und verwendet eine Konkatenation mit dem Shortcut anstelle einer elementweisen Addition. Hierbei ist das erste Factorized-Convolutions-Paar zugleich eine Depthwise Convolution. An das Depthwise Factorized-Convolutions-Paar schließt sich eine Pointwise Convolution für den Informationsaustausch zwischen den Kanälen an (wie bei der Depthwise Separable Convolution). Durch die Konkatenation mit dem Shortcut erhöht sich die Kanalanzahl. Diese kann mit einer, sich daran anschließenden, 1×1 Convolution wieder verringert werden.

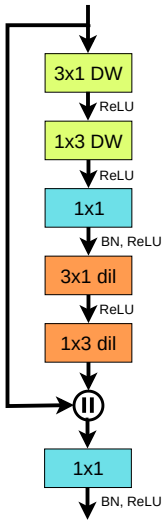


Abbildung A.5:
DwFA-Block
Depthwise-Feature-Aggregation-Block des MAVNets [Nguyen et al., 2019].

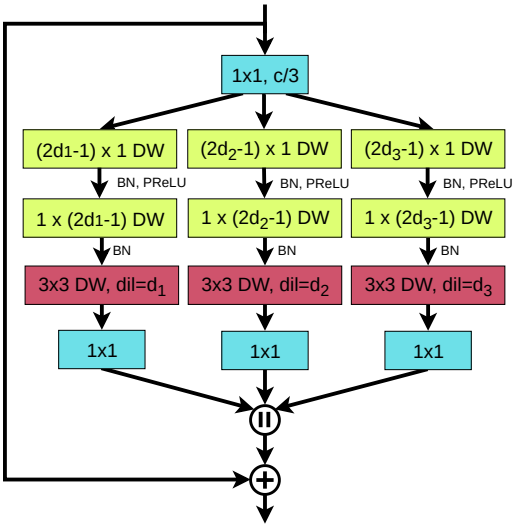


Abbildung A.6: C3-Modul
Das Concentrated-Comprehensive-Convolution-Modul-Modul verwendet Zweige mit Concentrated Comprehensive Convolutions. Dadurch soll der Informationsverlust von Depthwise Dilated Convolutions vermieden werden. Das C3-Modul wurde erstmals in [Park et al., 2018] verwendet und daraufhin im [Park et al., 2019] modifiziert.

Concentrated-Comprehensive-Convolution-Modul (C3-Modul). In [Park et al., 2018] wird das Problem von Depthwise Separable Dilated Convolution dargestellt: Durch die Depthwise Separable Convolution entsteht eine ungenaue

Approximation. Aufgrund der Lücken in der Dilated Convolution gehen zusätzlich Informationen von benachbarten Pixeln verloren, was zu Gridding-Artefakten[6] führen kann. Diese Kombination resultiert in allgemein schlechteren Ergebnissen. Aus diesem Grund führen die Autoren *Concentrated Comprehensive Convolutions* ein, welche in Abbildung A.6 dargestellt sind. In der *Concentration*-Phase wird eine Depthwise Convolution eingesetzt. Dabei ist die Kernelgröße $k = 2 \cdot d - 1$, wobei d die Dilationrate der nachfolgenden Dilated Convolution ist. Die Kernelgröße entspricht also der Größe des gesamten gestreckten Filters der nächsten Convolution. Dies soll den Informationsverlust von benachbarten Pixeln in der nachfolgenden Dilated Convolution vermeiden. Ist die Dilationrate der nachfolgenden Dilated Convolution groß, so wäre die Depthwise Convolution mit dem großen Kernel sehr rechenaufwendig. Aus diesem Grund werden Factorized Depthwise Convolutions eingesetzt. In der nachfolgenden *Comprehensive-Convolution*-Phase wird nun eine Depthwise Dilated Convolution verwendet, um das rezeptive Feld zu vergrößern. Mit einer 1×1 Convolution wird schließlich der Informationsaustausch zwischen den Kanälen wiederhergestellt. Das Concentrated Comprehensive-Convolution-Modul (C3-Modul), dargestellt in Abbildung A.6, baut auf dem ESP-Modul auf. Anstelle der 3×3 Dilated Convolutions wird die vorgestellte Concentrated Comprehensive Convolution verwendet. Die resultierenden Feature Maps werden schließlich (ohne hierarchische Feature Fusion) konkateniert. Während das C3Net noch 4 Zweige mit den Dilationraten 2, 4, 6 und 8 verwendet, werden im ExtremeC3Net [Park et al., 2019] nur noch 3 Zweige mit variablen Dilationraten eingesetzt. Dabei werden in früheren Schichten des Netzwerks geringere und in späteren Schichten höhere Dilationraten verwendet.

Feature-Pyramid-Encoding-Block (FPE-Block). Der Feature-Pyramid-Encoding (FPE) -Block des FPENets [Liu und Yin, 2019] baut auf dem Inverted-Residual-Bottleneck-Block[7] auf und ist in Abbildung A.7 dargestellt.

Wie im Inverted-Residual-Bottleneck-Block wird auch im FPE-Block die Kanalanzahl zuerst mit einer 1×1 Convolution expandiert. Der Expansionsfaktor ist hierbei 4. Anschließend werden die Feature Maps in vier Gruppen aufgeteilt. Auf jede Gruppe wird eine Dilated Depthwise Convolution angewandt. Die Dilationraten der vier Zweige

[6] Gridding-Artefakte sind in Abschnitt A.2.1 erklärt.
[7] Der Inverted-Residual-Bottleneck-Block wird in Abschnitt 3.2.3 erklärt

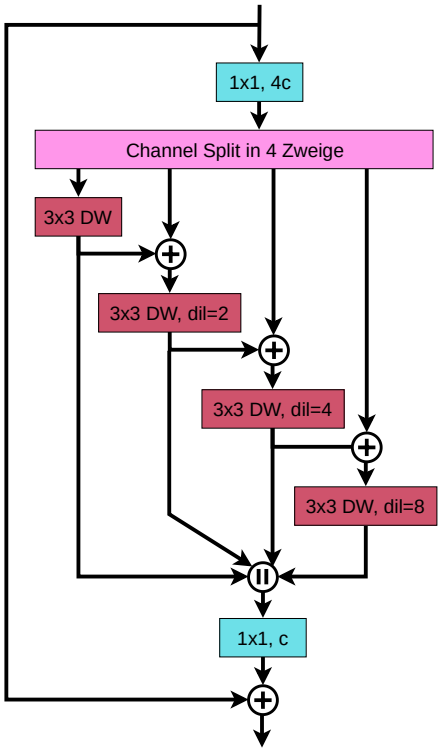


Abbildung A.7: Feature-Pyramid-Encoding-Block (FPE-Block)
*FPE-Block aus dem Encoder des FPENets von [*Liu *und* Yin*, 2019].*

betragen: 1, 2, 4 und 8. Um Gridding Artefakte zu vermeiden wird der Output einer Depthwise Dilated Convolution zum Input der nächsten Depthwise Dilated Convolution elementweise addiert. Die Outputs der 4 Zweige werden anschließend konkateniert und mittels einer 1×1 Convolution zur Kanalreduktion verarbeitet. Der resultierende Output wird zuletzt mit dem Shortcut addiert.

A.2.2 Weitere Fusionierungen von Encoder und Decoder

Ergänzend zu den in Abschnitt 3.4 (Seite 33) vorgestellten Fusionsmöglichkeiten von Encoder Feature Maps in den Decoder werden hier weitere Fusionsmöglichkeiten dargestellt. Die meisten hiervon sind selbsterklärend. Für weitere Informationen wird auf die entsprechende Publikation verwiesen.

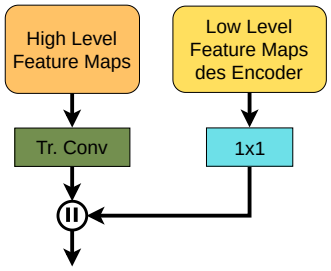


Abbildung A.8: ESPNet (Decoder) *Verwendung von Skip Connections im Decoder des ESPNets [Mehta et al., 2018].*

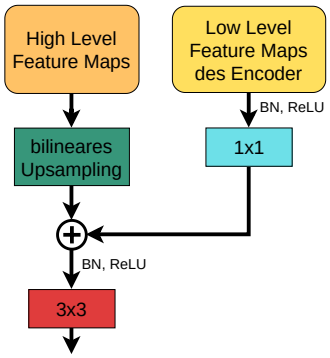


Abbildung A.9: SwiftNet (Decoder) *Verwendung von Skip Connections im Decoder des SwiftNets [Oršić et al., 2019].*

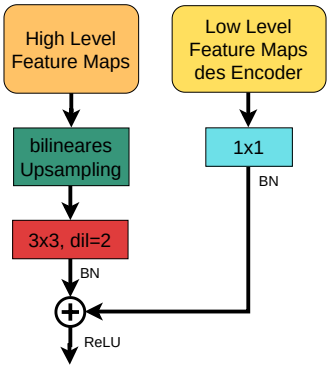


Abbildung A.10: Cascade Feature Fusion *[Zhao et al., 2018]*

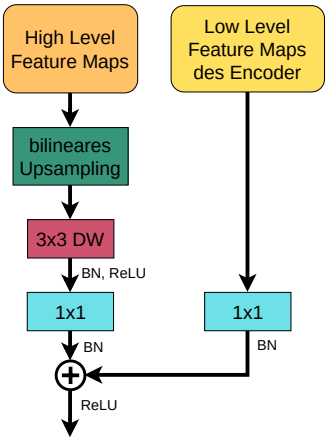


Abbildung A.11: Feature-Fusion-Modul *Fusionierung im Fast-SCNN [Poudel et al., 2019]*

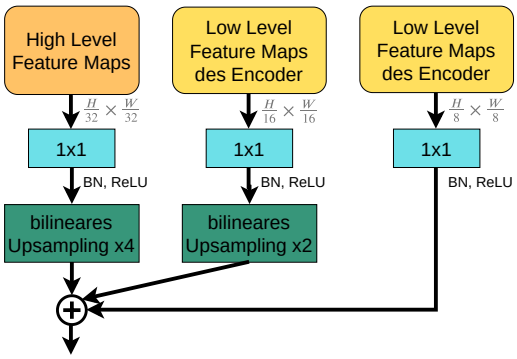


Abbildung A.12: DFANet (Decoder) *[Li et al., 2019b]*

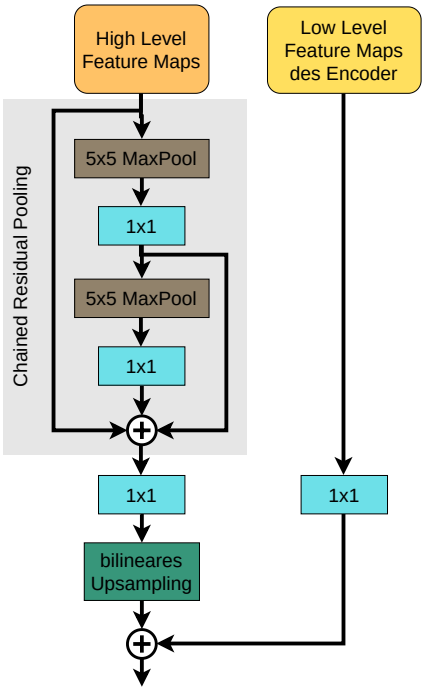


Abbildung A.13: Light-Weight RefineNet (Decoder) *[Nekrasov et al., 2018]*

Fusionierung im BiSeNet. Das *Attention-Refinement-Modul* (dargestellt in Abbildung A.14) im BiSeNet von [Yu et al., 2018] verwendet eine Gewichtung ähnlich wie im Squeeze-and-Excitation-Modul.

Das Attention-Refinement-Modul kommt sowohl für die höher- als auch auf die niedrigerauflösenden Feature Maps (High Level Feature Maps) zum Einsatz. Auf die High Level Feature Maps wird in einem zusätzlichen Zweig ein Global Average Pooling für globale Kontextinformationen angewandt. Der Output dieses Zweigs wird verwendet, um den Output aus dem Attention-Refinement-Modul nochmals zu wichten. Für diese nochmalige Wichtung ist allerdings keine Begründung gegeben. Die resultierenden niedriger- und höherauflösenden Feature Maps werden schließlich bilinear hochskaliert, bevor sie entlang der Kanalachse konkateniert werden. Diese Art der Fusionierung kommt nur ein einziges Mal im BiseNet vor.

