



ulm university universität
ulm

Fakultät für Ingenieurwissenschaften, Informatik und Psychologie
Institut für Mess-, Regel- und Mikrotechnik

Segmentierung von Punktwolken mit Neuronalen Netzen

Bachelorarbeit

von

Tarik Enderes

31.10.2019

Betreuer: Prof. Dr. rer. nat. Vasileios Belagiannis
1. Prüfer: Prof. Dr. rer. nat. Vasileios Belagiannis
2. Prüfer: Prof. Dr.-Ing. Klaus Dietmayer

Hiermit versichere ich, dass ich die vorliegende Arbeit mit dem Titel

Segmentierung von Punktwolken mit Neuronalen Netzen

bis auf die offizielle Betreuung selbstständig und ohne fremde Hilfe angefertigt habe und die benutzten Quellen und Hilfsmittel vollständig angegeben sind. Aus fremden Quellen direkt oder indirekt übernommene Gedanken sind jeweils unter Angabe der Quelle als solche kenntlich gemacht.

Ich erkläre außerdem, dass die vorliegende Arbeit entsprechend den Grundsätzen guten wissenschaftlichen Arbeitens gemäß der „Satzung der Universität Ulm zur Sicherung guter wissenschaftlicher Praxis“ erstellt wurde.

Ulm, den 31.10.2019

Tarik Enderes

Danksagung

Ich möchte ich bei allen bedanken, die mir geholfen haben, diese Arbeit anzufertigen. Insbesondere bedanke ich mich bei Prof. Dr. Vasileios Belagiannis für seine kompetente und geduldige Betreuung, sowie seine Vorschläge zur Gestaltung des Projekts. Ein besonderer Dank geht auch an Dipl.-Ing. Uwe Kerner und M. Sc. Nico Engel für ihre technische Unterstützung.

Inhaltsverzeichnis

1 Theorie	1
1.1 Segmentierung	1
1.1.1 Semantische Segmentierung	1
1.1.2 Instanz-Segmentierung	1
1.1.3 Panoptische Segmentierung	2
1.2 Technologien in DeepLab	2
1.2.1 Deep Convolutional Neural Networks für Semantische Segmentierung	3
Convolutional Neural Networks	3
Anpassungen für Semantische Segmentierung	3
1.2.2 Atrous Convolution	3
1.2.3 Atrous Spatial Pyramid Pooling	4
1.2.4 Fully-Connected Conditional Random Fields	6
1.2.5 Residual Networks	6
1.3 Kamerakalibrierung	6
2 Aufgabenstellung	7
2.1 Ziele und Anforderungen	7
2.2 Ähnliche Projekte	7
2.2.1 PointNet	7
2.2.2 UPSNet	7
3 Arbeitsmethodik	9
3.1 Integration von DeepLab	9
3.2 Backbones	9
3.2.1 Xception	9
3.2.2 MobileNetV2	9
3.3 Datensätze	9
3.3.1 Cityscapes	9
3.3.2 KITTI	9
4 Experimente	11
4.1 Technische Daten des für die Experimente verwendeten Rechners . . .	11

4.2	Backbones	11
4.2.1	Xeption	11
4.2.2	MobileNetV2	11
4.3	Auswirkungen der Trainingsdauer	11
4.4	Experimente mit Verfeinerung	11
4.4.1	Verfeinerung mit KITTI	11
4.5	Aufgetretene Probleme und Lösungen	11
4.5.1	False Positives	11
4.5.2	Overfitting	11
5	Ergebnisse	13
5.1	Ergebnisse auf Bildern	13
5.2	Ergebnisse auf Punktwolken	13
6	Zusammenfassung	15

1 Einleitung

Für zahlreiche Entwicklungsthemen der heutigen Zeit, wie beispielsweise autonomes Fahren [5940562], automatische Sicherheitsüberwachung [**ObjSegmentation**] und Augmented Reality [[doi:10.1162/pres.1997.6.4.355](https://doi.org/10.1162/pres.1997.6.4.355)], ist eine präzise Erkennung der Umweltbedingungen unerlässlich. Kameras und Laserscanner finden für diesen Zweck oft Verwendung, was die Verarbeitung von Bildern und Punktwolken zu einem verbreiteten Gegenstand moderner Forschung macht. Häufig wird zur Lösung dieser komplexen Probleme auf Elemente der Neuroinformatik zurückgegriffen.

Je nach Anwendungsfeld ist ein bestimmter Grad an Auswertung der gegebenen Daten erforderlich. Diese Arbeit befasst sich mit der Aufgabe, Bilder und Punktwolken für den Bereich des autonomen Fahrens zu segmentieren [**OffRoad**].