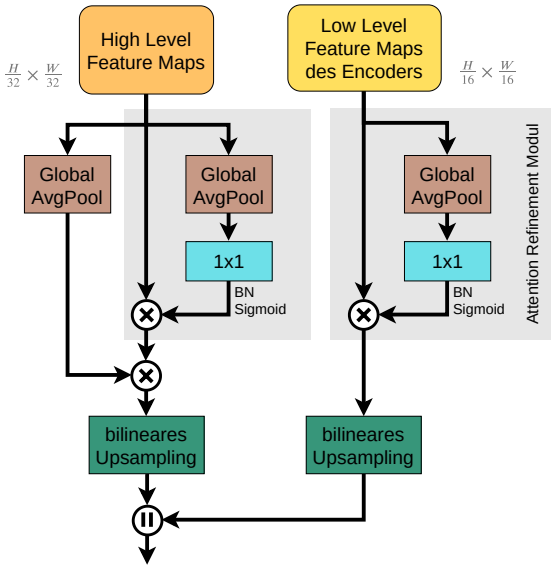


Abbildung A.14: Fusionierung im BiSeNet
Fusionierung von höherauflösenden Low Level Feature Maps des Encoders in die niedrigerauflösenden High Level Feature Maps im BiSeNet aus [Yu et al., 2018]

A.2.3 Effiziente Kontextmodule

Das hier erklärte Kontextmodul ergänzt die in Abschnitt 3.5 (Seite 36) vorgestellten effiziente Kontextmodule.

Attention Pyramid Network. Das Attention Pyramid Network aus dem LEDNet wird von [Wang et al., 2019a] als Decoder bezeichnet. Da dort jedoch alle Operationen vor dem Hochskalieren der räumlichen Auflösung stattfinden, wird das Attention Pyramid Network in dieser Masterarbeit als Kontextmodul eingeordnet. Auch im Attention Pyramid Network, dargestellt in Abbildung A.15, werden Features mit unterschiedlich großem rezeptiven Feld kombiniert.
Auf die Encoder Feature Maps wird in einem Zweig zuerst eine 3×3, eine 5×5 und eine 7×7 Convolution jeweils mit Stride 2 angewandt. Dadurch entsteht eine Feature-Pyramide mit Feature Maps unterschiedlicher Skalenstufen. Jede Skalenstufe wird nun erneut entsprechend mit einer 3×3, 5×5 beziehungsweise 7×7 Convolution verarbeitet. Da die Feature Maps eine geringe räumliche Auflösung haben, ist der Rechenaufwand für die großen Kernels an dieser Stelle nicht allzu hoch. Die verschiedenen Skalenstufen werden anschließend Schritt für Schritt fusioniert. Dafür werden die Feature Maps

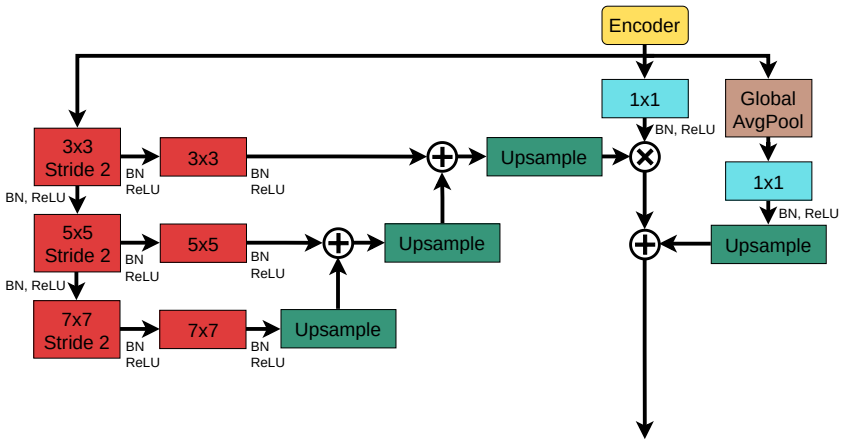


Abbildung A.15: Attention Pyramid Network (APN)
*Attention Pyramid Network aus dem LEDNet von [*Wang *et al., 2019a].*

der jeweils kleineren Skalenstufe bilinear hochskaliert und danach mit den Feature Maps der nächsthöheren Skalenstufe elementweise addiert. Auf die Encoder Feature Maps wird eine 1×1 Convolution angewandt. Das Ergebnis aus der Feature-Pyramide wird nun verwendet, um diese Feature Maps zu wichten. Zusätzlich gibt es noch einen weiteren Zweig bestehend aus Global Average Pooling, 1×1 Convolution und Upsampling, ähnlich wie in den anderen Kontextmodulen. Dieser Zweig extrahiert globale Kontextinformationen und wird anschließend mit den gewichteten Encoder Feature Maps elementweise addiert.

A.3 State of the Art: RGBD-Segmentierung

A.3.1 Fusionierung im Netzwerk

Dieser Abschnitt ergänzt die im Abschitt 4.1 (Seite 46) vorgestellten Fusionsmöglichkeiten im Netzwerk

zwei Netzwerke – Fusion am Ende

Gated Fusion Layer. Das *Gated Fusion Layer* (dargestellt in Abbildung A.16) fusioniert zwei komplette Netzwerke, indem automatisch der Anteil beider Netzwerkzweige an der Klassifizierung einzelner Pixel gelernt wird.

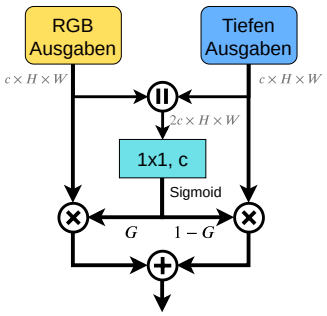


Abbildung A.16: Gated Fusion Layer
Mit dem Gated Fusion Layer wird gelernt, wie stark einzelne Elemente der RGB- beziehungsweise der Tiefen-Ausgabe berücksichtigt werden.

Dafür werden zuerst die Ausgaben des RGB- und Tiefen-Zweigs konkateniert. Mit einer 1×1 Convolution wird die Kanalanzahl anschließend wieder um die Hälfte reduziert. Über die Sigmoid-Ausgabefunktion wird ein Gewichtungstensor $G \in [0, 1]$ erstellt, mit dem die Ausgaben des RGB-Netzwerks gewichtet werden. Die Ausgaben des Tiefen-Netzwerks werden mit $1 - G$ gewichtet. Je größer einzelne Elemente in G sind, desto stärker bezieht das Netzwerk also die RGB-Ausgaben ein. Bei kleinen Elementen in G werden die Tiefen-Ausgaben mehr beachtet. Die beiden resultierenden Ausgaben werden anschließend für die finale Ausgabe elementweise addiert.

zwei Netzwerke – Austausch von Features

Hier werden die in Abschnitt 4.1.2 (Seite 48) angedeuteten Verfahren zum Austausch von Features zwischen zwei getrennten Netzwerken vorgestellt.

Feature Transformation Network. In [Wang et al., 2016] wird für den Feature-Austausch ein *Feature Transformation Network* vorgestellt, welches in Abbildung A.17 dargestellt ist.
Im Feature Transformation Network werden sowohl gemeinsame Features als auch modalitätsspezifische Features berechnet. Die gemeinsamen Features enthalten Informationen, die von beiden Modalitäten gemeinsam genutzt werden. Dahingegen enthalten

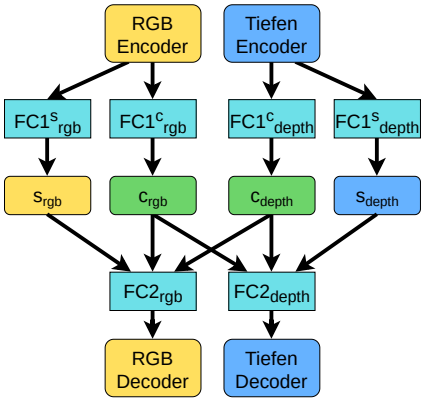


Abbildung A.17:
Feature Transformation Network
*Austausch an Features zwischen dem RGB- und dem Tiefennetzwerk in [*Wang *et al., 2016]. c (common) steht für gemeinsame Features. s (specific) steht für modalitätsspezifische Features. FC sind Fully Connected Layer.*

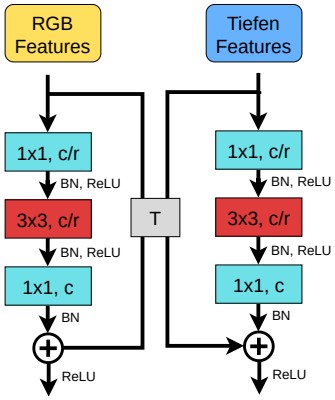


Abbildung A.18:
Idempotentes Mapping
*Über idempotentes Mapping können die beiden Netzwerke Informationen austauschen. Abbildung nach [*Xing *et al., 2019b].*

die modalitätsspezifischen Features Informationen, die jeweils nur in einer der beiden Modalitäten zu sehen sind. Das Feature Transformation Network wird zwischen den beiden Encodern und Decodern eingesetzt. Die beiden Encoder schließen mit einer 15×20 Convolution ab, um die Feature Maps der Größe $C \times H \times W = 512 \times 15 \times 20$ auf die Größe 4096×1×1 zu bringen. Nun folgen an jeden Encoder zwei Fully Connected Layer. Eines um gemeinsame und eines um modalitätsspezifische Features zu berechnen. Die RGB-spezifischen Features s_{rgb} und die beiden gemeinsamen Features c_{rgb} und c_{depth} werden anschließend mit einem weiteren Fully Connected Layer verarbeitet um den Input für den RGB-Decoder zu bilden. Analog werden s_{depth}, c_{depth} und c_{rgb} verwendet, um mit einem weiteren Fully Connected Layer den Input für den Tiefen-Decoder zu bilden.

Dadurch, dass jede Modalität auch die gemeinsamen Features der anderen Modalität nutzt, können komplementäre Features ausgetauscht werden. Über eine zusätzliche Lossfunktion wird sichergestellt, dass sich die Verteilung der beiden gemeinsamen Features c_{rgb} und c_{depth} ähnelt und sich die Verteilung der modalitätsspezifischen Features s_{rgb} und s_{depth} möglichst unterscheidet.

Idempotentes Mapping. In [Xing et al., 2019b] wird eine weitere Möglichkeit für den Informationsaustausch zwischen zwei getrennten ResNet-basierten Netzwerken vorgestellt. Indem die Identity-Verbindungen der Residual-Blöcke[8] durch *Idempotente Mappings* ersetzt werden, können die beiden Netzwerke Features austauschen. In Abbildung A.18 werden die so modifizieren Residual-Blöcke dargestellt. Ein idempotentes Mapping $\underline{\mathbf{T}}$ erfüllt $\underline{\mathbf{T}}^2 = \underline{\mathbf{T}}$. Ein Residual-Block kann auch geschrieben werden als:

$$\begin{bmatrix} \underline{\mathbf{x}}_{t+1}^0 \\ \underline{\mathbf{x}}_{t+1}^1 \end{bmatrix} = \begin{bmatrix} H^0(\underline{\mathbf{x}}_t^0) \\ H^1(\underline{\mathbf{x}}_t^1) \end{bmatrix} + \begin{bmatrix} \underline{\mathbf{x}}_t^0 \\ \underline{\mathbf{x}}_t^1 \end{bmatrix} \tag{A.1}$$

Dabei sind $\underline{\mathbf{x}}_t^0$ und $\underline{\mathbf{x}}_t^1$ die Eingaben des t-ten Residual-Blocks und H^0 und H^1 die Transitionsfunktionen des Residual-Zweigs. $\underline{\mathbf{x}}^0$ steht hierbei für Features des RGB-Netzwerks und $\underline{\mathbf{x}}^1$ für Features des Tiefennetzwerks. Zwischen $\underline{\mathbf{x}}^0$ und $\underline{\mathbf{x}}^1$ findet kein Informationsaustausch statt. Indem ein lineares Mapping $\underline{\mathbf{T}}$ in den Shortcut des Residual-Blocks eingefügt wird, können die beiden Netzwerke miteinander kommunizieren:

$$\begin{bmatrix} \underline{\mathbf{x}}_{t+1}^0 \\ \underline{\mathbf{x}}_{t+1}^1 \end{bmatrix} = \begin{bmatrix} H^0(\underline{\mathbf{x}}_t^0) \\ H^1(\underline{\mathbf{x}}_t^1) \end{bmatrix} + \underline{\mathbf{T}} \begin{bmatrix} \underline{\mathbf{x}}_t^0 \\ \underline{\mathbf{x}}_t^1 \end{bmatrix} \tag{A.2}$$

Wenn $\underline{\mathbf{x}}_t^0$ und $\underline{\mathbf{x}}_t^1$ jeweils c Kanäle haben, dann ist $\underline{\mathbf{T}}$ eine $2c \times 2c$ Matrix. Im Standard-Residual-Block ist $\underline{\mathbf{T}} = \underline{\mathbf{I}}_{2c \times 2c}$ eine Einheitsmatrix. $\underline{\mathbf{T}}$ kann auch im Blockformat geschrieben werden

$$\underline{\mathbf{T}} = \begin{bmatrix} \underline{\mathbf{T}}_{00} & \underline{\mathbf{T}}_{01} \\ \underline{\mathbf{T}}_{10} & \underline{\mathbf{T}}_{11} \end{bmatrix}, \tag{A.3}$$

wobei $\underline{\mathbf{T}}_{ij}$, $\forall i, j \in \{0, 1\}$ $c \times c$ Matrizen sind. Sobald nun $\underline{\mathbf{T}}_{01}$ und $\underline{\mathbf{T}}_{10}$ nicht $\underline{\mathbf{0}}$ sind, führt das dazu, dass die Outputs $\underline{\mathbf{x}}_{t+1}^0$ und $\underline{\mathbf{x}}_{t+1}^1$ von beiden Inputs $\underline{\mathbf{x}}_t^0$ und $\underline{\mathbf{x}}_t^1$ abhängen. Somit ist ein Informationsaustausch zwischen den beiden Netzwerken hergestellt. Als idempotentes Mapping $\underline{\mathbf{T}}$ schlagen die Autoren unter anderem das Merge-and-Run-Mapping

$$\underline{\mathbf{M}} = \frac{1}{2} \begin{bmatrix} \underline{\mathbf{I}} & \underline{\mathbf{I}} \\ \underline{\mathbf{I}} & \underline{\mathbf{I}} \end{bmatrix} \tag{A.4}$$

[8] Residual-Blöcke werden in Abschnitt 2.1 vorgestellt.

vor. Bei der Wahl des Mappings ist darauf zu achten, dass die Konvergenz Netzwerks garantiert wird, da alle Residual-Blöcke des Netzwerks modifiziert werden und es somit leicht zu explodierenden oder verschwindenden Gradienten kommen kann. Außerdem muss bei Verwendung von ReLU als Ausgabefunktion auch die Funktion $F(\underline{\mathbf{x}}) = \underline{\mathbf{T}}\ ReLU(\underline{\mathbf{x}})$ idempotent sein. Das führt dazu, dass $\underline{\mathbf{T}}$ keine negativen Zahlen enthalten darf.

Fusion der Tiefen-Features in den RGB-Encoder

Dieser Abschnitt enthält eine Ergänzung zum Hauptabschnitt 4.1.4 (Seite 49).

Global Convolutional Network. In [Gao et al., 2019] wird das FuseNet[9] um ein sogenanntes *Global Convolutional Network* als Skip Connections zwischen Encoder und Decoder erweitert, welches in Abbildung A.19 dargestellt ist.

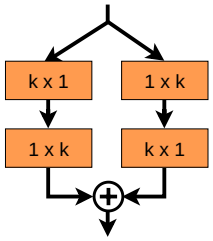


Abbildung A.19: Global Convolutional Network
Mithilfe von großen Kernels k können mehr Kontextinformationen extrahiert werden. Für einen geringeren Rechenaufwand werden Factorized Convolutions eingesetzt. Bereits in [Peng et al., 2017] konnte die Segmentierungsleistung mithilfe der Global Convolutional Networks als Skip Connections zwischen Encoder und Decoder verbessert werden.

Im Global Convolutional Network werden große Kernels verwendet, um mehr Kontextinformationen zu extrahieren. Die Verwendung von großen Kernels erhöht allerdings den Berechnungsaufwand deutlich. Daher werden hierfür Factorized Convolutions[10] eingesetzt. Eine $k \times k$ Convolution wird mittels zweier paralleler Factorized Convolutions approximiert. Im ersten Zweig wird zuerst ein horizontaler und anschließend

[9] Das FuseNet wird im Abschnitt 4.1.4, Seite 49 vorgestellt.
[10] Factorized Convolutions werden in Abschnitt 2.2.2 erklärt.

ein vertikaler Filter verwendet. Im zweiten Zweig ist die Reihenfolge umgekehrt. Die resultierenden Feature Maps beider Zweige werden daraufhin elementweise addiert. Während [Peng et al., 2017] Kernelgrößen von $k = 15$ einsetzen, verwenden [Gao et al., 2019] nur Kernelgrößen von $k = 5$.
Auf die summierten RGB- und Tiefen-Feature-Maps der letzten vier Auflösungsstufen wird jeweils ein Global Convolutional Network aufgesetzt. Die resultierenden Feature Maps werden mit den Feature Maps des Decoders derselben Auflösungsstufe elementweise addiert.

Fusion über einen dritten Encoder

Dieser Abschnitt enthält zusätzliche Fusionierungen über einen dritten Encoder zu der in Abschnitt 4.1.5 (Seite 51) vorgestellten Fusionierung.

V-FuseNet. Das V-FuseNet aus [Audebert et al., 2018] ergänzt das FuseNet um einen dritten Encoder. Bei der ersten Fusionierung bildet die Konkatenation der RGB- und Tiefen-Features den Input für den dritten Encoder.[11] Die konkatenierten Features werden daraufhin mit einer Convolution verarbeitet, wie in Abbildung A.20 zu erkennen ist.

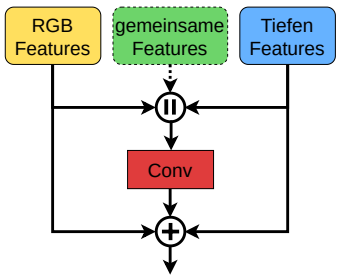


Abbildung A.20:
Fusion im V-FuseNet
Abbildung nach [Audebert et al., 2018]

In dieser Convolution wird ein Residual gelernt, welches anschließend mit den RGB- und Tiefen-Feature-Maps summiert wird. In jeder nachfolgenden Fusionierung werden

[11] Das V-FuseNet fusioniert eigentlich keine RGBD-Daten. Stattdessen werden RGB, Lidar und andere multispektrale Daten fusioniert. Der Ansatz ist jedoch auf eine RGBD-Fusion übertragbar. Für eine bessere Konsistenz innerhalb dieser Masterarbeit wird dennoch von RGB und Tiefe gesprochen.

die RGB- und Tiefen-Features mit den gemeinsamen Features konkateniert.

Composite Fusion. In [Piramanayagam et al., 2018] wird eine *Composite Fusion* vorgestellt: Die Features beider – bzw. ab der zweiten Fusion aller drei – Encoder werden hier einfach konkateniert. Auf die konkatenierten Features wird im dritten Encoder eine Convolution und ein Pooling angewandt, bevor die Features der nächstkleineren Auflösungsstufe mit in den dritten Encoder konkateniert werden. Die beiden modalitätsspezifischen Encoder verarbeiten ihre Features lediglich bis zu einem Downsamplingfaktor von 4. Die weitere Feature-Extraktion bis zu einem Downsamplingfaktor von 32 erfolgt ausschließlich im dritten Encoder.

A.3.2 Verwendung des Tiefenbilds nur zum Training

Dieser Abschnitt liefert ergänzende Informationen zum Abschnitt 4.2 (Seite 55).

Zusätzliche Lossfunktion

Im Depth-Assisted RefineNet von [Chang et al., 2018b] wird eine zusätzliche Lossfunktion zum Trainieren eines Segmentierungsnetzwerks vorgestellt. Die prinzipielle Idee ist, dass benachbarte Pixel desselben Objekts ähnliche Tiefenwerte aufweisen. Tritt zwischen zwei benachbarten Pixeln ein großer Tiefensprung auf, so gehören die beiden Pixel wahrscheinlich zu unterschiedlichen Objekten. Die zusätzliche Lossfunktion L_{depth} überprüft zuerst, ob die Prädiktion zwei benachbarte Pixel zur selben semantischen Klasse (Label) zuordnet. Ist dies der Fall und der Tiefensprung zwischen den beiden Pixeln überschreitet einen festgelegten Schwellwert, dann bestraft die Lossfunktion. Ansonsten passiert nichts und die Lossfunktion hat den Wert 0. Die Labeldifferenz[12] Δl und die Tiefendifferenz Δd werden jeweils für den rechten und den darunterliegenden Nachbarpixel bestimmt. Nur, wenn die $\Delta l = 0$ ist – also die Prädiktion beiden Pixeln dasselbe Label zuordnet – kann $L_{depth} > 0$ werden. Zusätzlich muss die quadrierte Tiefendifferenz größer als ein Schwellwert τ werden.
Die gesamte Lossfunktion zum Trainieren des Netzwerks ergibt sich zu:

$$L = L_{crossentropy} + \omega\, L_{depth} \tag{A.5}$$

[12] Die Labels werden typischerweise nach der Klassenanzahl durchnummeriert. Somit lässt sich auch eine Labeldifferenz berechnen.

Dabei ist $L_{crossentropy}$ der normale Kreuzentropie-Loss. Über ω kann die Gewichtung der zusätzlichen Lossfunktion variiert werden. Die Autoren trainierten ihr Netzwerk zuerst ausschließlich mit dem Kreuzentropie-Loss und nahmen in späteren Epochen die zweite Lossfunktion hinzu. Dabei setzen sie $\omega = 0.1$ und $\tau = 0.09$. Das RefineNet-152 wurde einmal mit und einmal ohne zusätzliche Lossfunktion auf dem SUN RGB-D Datensatz trainiert. Dabei wurde allerdings nur eine Verbesserung der mean Intersection over Union (mIoU)[13] von 45.9 auf 46.3 auf dem Testdatensatz erreicht. Die Tiefenbilder wurden zur Inferenz nicht verwendet.

Aufgrund der nur minimalen Verbesserung ist die alleinige Verwendung der zusätzlichen Lossfunktion eher weniger empfehlenswert. Sie kann jedoch mit anderen Verfahren zur RGBD-Segmentierung kombiniert werden. Für die eigene Anwendung ist darauf zu achten, dass τ nicht zu gering gewählt wird. Bei manchen Objekten kann durchaus ein Tiefensprung auftreten. Für geringe τ würde das Netzwerk trotz richtiger Prädiktion bestraft werden. Andererseits sollte τ auch nicht zu hoch gewählt werden. Je höher τ, desto seltener hat die zusätzliche Lossfunktion einen Einfluss.

Multi Task Learning

Sowohl [Vu et al., 2019] als auch [Jiao et al., 2019] verwenden in ihrem Netzwerk einen zusätzlichen zweiten Decoder für die Prädiktion des Tiefenbilds. Das Netzwerk besteht somit aus einem Encoder und zwei task-spezifischen Decodern. Während des Trainings gibt das Netzwerk sowohl die Segmentierung als auch das Tiefenbild aus. Für beide existieren Ground-Truth-Bilder. Das Netzwerk wird mittels jeweils einer Lossfunktion für die Segmentierung und für das Tiefenbild trainiert. Das Training mit mehr als einer Ausgabe wird als *Multi Task Learning* bezeichnet. Multi Task Learning ermöglicht höhere Generalisierungsfähigkeiten und bessere Performance auf jedem einzelnen Task, wie bereits in [Caruana et al., 1997] gezeigt wurde. Das heißt, die zusätzliche Prädiktion des Tiefenbilds kann dabei helfen die Segmentierung zu verbessern. In [Vu et al., 2019] wird zur Inferenz nur noch der Segmentierungs-Decoder benötigt. In [Jiao et al., 2019] werden dahingegen vom Tiefen-Decoder Feature Maps in den Segmentierungs-Decoder fusioniert. Die berechneten Tiefen-Features sollen dadurch dabei helfen die Segmentierung zu verbessern.

[13] Die mIoU wird in Abschnitt 5.7 definiert.

A.3.3 Verarbeitung von 3D-Daten

Das hier vorgestellte Verfahren erweitert den Abschnitt 4.3 (Seite 56).