Das autonome Fahren setzt, wie in [5940562] beschrieben, eine Kette von Prozessen voraus. Diese umfasst die Aufnahme der Umgebung durch Messungen, die Erkennung der Situation durch Auswertung dieser Messungen, die Planung des weiteren Vorgehens und die Kontrolle des Fahrzeugs zur Ausführung der so berechneten Maßnahmen. Diese Arbeit befasst sich mit dem Schritt der Umgebungswahrnehmung aufgrund vorhandener Messdaten. Da es sich dabei letztendlich um ein dreidimensionales Problem handelt, nicht um ein zweidimensionales, sollen im Rahmen dieser Arbeit nicht nur Bilder, sondern auch Punktwolken für die Auswertung verwendet werden. Zu diesem Zweck ist eine zuverlässige, echtzeitfähige Segmentierung dreidimensionaler Strukturen wünschenswert, da somit eine weitgehend vollständige Erkennung der Umgebung, auf einem ähnlichen Niveau wie bei der menschlichen Wahrnehmung möglich ist und durchzuführende Aktionen aufgrund eines möglichst großen Informationsgehalts geplant werden können.

Die Durchführung von Bild- und Punktwolkensegmentierung sowie ihr Einsatz im Bereich des autonomen Fahrens ist eine derzeit offene Aufgabe. Das Erzeugen von qualitativ hochwertigen Ergebnissen bei gleichzeitiger Gewährleistung von Echtzeitfähigkeit stellt ein Problem dar. Eine weitere besondere Schwierigkeit im Umgang mit Punktwolken kommt daher, dass sie im Allgemeinen ungeordnete Strukturen sind, was den Einsatz Neuronaler Netze für deren Verarbeitung nicht trivial macht, wie in [**PointNet**] dargelegt. Die Handhabung derartiger Strukturen fällt in den Bereich der Geometrie. Neben Neuroinformatik kommt deshalb auch der dreidimensionalen Geometrie, insbesondere der Projektionsgeometrie, eine wichtige Rolle in dieser Arbeit zu.

1.1 Segmentierung

Segmentierung bezeichnet einen Vorgang, bei dem ein Bild nach bestimmten Homogenitätskriterien in inhaltlich zusammenhängende Regionen eingeteilt wird. Von den verschiedenen Ansätzen, die das erreichen sollen, befasst sich diese Arbeit mit pixelbasierten Verfahren, bei denen jedem Pixel in einem Bild eine Klasse zugeordnet wird. Man unterscheidet zwischen den in [UPSNet] beschriebenen Bereichen semantische Segmentierung, Instanz-Segmentierung und panoptische Segmentierung und der in [DBLP:journals/corr/abs-1904-09172] ausgeführten Objekt-Segmentierung. Für weitere Informationen siehe [gonzalez2008digital].

1.1.1 Semantische Segmentierung

Bei der semantischen Segmentierung soll jeder Pixel eine valide Klasse erhalten. Es wird dabei nicht zwischen unterschiedlichen Instanzen einer Objektklasse unterschieden. Wenn beispielsweise auf einem Bild zwei Fahrzeuge zu sehen sind und bei der Segmentierung die Klasse „Fahrzeug“ zugeteilt werden soll, erhalten die Pixel beider Fahrzeuge das Label „Fahrzeug“. Die Anzahl valider Klassen bleibt somit bei jedem prozessierten Bild gleich.

Einige Anwendungsgebiete von semantischer Segmentierung sind autonomes Fahren im Gelände [OffRoad], Zellanalyse in der Biomedizin [journals/corr/RonnebergerFB15] und Auswertung von Satellitenbildern für Kartographie [DBLP:journals/corr/abs-1904-03983].

1.1.2 Instanz-Segmentierung

Im Gegensatz zur semantischen Segmentierung werden bei der Instanz-Segmentierung nurzählbare Objekte betrachtet und deren Instanzen berücksichtigt. Übertragen auf vorheriges Beispiel würden die Pixel des einen Fahrzeug ein Label wie „Fahrzeug1“ und die des anderen analog „Fahrzeug2“ erhalten.

Instanz-Segmentierung findet beispielsweise Anwendung zur Detektion von Personen in Videodaten für Verhaltensanalysen und Überwachung [HumanSegmentation].

1.1.3 Panoptische Segmentierung

Die panoptische Segmentierung stellt eine Kombination der vorherigen Segmentations-Arten dar. Zählbare Objekte werden demnach nach dem Prinzip der Instanz-Segmentierung und amorphe nach dem der semantischen Segmentierung segmentiert. Die Ergebnisse beider Verfahren werden anschließend kombiniert.