3D-Convolution. In [Zhong et al., 2018] wird das 2D-RGB-Bild mithilfe des Tiefenbilds in den 3D-Raum projiziert. Anstelle eines XYZ-Raums verwenden die Autoren eine UVD-Voxel-Repräsentation. Dabei sind (U, V) die Koordinaten des 2D-Bilds. D gibt die Disparität aus dem Tiefenbild an. Obwohl die UVD-Voxel-Repräsentation keine tatsächlichen 3D-Darstellung repräsentiert, enthält sie dennoch genügend geometrische Informationen. Die Disparität D wird auf weniger als 200 Disparitätsstufen skaliert. Der in den UVD-Raum projizierte Input enthält für einen Disparitätswert die RGB-Werte des RGB-Bilds. Alle anderen Werte entlang der Disparitätsachse sind 0. Die Ground-Truth-Segmentierung wird ähnlich in den 3D-Raum projiziert: Bis auf eine Position sind alle Labels entlang der Disparitätsachse 0, also der Void-Klasse zuzuordnen. Der Input wird im Encoder mit vier 3D-Convolutions der Größe 3×3×3 und Stride 2 verarbeitet. Die ursprüngliche Auflösung wird symmetrisch im Decoder mit 3D-Transposed-Convolutions wiederhergestellt. Da der Input zu einem großen Teil aus Nullen besteht, wird jede Auflösungsstufe des Decoders mit Feature Maps aus dem Encoder über Skip Connections verbunden. Die Feature Maps des Encoders werden hierfür zuerst mit einem 3D-Residual-Block verarbeitet und anschließend mit den Decoder Feature Maps addiert. Ein weiterer 3D-Residual-Block wird direkt zwischen Input und Output platziert. Durch den Identity Shorcut im 3D-Residual-Block muss im gesamten Netzwerk lediglich die Differenz zu einem Output bestehend aus ausschließlich Nullen berechnet werden. Die 2D-Segmentierung wird über ein Max-Pooling entlang der Disparitätsachse bestimmt.

A.3.4 Depth-Aware Convolution

In [Wang und Neumann, 2018] wird die Convolution um einen Tiefenähnlichkeits-Term erweitert. Damit soll die 2D-Convolution in der Lage sein geometrische Informationen auszunutzen. Anstelle aufwendiger 3D-Convolutions oder zwei parallelen Netzwerkzweigen für Tiefe und RGB kann jede beliebige Segmentierungsarchitektur für 2D-RGB-Bilder verwendet werden. Die Standard-Convolution kann dabei einfach durch die sogenannte *Depth-Aware Convolution* ersetzt werden. Mithilfe des Tiefenähnlichkeits-

Terms wird der Einfluss einzelner Pixel in einer Convolution gesteuert. Pixel mit einem ähnlichen Tiefenwert wie der Aufpunkt-Pixel erhalten mehr Einfluss als Pixel mit einem anderen Tiefenwert. Dieser Ansatz basiert auf der Intuition, dass Pixel desselben Objekts ähnliche Tiefenwerte haben und damit einen größeren Einfluss haben sollten. In Abbildung A.21 gehören Pixel A und C zum selben Objekt und sollten daher stärker korreliert sein als Pixel A und B. Dieser Korrelationsunterschied ist im Tiefenbild sichtbar, aber im RGB-Bild nicht zu erkennen. Mithilfe des Tiefenähnlichkeits-Terms werden nun die Tiefendifferenzen zum Aufpunktpixel A bestimmt. Die resultierenden Ähnlichkeiten sind in der Abbildung oben zu sehen. Je ähnlicher die Tiefenwerte, desto mehr Einfluss erhält ein Pixel.

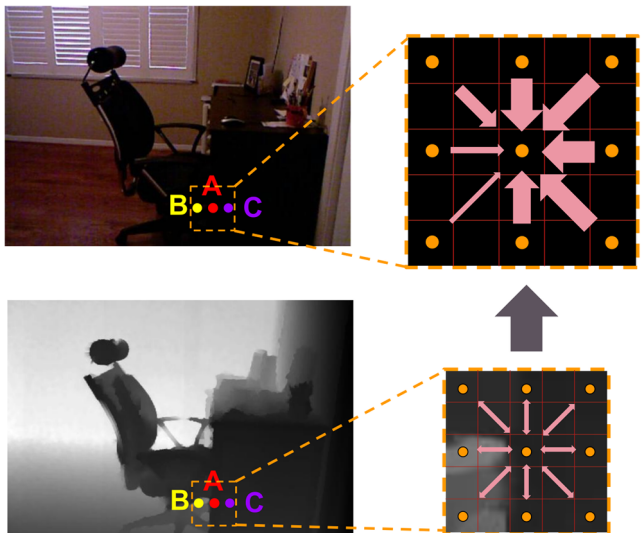


Abbildung A.21: Depth-Aware Convolution
*Pixel mit einem ähnlichen Tiefenwert wie der Aufpunktpixel erhalten einen größeren Einfluss bei der Convolution. Abbildung aus [*Wang *und* Neumann*, 2018]*

Der Tiefenähnlichkeits-Term F_D zwischen zwei Pixeln p_i und p_j ist folgendermaßen definiert:

$$F_D\left(p_i, p_j\right) = exp\left(-\alpha|D\left(p_i\right) - D\left(p_j\right)|\right) \tag{A.6}$$

Dabei gibt $D\left(p_i\right)$ den Tiefenwert des Pixels p_i an. α is eine Konstante, die von den Autoren auf $\alpha = 8.3$ gesetzt wurde. Zusätzlich zum Kernelgewicht wird bei der Depth-Aware Convolution jeder Pixel mit dem berechneten Tiefenähnlichkeits-Term F_D

gewichtet.
Der Tiefenähnlichkeits-Term kann auch für das Average Pooling angewandt werden. Beim *Depth-Aware Average Poooling* wird somit ein gewichteter Mittelwert berechnet.

Für die Umsetzung der Depth Aware Convolution muss die Convolution-Operation neu implementiert werden. Zudem wird aus dem Paper nicht ersichtlich wie die Tiefenwerte in späteren Netzwerkschichten mit geringerer räumlicher Auflösung einbezogen werden.

A.4 Experimente

Für die beiden Trainings #1 und #2 sind in den nachfolgenden Tabellen jeweils die maximal erzielte mIoU und die zugehörige Epoche angegeben. Daneben steht der Mittelwert und die Standardabweichung aus beiden Werten. Die angegebenen Epochen sind 1-indiziert. Für die zugehörige Datei mit den Gewichten des Netzwerks muss 1 subtrahiert werden.

A.4.1 Netzwerktiefe vs. Modalität

Mod	Encoder	Lernraten											
		0.00125			0.0025			0.005			0.01		
		#1	#2	∅	#1	#2	∅	#1	#2	∅	#1	#2	∅
Tiefe	ResNet18	35.12	35.11	35.12	35.57	35.73	35.65	35.97	35.81	35.89	36.05	35.82	35.94
		(257)	(487)	±0.01	(322)	(202)	±0.08	(382)	(427)	±0.08	(471)	(366)	±0.12
	ResNet34	37.24	36.54	36.89	37.33	37.57	37.45	37.50	37.98	37.74	38.17	37.70	37.94
		(249)	(178)	±0.35	(344)	(478)	±0.12	(276)	(480)	±0.24	(401)	(470)	±0.24
	ResNet50	37.66	37.69	37.68	37.98	37.90	37.94	38.36	38.20	38.28	38.39	38.38	38.39
		(392)	(485)	±0.01	(366)	(215)	±0.04	(385)	(354)	±0.08	(340)	(457)	±0.01
RGB	ResNet18	39.25	39.75	39.50	39.15	39.45	39.30	39.45	39.59	39.52	39.27	39.12	39.19
		(229)	(309)	±0.25	(273)	(454)	±0.15	(488)	(359)	±0.07	(467)	(399)	±0.07
	ResNet34	42.16	42.44	42.30	42.00	41.81	41.91	41.78	41.82	41.80	41.38	40.97	41.18
		(451)	(428)	±0.14	(373)	(210)	±0.09	(336)	(375)	±0.02	(459)	(500)	±0.20
	ResNet50	42.84	42.66	42.75	42.87	42.23	42.55	42.40	42.44	42.42	41.59	42.05	41.82
		(410)	(322)	±0.09	(179)	(133)	±0.32	(299)	(280)	±0.02	(436)	(460)	±0.23
RGBD	ResNet18	43.46	43.87	43.66	44.06	44.08	44.07	44.11	44.09	44.10	43.96	43.34	43.65
		(430)	(407)	±0.20	(217)	(253)	±0.01	(246)	(492)	±0.01	(181)	(259)	±0.31
	ResNet34	45.29	45.57	45.43	45.23	45.76	45.49	45.30	45.41	45.35	45.03	44.91	44.97
		(205)	(265)	±0.14	(266)	(283)	±0.26	(151)	(323)	±0.05	(189)	(317)	±0.06
	ResNet50	46.29	46.30	46.30	46.03	46.21	46.12	46.54	46.73	46.64	46.42	46.97	46.70
		(308)	(178)	±0.01	(184)	(204)	±0.09	(283)	(311)	±0.09	(279)	(444)	±0.28

Tabelle A.1: Netzwerktiefe vs. Modalität – Baseline-Experimente – mIoU
In Abbildung 6.1 (Seite 81) sind die erreichten Ergebnisse visuell dargestellt.

Mod	Encoder	FPS		
		PyTorch	TensorRT Float32	TensorRT Float16
Tiefe	ResNet18	28.3	49.1	91.1
	ResNet34	20.2	31.8	68.8
	ResNet50	13.7	22.0	56.9
RGB	ResNet18	27.0	46.9	88.4
	ResNet34	19.8	30.6	67.1
	ResNet50	13.5	21.5	55.9
RGBD	ResNet18	17.2	26.6	57.7
	ResNet34	11.5	16.7	40.5
	ResNet50	5.0	8.3	25.8

Tabelle A.2: Netzwerktiefe vs. Modalität – FPS
Verarbeitungsfrequenzen auf einem Jetson AGX Xavier. Die Verarbeitungsfrequenzen mit TensorRT Float32 sind in Abbildung 6.3 (Seite 82) visuell dargestellt.

	ResNet18			ResNet34		
	#1	#2	∅	#1	#2	∅
feste Lernrate	44.11	44.09	44.10	45.23	45.76	45.49
	(246)	(492)	±0.01	(266)	(283)	±0.26
RGB-Encoder und Decoder: lr=0.00125	44.25	44.46	44.36	45.22	45.21	45.22
Tiefen-Encoder: lr=0.01	(289)	(317)	±0.10	(384)	(129)	±0.01
RGB-Encoder: lr=0.00125	44.18	43.99	44.09	45.55	45.50	45.53
Tiefen-Encoder und Decoder: lr=0.01	(240)	(487)	±0.10	(196)	(497)	±0.03

Tabelle A.3: Verschiedene Lernraten für verschiedene Netzwerkteile – mIoU
Zugehörige Tabelle im Hauptteil der Masterarbeit: Tabelle 6.1 (Seite 83)

Encoder	Lernraten														
	0.00125			0.0025			0.005			0.01			0.02		
	#1	#2	∅	#1	#2	∅	#1	#2	∅	#1	#2	∅	#1	#2	∅
nur ImageNet-Vortraining	43.46	43.87	43.66	44.06	44.08	44.07	44.11	44.09	44.10	43.96	43.34	43.65	-	-	-
	(430)	(407)	±0.20	(217)	(253)	±0.01	(246)	(492)	±0.01	(181)	(259)	±0.31			
Encoder vortrainiert	42.30	42.22	42.26	42.43	42.57	42.50	43.50	43.35	43.42	43.70	43.61	43.65	43.81	43.44	43.63
	(110)	(40)	±0.04	(74)	(301)	±0.07	(266)	(242)	±0.08	(436)	(294)	±0.04	(489)	(404)	±0.19
Encoder und Decoder vortrainiert	42.43	42.19	42.31	42.37	42.51	42.44	43.38	43.17	43.28	-	-	-	-	-	-
	(139)	(192)	±0.12	(217)	(244)	±0.07	(352)	(211)	±0.10						

Tabelle A.4: Vortrainieren auf unimodalen Segmentierungsnetzwerken – mIoU
RGBD-ResNet18-Netzwerk unterschiedlich vortrainiert. In Abbildung 6.4 (Seite 85) sind die Ergebnisse visuell dargestellt.

Encoder	Lernraten											
	0.00125			0.0025			0.005			0.01		
	#1	#2	∅	#1	#2	∅	#1	#2	∅	#1	#2	∅
ResNet18	43.46	43.87	43.66	44.06	44.08	44.07	44.11	44.09	44.10	43.96	43.34	43.65
	(430)	(407)	±0.20	(217)	(253)	±0.01	(246)	(492)	±0.01	(181)	(259)	±0.31
RGB: ResNet34 Tiefe: ResNet18	44.57	44.53	44.55	44.57	45.19	44.88	45.09	45.23	45.16	44.21	44.80	44.51
	(164)	(235)	±0.02	(138)	(263)	±0.31	(254)	(360)	±0.07	(218)	(358)	±0.30
ResNet34	45.29	45.57	45.43	45.23	45.76	45.49	45.30	45.41	45.35	45.03	44.91	44.97
	(205)	(265)	±0.14	(266)	(283)	±0.26	(151)	(323)	±0.05	(189)	(317)	±0.06
ResNet50	46.29	46.30	46.30	46.03	46.21	46.12	46.54	46.73	46.64	46.42	46.97	46.70
	(308)	(178)	±0.01	(184)	(204)	±0.09	(283)	(311)	±0.09	(279)	(444)	±0.28

Tabelle A.5: Unterschiedlich tiefe Encoder – mIoU
In Abbildung 6.6 (Seite 87) sind die Ergebnisse visuell dargestellt.

A.4.2 Variation der Encoder-Blöcke

Encoder		FPS PyTorch	FPS TensorRT Float32	FPS TensorRT Float16
ResNet34	Non-Bottleneck-Block (Baseline)	11.5	16.7	40.7
	Non-Bottleneck-1D-Block	9.0	14.9	43.3
	SS-NBt-Block	10.2	18.6	37.5
	PFCU	5.3	8.3	25.0
	DAB-Block	12.3	16.6	22.7
	eigener Block	10.6	16.2	36.4
EfficientNet-B0		13.3	19.4	26.1

Tabelle A.6: Unterschiedliche Encoder – FPS

Zugehörige Tabelle im Hautteil der Masterarbeit: Tabelle 6.3 (Seite 91)

Encoder		Lernraten 0.00125 #1	0.00125 #2	0.00125 ∅	0.0025 #1	0.0025 #2	0.0025 ∅	0.005 #1	0.005 #2	0.005 ∅	0.01 #1	0.01 #2	0.01 ∅
ResNet34	**Non-Bottleneck-Block (Baseline)**	45.29 (205)	45.57 (265)	45.43 ±0.14	45.23 (266)	45.76 (283)	45.49 ±0.26	45.30 (151)	45.41 (323)	45.35 ±0.05	45.03 (189)	44.91 (317)	44.97 ±0.06
	Non-Bottleneck-1D-Block	45.79 (294)	45.59 (239)	45.69 ±0.10	45.42 (360)	45.90 (277)	45.66 ±0.24	45.28 (313)	45.68 (311)	45.48 ±0.20	45.35 (325)	45.34 (370)	45.34 ±0.01
	eigener Block	45.28 (268)	45.09 (288)	45.19 ±0.09	45.33 (289)	45.31 (235)	45.32 ±0.01	45.58 (138)	45.65 (240)	45.61 ±0.03	45.46 (188)	45.43 (373)	45.45 ±0.01
	DAB-Block	42.76 (309)	42.56 (298)	42.66 ±0.10	43.28 (410)	43.27 (459)	43.27 ±0.00	43.04 (136)	43.11 (497)	43.08 ±0.04	42.85 (233)	43.25 (384)	43.05 ±0.20
	DAB-Block Dilation 2	42.76 (375)	42.96 (256)	42.86 ±0.10	43.37 (252)	42.80 (397)	43.09 ±0.28	43.28 (462)	43.57 (438)	43.42 ±0.14	42.91 (398)	43.19 (495)	43.05 ±0.14
	EfficientNet-B0	44.32 (298)	-	44.32 ±0.00	44.76 (270)	-	44.76 ±0.00	44.73 (238)	-	44.73 ±0.00	-	-	-
	EfficientNet-B0 mit letzter 1×1 Conv	43.75 (236)	-	43.75 ±0.00	43.80 (119)	-	43.80 ±0.00	44.52 (166)	-	44.52 ±0.00	-	-	-

Tabelle A.7: Variation der Encoder-Blöcke – mIoU

In Abbildung 6.8 (Seite 96) sind die Ergebnisse visuell dargestellt.

A.4.3 Variation im Decoder

	Lernraten								
	0.00125			0.0025			0.005		
	#1	#2	∅	#1	#2	∅	#1	#2	∅
● **gewichtet**	45.79	45.59	45.69	45.42	45.90	45.66	45.28	45.68	45.58
	(294)	(239)	±0.10	(360)	(277)	±0.24	(313)	(311)	±0.20
● **Konkatenation**	45.56	45.51	45.53	45.49	45.66	45.57	45.55	43.36	45.45
	(235)	(297)	±0.03	(271)	(399)	±0.09	(269)	(380)	±0.09
● **keine**	44.96	44.92	44.94	45.03	44.78	44.91	45.00	44.90	44.95
	(205)	(406)	±0.02	(358)	(220)	±0.12	(296)	(357)	±0.05

Tabelle A.8: Verschiedene Encoder-Decoder-Fusionierungen – mIoU
In Abbildung 6.9 (Seite 99) sind die Ergebnisse visuell dargestellt.

	FPS		
	PyTorch	TensorRT Float32	TensorRT Float16
● **gewichtet**	9.1	14.9	46.3
● **Konkatenation**	9.4	14.9	43.6
● **keine**	9.6	15.5	46.2

Tabelle A.9: Verschiedene Encoder-Decoder-Fusionierungen – FPS
Zugehörige Segmentierungsleistungen sind in Abbildung 6.9 (Seite 99) dargestellt bzw. exakt in Tabelle A.8 enthalten.

	C_{dec}	C_{enc}	Lernraten								
			0.00125			0.0025			0.005		
			#1	#2	∅	#1	#2	∅	#1	#2	∅
●	**128**	**24**	45.79	45.59	45.69	45.42	45.90	45.66	45.28	45.68	45.58
			(294)	(239)	±0.10	(360)	(277)	±0.24	(313)	(311)	±0.20
●	**128**	**48**	45.61	45.11	45.36	45.98	45.91	45.95	45.90	45.99	45.94
			(317)	(108)	±0.25	(173)	(237)	±0.04	(284)	(262)	±0.05
●	**256**	**24**	45.97	46.04	46.01	46.12	46.09	46.11	46.03	46.12	46.08
			(179)	(204)	±0.03	(261)	(289)	±0.01	(159)	(306)	±0.05
●	**256**	**48**	45.87	45.81	45.84	46.35	45.83	46.09	46.12	46.42	46.27
			(308)	(248)	±0.03	(208)	(161)	±0.26	(335)	(307)	±0.15
●	**[512, 256, 128]**	**[12, 24, 48]**	45.71	45.99	45.85	46.38	46.39	46.38	46.13	46.02	46.07
			(236)	(230)	±0.14	(276)	(305)	±0.00	(388)	(350)	±0.05

Tabelle A.10: Breiterer Decoder – mIoU
In Abbildung 6.11 (Seite 102) sind die Ergebnisse visuell dargestellt.

	C_{dec}	C_{enc}	FPS PyTorch	FPS TensorRT Float32	FPS TensorRT Float16
●	**128**	**24**	9.1	14.9	43.3
●	**128**	**48**	8.9	14.4	41.5
●	**256**	**24**	8.6	13.9	38.3
●	**256**	**48**	8.4	13.4	38.1
●	[**512**, **256**, **128**]	[**12**, **24**, **48**]	8.6	13.9	40.0

Tabelle A.11: Breiterer Decoder – FPS

Erzielte Segmentierungsleistungen sind in Abbildung 6.11 (Seite 102) dargestellt bzw. exakt in Tabelle A.10 enthalten.

	Lernraten											
	0.00125			**0.0025**			**0.005**			**0.01**		
	#1	**#2**	∅	**#1**	**#2**	∅	**#1**	**#2**	∅	**#1**	**#2**	∅
● **Baseline**	45.71 (236)	45.99 (230)	45.85 ±0.14	46.38 (276)	46.39 (305)	46.38 ±0.00	46.13 (388)	46.02 (350)	46.07 ±0.05	-	-	-
▪▪ **Non-Bottleneck-1D-Block**	46.55 (426)	46.55 (253)	46.55 ±0.00	46.53 (341)	45.32 (327)	46.43 ±0.10	46.60 (319)	46.44 (293)	46.52 ±0.08	-	-	-
▪▪ **eigener Block**	45.95 (231)	46.11 (213)	46.03 ±0.08	46.07 (239)	46.17 (124)	46.12 ±0.05	46.18 (240)	46.38 (195)	46.28 ±0.10	46.09 (324)	-	46.09 ±0.00

Tabelle A.12: Tieferer Decoder – mIoU

In Abbildung 6.13 (Seite 104) sind die Ergebnisse visuell dargestellt.

	FPS PyTorch	FPS TensorRT Float32	FPS TensorRT Float16
● **Baseline**	8.6	13.9	40.0
▪▪ **Non-Bottleneck-1D-Block**	8.1	13.0	37.7
▪▪ **eigener Block**	9.3	14.2	32.9

Tabelle A.13: Tieferer Decoder – FPS

Zugehörige Segmentierungsleistungen sind in Abbildung 6.13 (Seite 104) dargestellt bzw. exakt in Tabelle A.12 enthalten.

A.4.4 Variation der RGBD-Fusion

	Gewichtung bzw Fusionsart	Lernraten											
		0.00125			0.0025			0.005			0.01		
		#1	#2	∅	#1	#2	∅	#1	#2	∅	#1	#2	∅
Baseline	pixelweise nach SSMA (Baseline)	46.55 (426)	46.55 (253)	46.55 ±0.00	46.53 (341)	46.32 (327)	46.43 ±0.10	46.60 (319)	46.44 (293)	46.52 ±0.08	-	-	-
Baseline	SE mit 3×3 Conv	46.40 (194)	46.49 (404)	46.44 ±0.04	46.26 (243)	46.33 (340)	46.30 ±0.04	45.87 (235)	46.43 (224)	46.15 ±0.28	-	-	-
Baseline	SE mit 1×1 Conv	46.14 (419)	46.45 (261)	46.30 ±0.15	46.35 (157)	46.66 (230)	46.51 ±0.15	46.38 (439)	46.79 (295)	46.58 ±0.21	-	-	-
3. Encoder	pixelweise nach SSMA (Baseline)	46.32 (377)	46.26 (256)	46.29 ±0.03	46.63 (277)	46.41 (395)	46.52 ±0.11	46.03 (500)	46.45 (467)	46.24 ±0.21	-	-	-
3. Encoder	SE mit 3×3 Conv	46.41 (430)	46.49 (273)	46.45 ±0.04	46.69 (278)	46.75 (254)	46.72 ±0.03	46.66 (208)	46.36 (330)	46.51 ±0.16	-	-	-
3. Encoder	SE mit 1×1 Conv	46.25 (230)	46.50 (254)	46.37 ±0.13	46.79 (286)	46.33 (233)	46.56 ±0.23	46.32 (469)	46.53 (390)	46.42 ±0.11	-	-	-
Fusion in RGB	Encoder: SE Addition Decoder: Konkatenation	46.54 (363)	46.73 (313)	46.63 ±0.09	47.20 (227)	47.32 (404)	47.26 ±0.06	47.04 (409)	46.86 (397)	46.95 ±0.09	-	-	-
Fusion in RGB	Encoder: SE Addition Decoder: Addition	47.19 (417)	47.09 (387)	47.14 ±0.05	47.49 (232)	47.34 (341)	47.41 ±0.08	47.62 (303)	47.60 (339)	47.61 ±0.01	46.94 (477)	47.32 (487)	47.13 ±0.19
Fusion in RGB	Encoder: Addition Decoder: Addition	46.95 (231)	46.33 (170)	46.64 ±0.31	47.15 (336)	47.10 (325)	47.12 ±0.03	46.97 (244)	47.36 (284)	47.16 ±0.20	-	-	-

Tabelle A.14: Variation der RGBD-Fusion – mIoU
Die Ergebnisse sind visuell dargestellt in Abbildung 6.16 (Seite 108) und in Abbildung 6.18 (Seite 112). Zugehörige Verarbeitungsfrequenzen befinden sich in Tabelle A.15.

	Gewichtung bzw. Fusionsart	FPS		
		PyTorch	TensorRT Float32	TensorRT Float16
Baseline	pixelweise nach SSMA (Baseline)	8.1	13.0	37.7
Baseline	SE mit 3×3 Conv	8.2	13.2	38.9
Baseline	SE mit 1×1 Conv	8.4	13.6	40.2
3. Enc	pixelweise nach SSMA (Baseline)	8.0	12.8	37.9
3. Enc	SE mit 3×3 Conv	8.2	13.1	38.6
3. Enc	SE mit 1×1 Conv	8.4	13.5	39.8
Fusion in RGB	Encoder: SE Addition Decoder: gewichtete Konkatenation	8.3	13.1	35.1
Fusion in RGB	Encoder: SE Addition Decoder: Addition	8.6	13.2	35.5
Fusion in RGB	Encoder: Addition Decoder: Addition	8.7	13.8	42.2

Tabelle A.15: Variation der RGBD-Fusion – FPS
Erzielte Segmentierungsleistungen sind visuell dargestellt in Abbildung 6.16 (Seite 108) und in Abbildung 6.18 (Seite 112) bzw. exakt enthalten in Tabelle A.14.

A.4.5 Einfluss des Kontextmoduls

Kontextmodul	Lernraten								
	0.00125			0.0025			0.005		
	#1	#2	∅	#1	#2	∅	#1	#2	∅
PPM (Baseline)	47.19	47.09	47.14	47.49	47.34	47.41	47.62	47.60	47.61
	(417)	(387)	±0.05	(232)	(341)	±0.08	(303)	(339)	±0.01
eASPP	46.52	47.05	46.79	47.15	47.38	47.26	47.27	46.97	47.12
	(333)	(425)	±0.27	(380)	(229)	±0.12	(437)	(259)	±0.15
keins	46.87	46.47	46.67	47.01	46.94	46.97	46.50	47.32	46.91
	(212)	(235)	±0.20	(388)	(265)	±0.04	(399)	(303)	±0.41

Tabelle A.16: Einfluss des Kontextmoduls – mIoU
In Abbildung 6.20 (Seite 114) sind die Ergebnisse visuell dargestellt. Zugehörige Verarbeitungsfrequenzen sind in Tabelle A.17 enthalten.

verwendetes Kontextmodul	FPS		
	PyTorch	TensorRT Float32	TensorRT Float16
PPM (Baseline)	8.6	13.2	35.5
eASPP	8.4	12.8	34.8
keins	8.6	13.2	35.8

Tabelle A.17: Einfluss des Kontextmoduls – FPS
Zugehörige Segmentierungsergebnisse sind in Abbildung 6.20 (Seite 114) visuell dargestellt bzw. exakt in Tabelle A.16 enthalten.

A.4.6 Verbesserte vs. originale Tiefenbilder

	Lernraten								
	0.00125			0.0025			0.005		
	#1	#2	∅	#1	#2	∅	#1	#2	∅
verbesserte Tiefenbilder (Baseline)	47.19	47.09	47.14	47.49	47.34	47.41	47.62	47.60	47.61
	(417)	(387)	±0.05	(232)	(341)	±0.08	(303)	(339)	±0.01
originale Tiefenbilder	46.95	-	46.95	47.09	-	47.09	47.14	-	47.14
	(457)		±0.00	(195)		±0.00	(290)		±0.00

Tabelle A.18: Verbesserte vs. originale Tiefenbilder
In Abbildung 6.27 (Seite 127) sind die Ergebnisse visuell dargestellt.

A.4.7 Klassenverteilung pro Kamera

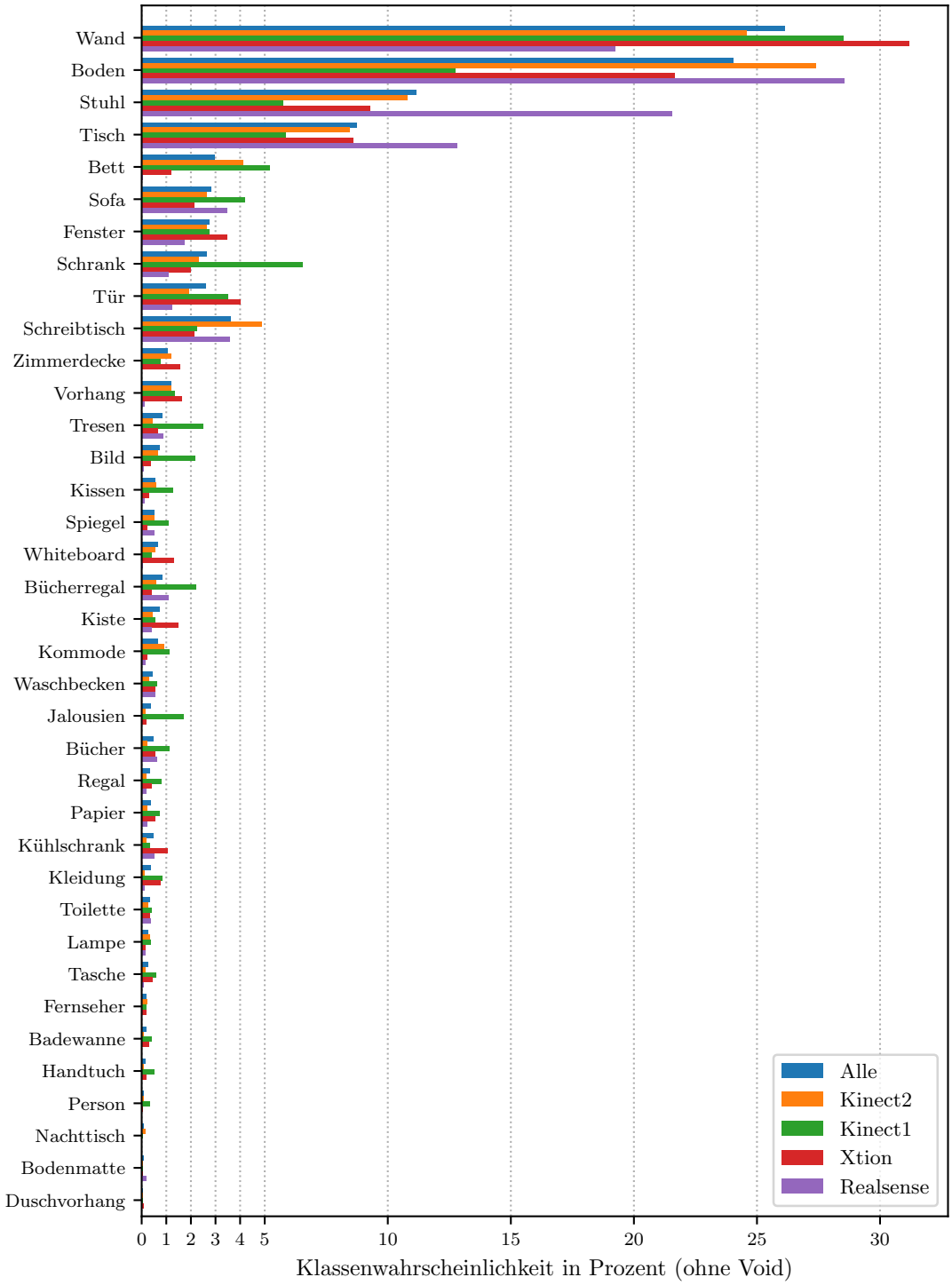


Abbildung A.22: Klassenverteilung pro Kamera im SUN RGB-D Testdatensatz
Die Verteilung wurde pro Kamera normiert. In Abbildung 5.4 ist die Klassenverteilung des Trainingsdatensatzes dargestellt. Diese Klassenverteilung ist hilfreich bei der Bewertung der IoUs pro Klasse nach Kamera in Abbbildung 6.22 (Seite 118).

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[Agoes et al., 2019] Agoes, Ali Suryaperdana, Z. Hu und N. Matsunaga (2019). *LICODS: a CNN Based, Lightweight RGB-D Semantic Segmentation for Outdoor Scenes*. International Journal of Innovative Computing, Information and Control (IJICIC), S. 1349–4198. *(Seite: 47, 50)*

[Armeni et al., 2017] Armeni, Iro, S. Sax, A. R. Zamir und S. Savarese (2017). *Joint 2D-3D-Semantic Data for Indoor Scene Understanding*. In: *arXiv preprint arXiv:1702.01105*. *(Seite: 65)*

[Audebert et al., 2018] Audebert, Nicolas, B. Le Saux und S. Lefèvre (2018). *Beyond RGB: Very high resolution urban remote sensing with multimodal deep networks*. Congress of the Interntional Society for Photogrammetry and Remote Sensing (ISPRS), S. 20–32. *(Seite: 47, 52, 151)*

[Badrinarayanan et al., 2017] Badrinarayanan, Vijay, A. Kendall, R. Cipolla und S. Member (2017). *SegNet : A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation*. IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI), S. 2481–2495. *(Seite: 49)*

[Bai, 2019] Bai, Kunlun (2019). *A Comprehensive Introduction to Different Types of Convolutions in Deep Learning*. `https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215`. Abgerufen am 04.11.2019. *(Seite: 10, 11, 13)*

[Cao et al., 2016] Cao, Yuanzhouhan, C. Shen, H. T. Shen und H. Tao Shen (2016). *Exploiting Depth from Single Monocular Images for Object Detection and Semantic Segmentation*. IEEE Transactions on Image Processing, S. 836–846. *(Seite: 47, 49)*

[Caruana et al., 1997] Caruana, Rich, L. Pratt und S. Thrun (1997). *Multitask Learning*. Machine Learning, 28(1):41–75. *(Seite: 153)*

[Chang et al., 2018a] Chang, Angel, A. Dai, T. Funkhouser, M. Halber, M. Niebner, M. Savva, S. Song, A. Zeng und Y. Zhang (2018a). *Matterport3D: Learning from RGB-D data in indoor environments*. International Conference on 3D Vision (3DV), S. 667–676. *(Seite: 65)*

[Chang et al., 2018b] Chang, Manyu, F. Guo und R. Ji (2018b). *Depth-assisted RefineNet for Indoor Semantic Segmentation*. In: *International Conference on Pattern Recognition (ICPR)*, S. 1845–1850. *(Seite: 55, 57, 152)*

[Chen et al., 2018a] Chen, Liang-Chieh, G. Papandreou, I. Kokkinos, K. Murphy und A. L. Yuille (2018a). *DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs.*. IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI), 40(4):834–848. *(Seite: 1, 38)*

[Chen et al., 2015] Chen, Liang-Chieh, G. Papandreou, K. Murphy und A. L. Yuille (2015). *Semantic Image Segmentation with Deep Convolutional Nets and Fully Connected CRFs*. In: *International Conference on Learning Representations (ICLR). (Seite: 1)*

[Chen et al., 2017] Chen, Liang-Chieh, G. Papandreou, F. Schroff und H. Adam (2017). *Rethinking Atrous Convolution for Semantic Image Segmentation*. arXiv preprint arXiv:1706.05587. *(Seite: 1, 138)*

[Chen et al., 2018b] Chen, Liang-Chieh, Y. Zhu, G. Papandreou, F. Schroff und H. Adam (2018b). *Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation*. In: *European Conference on Computer Vision (ECCV)*, S. 801–818. *(Seite: 1)*

[Cheng et al., 2017] Cheng, Yanhua, R. Cai, Z. Li, X. Zhao und K. Huang (2017). *Locality-sensitive deconvolution networks with gated fusion for RGB-D indoor semantic segmentation*. In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, S. 1475–1483. *(Seite: 47, 48)*

[Chollet, 2017] Chollet, François (2017). *Xception: Deep learning with depthwise separable convolutions.* In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, S. 1800–1807. *(Seite: 9)*

[Cordts et al., 2016] Cordts, Marius, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth und B. Schiele (2016). *The Cityscapes Dataset for Semantic Urban Scene Understanding.* IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), S. 3213–3223. *(Seite: 40)*

[Deng et al., 2019] Deng, Liuyuan, M. Yang, T. Li, Y. He und C. Wang (2019). *RFBNet: Deep Multimodal Networks with Residual Fusion Blocks for RGB-D Semantic Segmentation.* arXiv preprint arXiv:1907.00135. *(Seite: 47, 52, 55)*

[Dumoulin und Visin, 2016] Dumoulin, Vincent und F. Visin (2016). *A guide to convolution arithmetic for deep learning.* arXiv preprint arXiv:1603.07285. *(Seite: 16)*

[Eigen und Fergus, 2015] Eigen, David und R. Fergus (2015). *Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture.* International Conference on Computer Vision (ICCV), S. 2650–2658. *(Seite: 71)*

[Fischler und Bolles, 1981] Fischler, Martin a und R. C. Bolles (1981). *Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography.* Communications of the ACM, 24(6):381–395. *(Seite: 68)*

[Gao et al., 2019] Gao, Xiaoning, J. Yu und J. Li (2019). *RGBD Semantic Segmentation Based on Global Convolutional Network.* In: *International Conference on Informations in Control, Automation and Robotics (ICINCO)*, S. 192–197. *(Seite: 47, 49, 150, 151)*

[Giancola et al., 2018] Giancola, Silvio, M. Valenti und R. Sala (2018). *A survey on 3D cameras: Metrological comparison of time-of-flight, structured-light and active stereoscopy technologies.* Springer. *(Seite: 17)*

[Gross, 2019] Gross, H.-M. (2019). *Robotvision*. Vorlesungsskript Wintersemester 2018/19. *(Seite: 17, 66)*

[Gupta et al., 2013] Gupta, Saurabh, P. Arbelaez und J. Malik (2013). *Perceptual organization and recognition of indoor scenes from RGB-D images*. In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, S. 564–571. *(Seite: 4, 137)*

[Gupta et al., 2014] Gupta, Saurabh, R. Girshick, P. Arbeláez und J. Malik (2014). *Learning Rich Features from RGB-D Images for Object Detection and Segmentation*. In: *European Conference on Computer Vision (ECCV)*, S. 345—-360. Springer. *(Seite: 136, 137)*

[Ha et al., 2017] Ha, Qishen, K. Watanabe, T. Karasawa, Y. Ushiku und T. Harada (2017). *MFNet: Towards real-time semantic segmentation for autonomous vehicles with multi-spectral scenes*. In: *International Conference on Intelligent Robots and Systems (IROS)*, S. 5108–5115. *(Seite: 47, 52)*

[Hazirbas et al., 2016] Hazirbas, Caner, L. Ma, C. Domokos und D. Cremers (2016). *FuseNet: Incorporating Depth into Semantic Segmentation via Fusion-based CNN Architecture*. In: *Asian Conference on Computer Vision (ACCV)*, S. 213–228. *(Seite: 45, 47, 49, 57, 76)*

[He et al., 2015] He, Kaiming, X. Zhang, S. Ren und J. Sun (2015). *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*. In: *International Conference on Computer Vision (ICCV)*, S. 1026–1034. *(Seite: 29)*

[He et al., 2016a] He, Kaiming, X. Zhang, S. Ren und J. Sun (2016a). *Deep residual learning for image recognition*. IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), S. 770–778. *(Seite: 5, 6, 21, 26, 50)*

[He et al., 2016b] He, Kaiming, X. Zhang, S. Ren und J. Sun (2016b). *Identity mappings in deep residual networks*. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) (LNCS), S. 630–645. *(Seite: 6, 26)*

[Hinton, 2012] Hinton, Geoffrey. (2012). *Neural Networks for Machine Learning - Lecture 6a - Overview of mini-batch gradient descent. (Seite: 74)*

[Hochreiter und Schmidhuber, 1997] Hochreiter, Sepp und J. Schmidhuber (1997). *Long Short-Term Memory.* Neural Computation, 9(8):1735–1780. *(Seite: 49)*

[Howard et al., 2019] Howard, Andrew, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan und Others (2019). *Searching for mobilenetv3.* In: *International Conference on Computer Vision (ICCV)*, S. 1314–1324. *(Seite: 20, 30, 31, 34, 42)*

[Hu et al., 2018] Hu, Jie, L. Shen und G. Sun (2018). *Squeeze-and-excitation networks.* In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, S. 7132–7141. *(Seite: 30)*

[Hu et al., 2019] Hu, Xinxin, K. Yang, L. Fei und K. Wang (2019). *ACNet: Attention Based Network to Exploit Complementary Features for RGBD Semantic Segmentation.* International Conference on Image Processing (ICIP). *(Seite: 47, 51, 54, 57, 60, 68, 72, 73, 100, 124)*

[Hua et al., 2019] Hua, Minjie, Y. Nan und S. Lian (2019). *Small Obstacle Avoidance Based on RGB-D Semantic Segmentation.* In: *International Conference on Computer Vision (ICCV). (Seite: 1)*

[Huang et al., 2017] Huang, Gao, Z. Liu, L. Van Der Maaten und K. Q. Weinberger (2017). *Densely Connected Convolutional Networks.* In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, S. 2261–2269. *(Seite: 7, 135, 136)*

[Hung et al., 2019] Hung, Shang-Wei, S.-Y. Lo und H.-M. Hang (2019). *Incorporating Luminance, Depth and Color Information by a Fusion-Based Network for Semantic Segmentation.* In: *International Conference on Image Processing (ICIP)*, S. 2374–2378. *(Seite: 47, 50)*

[Iandola et al., 2016] Iandola, Forrest N., S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally und K. Keutzer (2016). *SqueezeNet: AlexNet-level*

accuracy with 50x fewer parameters and <0.5MB model size. arXiv preprint arXiv:1602.07360. *(Seite: 50)*

[Ioffe und Szegedy, 2015] Ioffe, Sergey und C. Szegedy (2015). *Batch normalization: Accelerating deep network training by reducing internal covariate shift.* In: *International Conference on Machine Learning (ICML)*, S. 448–456. *(Seite: 6, 7)*

[Janoch et al., 2011] Janoch, A., S. Karayev, Y. Jia, J. Barron, M. Fritz, K. Saenko und T. Darrell (2011). *A Category-level 3-D Database: Putting the Kinect to Work.* In: *Internation Conference on Computer Vision (ICCV) Workshop on Consumer Depth Cameras for Computer Vision*, S. 141–165. *(Seite: 65)*

[Jiang et al., 2018] Jiang, Jindong, L. Zheng, F. Luo und Z. Zhang (2018). *RedNet: Residual Encoder-Decoder Network for indoor RGB-D Semantic Segmentation.* arXiv preprint arXiv:1806.01054. *(Seite: 47, 50, 54, 57, 60, 68, 70, 72, 73, 74, 100, 105, 109, 124)*

[Jiao et al., 2019] Jiao, Jianbo, Y. Wei, Z. Jie, H. Shi, R. Lau und T. S. Huang (2019). *Geometry-Aware Distillation for Indoor Semantic Segmentation.* In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, S. 2869–2878. *(Seite: 57, 58, 153)*

[Krizhevsky et al., 2012] Krizhevsky, Alex, I. Sutskever und G. E. Hinton (2012). *ImageNet classification with deep convolutional neural networks.* In: *International Conference on Neural Information Processing Systems (NIPS)*, S. 1097–1105. *(Seite: 12)*

[Lee et al., 2017] Lee, Seungyong, S. J. Park und K. S. Hong (2017). *RDFNet: RGB-D Multi-level Residual Feature Fusion for Indoor Semantic Segmentation.* International Conference on Computer Vision (ICCV), S. 4990–4999. *(Seite: 47, 53, 57)*

[Li et al., 2019a] Li, Gen, I. Yun, J. Kim und J. Kim (2019a). *DABNet: Depth-wise Asymmetric Bottleneck for Real-time Semantic Segmentation.* British Machine Vision Conference (BMVC). *(Seite: 20, 23, 28, 31, 32, 42, 88, 92, 97)*

[Li et al., 2019b] Li, Hanchao, P. Xiong, H. Fan und J. Sun (2019b). *DFANet: Deep Feature Aggregation for Real-Time Semantic Segmentation*. In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, S. 9522–9531. *(Seite: 20, 31, 43, 144)*

[Li et al., 2017] Li, Yabei, J. Zhang, Y. Cheng, K. Huang und T. Tan (2017). *Semantics-guided multi-level RGB-D feature fusion for indoor semantic segmentation*. In: *International Conference on Image Processing (ICIP)*, S. 1262–1266. *(Seite: 47, 52, 57)*

[Li et al., 2016] Li, Zhen, Y. Gan, X. Liang, Y. Yu, H. Cheng und L. Lin (2016). *Lstm-cf: Unifying context modeling and fusion with lstms for rgb-d scene labeling*. In: *European Conference on Computer Vision (ECCV)*, S. 541–557. Springer. *(Seite: 47, 49)*

[Lin et al., 2017] Lin, Guosheng, A. Milan, C. Shen und I. Reid (2017). *RefineNet: Multi-Path Refinement Networks for High-Resolution Semantic Segmentation*. In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, S. 1925–1934. *(Seite: 53)*

[Lin et al., 2020] Lin, Tsung Yi, P. Goyal, R. Girshick, K. He und P. Dollar (2020). *Focal Loss for Dense Object Detection*. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, Bd. 42, S. 318–327. *(Seite: 134)*

[Liu et al., 2019] Liu, Liyuan, H. Jiang, P. He, W. Chen, X. Liu, J. Gao und J. Han (2019). *On the Variance of the Adaptive Learning Rate and Beyond*. arXiv preprint arXiv:1908.03265. *(Seite: 74)*

[Liu und Yin, 2019] Liu, Mengyu und H. Yin (2019). *Feature Pyramid Encoding Network for Real-time Semantic Segmentation*. arXiv preprint arXiv:1909.08599. *(Seite: 20, 31, 35, 43, 141, 142)*

[Lo et al., 2019] Lo, Shao-Yuan, H.-M. Hang, S.-W. Chan und J.-J. Lin (2019). *Efficient dense modules of asymmetric convolution for real-time semantic segmentation*. In: *Proceedings of the ACM Multimedia Asia on ZZZ*, S. 1–6. *(Seite: 20, 23, 26, 31, 32, 42, 139)*

[Lowe, 1999] Lowe, David G (1999). *Object Recognition from Local Scale-Invariant Features*. In: *International Conference on Computer Vision (ICCV)*, S. 1150–1157. *(Seite: 68)*

[Lyu et al., 2019] Lyu, Haoran, H. Fu, X. Hu und L. Liu (2019). *Esnet: Edge-Based Segmentation Network for Real-Time Semantic Segmentation in Traffic Scenes*. In: *International Conference on Image Processing (ICIP)*, S. 1855–1859. *(Seite: 20, 29, 31, 37, 43)*

[Mahmood et al., 2018] Mahmood, Faisal, Z. Yang, T. Ashley und N. J. Durr (2018). *Multimodal Densenet*. arXiv preprint arXiv:1811.07407. *(Seite: 47, 49, 50)*

[McCormac et al., 2017] McCormac, John, A. Handa, S. Leutenegger und A. J. Davison (2017). *SceneNet RGB-D: Can 5M Synthetic Images Beat Generic ImageNet Pre-training on Indoor Segmentation?*. International Conference on Computer Vision (ICCV), S. 2697–2706. *(Seite: 65)*

[Mehta et al., 2019a] Mehta, Sachin, H. Hajishirzi und M. Rastegari (2019a). *DiCENet: Dimension-wise Convolutions for Efficient Networks*. arXiv preprint arXiv:1906.03516. *(Seite: 20, 31, 42)*

[Mehta et al., 2018] Mehta, Sachin, M. Rastegari, A. Caspi, L. Shapiro und H. Hajishirzi (2018). *ESPNet: Efficient Spatial Pyramid of Dilated Convolutions for Semantic Segmentation*. In: *European Conference on Computer Vision (ECCV)*, S. 552–568. *(Seite: 20, 27, 31, 33, 42, 88, 143)*

[Mehta et al., 2019b] Mehta, Sachin, M. Rastegari, L. Shapiro und H. Hajishirzi (2019b). *Espnetv2: A light-weight, power efficient, and general purpose convolutional neural network*. In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, S. 9190–9200. *(Seite: 20, 27, 28, 31, 32, 42, 88)*

[Nekrasov et al., 2018] Nekrasov, Vladimir, C. Shen und I. Reid (2018). *Light-Weight RefineNet for Real-Time Semantic Segmentation*. arXiv preprint arXiv:1810.03272. *(Seite: 20, 31, 40, 144)*

[Nguyen et al., 2019] Nguyen, Ty, S. S. Shivakumar, I. D. Miller, J. Keller, E. S. Lee, A. Zhou, T. Ozaslan, G. Loianno, J. H. Harwood, J. Wozencraft, C. J. Taylor und V. Kumar (2019). *MAVNet: An Effective Semantic Segmentation Micro-Network for MAV-Based Tasks.* In: *IEEE Robotics and Automation Letters (RA-L)*, S. 3908–3915. *(Seite: 20, 23, 32, 40, 140)*

[Oršić et al., 2019] Oršić, Marin, I. Krešo, P. Bevandić und S. Šegvić (2019). *In Defense of Pre-trained ImageNet Architectures for Real-time Semantic Segmentation of Road-driving Images.* IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), S. 12607–12616. *(Seite: 20, 31, 37, 43, 44, 60, 143)*

[Park et al., 2019] Park, Hyojin, L. L. Sjösund, Y. Yoo und N. Kwak (2019). *ExtremeC3Net: Extreme Lightweight Portrait Segmentation Networks using Advanced C3-modules.* arXiv preprint arXiv:1908.03093. *(Seite: 20, 40, 140, 141)*

[Park et al., 2018] Park, Hyojin, Y. Yoo, G. Seo, D. Han, S. Yun und N. Kwak (2018). *Concentrated-Comprehensive Convolutions for lightweight semantic segmentation.* arXiv preprint arXiv:1812.04920. *(Seite: 140)*

[Paszke et al., 2016] Paszke, Adam, A. Chaurasia, S. Kim und E. Culurciello (2016). *ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation.* arXiv preprint arXiv:1606.02147. *(Seite: 9, 20, 23, 25, 31, 32, 39, 71)*

[Paszke et al., 2019] Paszke, Adam, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai und S. Chintala (2019). *PyTorch: An Imperative Style, High-Performance Deep Learning Library.* In: *International Conference on Neural Information Processing Systems (NIPS)*, S. 8024–8035. *(Seite: 3, 74)*

[Peng et al., 2017] Peng, Chao, X. Zhang, G. Yu, G. Luo und J. Sun (2017). *Large kernel matters - Improve semantic segmentation by global convolutional network.* IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), S. 1743–1751. *(Seite: 150, 151)*

[Piramanayagam et al., 2018] Piramanayagam, Sankaranarayanan, E. Saber, W. Schwartzkopf und F. Koehler (2018). *Supervised classification of multisensor remotely sensed images using a deep learning framework*. Remote Sensing, 10(9). *(Seite: 47, 52, 152)*

[Poudel et al., 2019] Poudel, Rudra P K, S. Liwicki und R. Cipolla (2019). *Fast-SCNN: Fast Semantic Segmentation Network*. arXiv preprint arXiv:1902.04502. *(Seite: 20, 22, 29, 31, 37, 43, 143)*

[Qin et al., 2018] Qin, Zheng, Z. Zhang, D. Li, Y. Zhang und Y. Peng (2018). *Diagonalwise Refactorization: An Efficient Training Method for Depthwise Convolutions*. In: *International Joint Conference on Neural Networks (IJNNS)*, S. 1–8. *(Seite: 44)*

[Ramachandran et al., 2018] Ramachandran, Prajit, B. Zoph und Q. V. Le (2018). *Searching for activation functions*. In: *International Conference on Learning Representations (ICLR). (Seite: 31)*

[Romera et al., 2018] Romera, Eduardo, J. M. Alvarez, L. M. Bergasa und R. Arroyo (2018). *ERFNet: Efficient Residual Factorized ConvNet for Real-Time Semantic Segmentation*. IEEE Transactions on Intelligent Transportation Systems (ITS), S. 263–272. *(Seite: 20, 23, 25, 31, 32, 33, 42, 50, 88, 92)*

[Rusinkiewicz und Levoy, 2001] Rusinkiewicz, Szymon und M. Levoy (2001). *Efficient variants of the ICP algorithm*. International Conference on 3-D Digital Imaging and Modeling (3DIM), S. 145–152. *(Seite: 68)*

[Russakovsky et al., 2015] Russakovsky, Olga, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg und L. Fei-Fei (2015). *ImageNet Large Scale Visual Recognition Challenge*. International Journal of Computer Vision (IJCV), 115(3):211–252. *(Seite: 62, 65, 88, 136)*

[Sandler et al., 2018] Sandler, Mark, A. Howard, M. Zhu, A. Zhmoginov und L.-C. Chen (2018). *Mobilenetv2: Inverted residuals and linear bottlenecks*. In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, S. 4510–4520. *(Seite: 29, 30, 88)*

[Sifre, 2014] Sifre, Laurent (2014). *Rigid-Motion Scattering For Image Classification*. Phd thesis. *(Seite: 9)*

[Silberman et al., 2012] Silberman, Nathan, D. Hoiem, P. Kohli und R. Fergus (2012). *Indoor Segmentation and Support Inference from RGBD Images*. In: *European Conference on Computer Vision (ECCV). (Seite: 65)*

[Simonyan und Zisserman, 2014] Simonyan, Karen und A. Zisserman (2014). *Very Deep Convolutional Networks for Large-Scale Image Recognition*. arXiv preprint arxiv:1409.1556. *(Seite: 5, 32, 49)*

[Smith, 2018] Smith, Leslie N. (2018). *A disciplined approach to neural network hyper-parameters: Part 1 – learning rate, batch size, momentum, and weight decay*. arXiv preprint arXiv:1803.09820. *(Seite: 75)*

[Song et al., 2015] Song, Shuran, S. P. Lichtenberg und J. Xiao (2015). *SUN RGB-D: A RGB-D Scene Understanding Benchmark Suite*. In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, S. 567–576. *(Seite: 2, 3, 65, 66, 68)*

[Sutskever et al., 2013] Sutskever, Ilya., J. Martens, G. Dahl und G. Hinton (2013). *On the importance of initialization and momentum in deep learning*. In: *International Conference on Machine Learning (ICML)*, S. 1139–1147. *(Seite: 74)*

[Sutton und McCallum, 2011] Sutton, Charles und A. McCallum (2011). *An introduction to conditional random fields*. Foundations and Trends in Machine Learning, 4(4):267–373. *(Seite: 4)*

[Tan und Le, 2019] Tan, Mingxing und Q. V. Le (2019). *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*. International Conference on Machine Learning (ICML). *(Seite: 30, 89)*

[Treml et al., 2016] Treml, Michael, J. Arjona-medina, T. Unterthiner, R. Durgesh, F. Friedmann, P. Schuberth, A. Mayr, M. Heusel, M. Hofmarcher, M. Widrich, B. Nessler und S. Hochreiter (2016). *Speeding up Semantic Segmentation for Autonomous Driving*. In: *International Conference on*

Neural Information Processing Systems (NIPS) Workshop MLITS, S. 1–7. *(Seite: 20, 31, 33, 39)*

[Valada et al., 2019] Valada, Abhinav, R. Mohan und W. Burgard (2019). *Self-supervised model adaptation for multimodal semantic segmentation.* International Journal of Computer Vision (IJCV). *(Seite: 38, 39, 47, 53, 54, 57, 58, 60, 64, 98, 105, 106, 109, 113, 124)*

[Vu et al., 2019] Vu, Tuan-Hung, H. Jain, M. Bucher, M. Cord und P. Pérez (2019). *DADA: Depth-aware Domain Adaptation in Semantic Segmentation.* In: *International Conference on Computer Vision (ICCV)*, S. 7364–7373. *(Seite: 153)*

[Wang et al., 2016] Wang, Jinghua, Z. Wang, D. Tao, S. See und G. Wang (2016). *Learning common and specific features for RGB-D semantic segmentation with deconvolutional networks.* In: *European Conference on Computer Vision (ECCV)*, S. 664–679. *(Seite: 47, 48, 147, 148)*

[Wang et al., 2018] Wang, Panqu, P. Chen, Y. Yuan, D. Liu, Z. Huang, X. Hou und G. Cottrell (2018). *Understanding Convolution for Semantic Segmentation.* In: *Proceedings - 2018 IEEE Winter Conference on Applications of Computer Vision, WACV 2018*, S. 1451–1460. *(Seite: 138)*

[Wang und Neumann, 2018] Wang, Weiyue und U. Neumann (2018). *Depth-Aware CNN for RGB-D Segmentation.* In: *European Conference on Computer Vision (ECCV)*, S. 144–161. *(Seite: 56, 57, 154, 155)*

[Wang et al., 2019a] Wang, Yu, Q. Zhou, J. Liu, J. Xiong, G. Gao, X. Wu und L. J. Latecki (2019a). *LEDnet: A Lightweight Encoder-Decoder Network for Real-Time Semantic Segmentation.* In: *International Conference on Image Processing (ICIP)*, S. 1860–1864. *(Seite: 20, 21, 25, 26, 31, 32, 39, 42, 88, 139, 145, 146)*

[Wang et al., 2019b] Wang, Yu, Q. Zhou und X. Wu (2019b). *ESNet: An Efficient Symmetric Network for Real-time Semantic Segmentation.* In: *Chinese Conference on Pattern Recognition and Computer Vision (PRCV)*, S. 41–52. *(Seite: 20, 25, 26, 31, 32, 33, 42, 88, 91, 139)*

[Wang und Ji, 2018] Wang, Zhengyang und S. Ji (2018). *Smoothed Dilated Convolutions for Improved Dense Prediction.* International Conference on Knowledge Discovery & Data Mining (SIGKDD), S. 2486–2495. *(Seite: 138, 139)*

[Wu et al., 2018] Wu, Bichen, A. Wan, X. Yue, P. Jin, S. Zhao, N. Golmant, A. Gholaminejad, J. Gonzalez und K. Keutzer (2018). *Shift: A Zero FLOP, Zero Parameter Alternative to Spatial Convolutions.* IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), S. 9127–9135. *(Seite: 44)*

[Xiao et al., 2013] Xiao, Jianxiong, A. Owens und A. Torralba (2013). *SUN3D: A database of big spaces reconstructed using SfM and object labels.* International Conference on Computer Vision (ICCV), S. 1625–1632. *(Seite: 65)*

[Xing et al., 2019a] Xing, Yajie, J. Wang, X. Chen und G. Zeng (2019a). *2.5D Convolution for RGB-D Semantic Segmentation.* In: *International Conference on Image Processing (ICIP)*, S. 1410–1414. *(Seite: 56, 57, 58)*

[Xing et al., 2019b] Xing, Yajie, J. Wang, X. Chen und G. Zeng (2019b). *Coupling Two-Stream RGB-D Semantic Segmentation Network by Idempotent Mappings.* In: *International Conference on Image Processing (ICIP)*, S. 1850–1854. *(Seite: 47, 48, 57, 58, 148, 149)*

[Yu et al., 2018] Yu, Changqian, J. Wang, C. Peng, C. Gao, G. Yu und N. Sang (2018). *BiSeNet: Bilateral segmentation network for real-time semantic segmentation.* Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) (LNCS), S. 334–349. *(Seite: 20, 31, 43, 144, 145)*

[Yu und Koltun, 2016] Yu, Fisher und V. Koltun (2016). *Multi-scale context aggregation by dilated convolutions.* In: *International Conference on Learning Representations (ICLR). (Seite: 8)*

[Zhang et al., 2018] Zhang, Xiangyu, X. Zhou, M. Lin und J. Sun (2018). *ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices.* In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, S. 6848–6856. *(Seite: 12, 13, 14, 44)*

[Zhao et al., 2018] Zhao, Hengshuang, X. Qi, X. Shen, J. Shi und J. Jia (2018). *Icnet for real-time semantic segmentation on high-resolution images*. In: *European Conference on Computer Vision (ECCV)*, S. 405–420. *(Seite: 20, 23, 31, 37, 43, 138, 143)*

[Zhao et al., 2017] Zhao, Hengshuang, J. Shi, X. Qi, X. Wang und J. Jia (2017). *Pyramid scene parsing network*. In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR). (Seite: 1, 36, 37, 63, 138)*

[Zhong et al., 2018] Zhong, Yiran, Y. Dai und H. Li (2018). *3D Geometry-Aware Semantic Labeling of Outdoor Street Scenes*. In: *International Conference on Pattern Recognition (ICPR)*, S. 2343–2349. *(Seite: 56, 154)*

[Ziegler et al., 2019] Ziegler, Thomas, M. Fritsche, L. Kuhn und K. Donhauser (2019). *Efficient Smoothing of Dilated Convolutions for Image Segmentation*. arXiv preprint arXiv:1903.07992. *(Seite: 139)*