1.1.4 Objekt-Segmentierung

Bei der Objekt-Segmentierung soll für jeden Pixel eines Bildes entschieden werden, ob er Teil des Vorder- oder des Hintergrundes ist, weshalb sie häufig als Vordergrund-Hintergrund-Segmentierung bezeichnet wird. Von Interesse ist dabei nur, wo sich Objekte im Bild befinden, nicht, wie bei den anderen Disziplinen, worum es sich handelt. Oft ist das Ziel dabei die Erkennung von Bewegung in Videodaten. Objekt-Segmentierung findet Verwendung im Bereich der Videoüberwachung [**ObjSegmentation**].

1.2 Projektive Geometrie

Wie in [**Hartley**] beschrieben, geht die Projektion dreidimensionaler Objekte auf eine zweidimensionale Ebene mit einem Verlust an Informationen einher. Ein perfekter Kreis erscheint auf einem Bild in der Regel als Ellipse, ein rechter Winkel erscheint je nach Perspektive stumpf oder spitz. Ein Pkw kann, entsprechend in Szene gesetzt, aussehen wie ein Spielzeug und ein paralleles Paar Schienen schneidet sich am Horizont. Mit diesen und anderen Phänomenen beschäftigt sich eine Unterart der Geometrie, die projektive Geometrie. Um von einer zweidimensionalen Darstellung auf die dreidimensionale Wirklichkeit zu schließen, werden Methoden aus diesem Bereich angewandt. Für den Zweck dieser Arbeit ist die Kamerakalibrierung [**Zhang:2000:FNT:357014.357025**], auf die in Abschnitt ?? eingegangen wird, von besonderer Bedeutung.

projektive Geometrie spielt beispielsweise eine wichtige Rolle bei der Verwendung und Verbesserung von Röntgengeräten [**10.1117/12.479945**] und Mikroskopen [**Projection**].

1.3 Ziele und Anforderungen

Ziel der Arbeit ist es, ein System zu entwickeln, das mit Hilfe von Neuronalen Netzen und dreidimensionaler Geometrie jedem Punkt einer Punktwolke eine Klasse zuweist. Dazu soll ein Bild semantisch segmentiert und die so erhaltenen Ergebnisse auf eine analog dazu aufgenommene Punktwolke projiziert werden. Mit diesem Vorgehen soll die Entwicklung im Bereich der digitalen Bildsegmentierung der letzten Jahre ausgenutzt werden. Für die Ausführung dieses Verfahrens sind offensichtlich sowohl Bilder als auch Punktwolken zur Erkennung eines Szenarios notwendig. Entsprechend werden Daten aus mehreren Sensoren wie Stereo-Kameras und Laserscannern benötigt. Eine Aufgabe wird also sein, einen entsprechenden Datensatz auszuwählen, der diese Anforderungen erfüllt.

Der Anwendungsbereich des Systems soll autonomes Fahren sein, weshalb Entwicklung und Experimente mit Datensätzen für diesen durchgeführt werden sollen. Besondere Wichtigkeit kommt dabei der Erkennung von Fahrzeugen, Personen und Straßen zu, da diese den größten Einfluss auf Entscheidungen bezüglich des Verhaltens im Straßenverkehr haben. Eine Kernanforderung ist dabei Echtzeitfähigkeit. Optimierung der Laufzeit hat dem entsprechend Priorität. Weiterhin soll das System transportabel, leicht zu verwenden, benutzerfreundlich und ressourcenschonend sein.

Die Entwicklung des Systems in Python soll die Implementierung transportabel machen. Dazu sollen Zeitintensive Berechnungen durch CUDA-Unterstützung auf die Grafikkarte ausgelagert werden, was für die Verbesserung der Laufzeit essentiell und bei der Verwendung von Neuronaler Netze üblich ist. Als Framework zur Bildsegmentierung soll DeepLab [**DeepLab2**] genutzt werden, das zur Zeit der Entstehung dieser Arbeit als State-of-the-Art angesehen wird. Den Backbone dafür soll MobileNetV2 [**MobileNetV2**] bilden, eine ressourcenschonende Netzwerkarchitektur, die sich durch Geschwindigkeit auszeichnet. Der Begriff „Backbone“ beschreibt in dieser Arbeit, analog zu [**UPSNet**], eine Netzarchitektur, die ein Bild als Eingabe erhält und eine Feature Map ausgibt. Der Begriff ist synonym mit „Feature Extractor“.

Nach der Entwicklung des Systems sollen dessen Eigenschaften durch Experimente genauer untersucht und diskutiert werden. Im Vordergrund steht dabei die Untersuchung des Neuronalen Netzes. Die Eigenschaften von MobileNetV2 sollen ausgewertet und mit denen der Architektur Xception65 [**Xception**] verglichen werden. Außerdem sollen die Auswirkungen verschiedener Trainingsmuster ermittelt werden.

2 Literatur

2.1 Digitale Bildverarbeitung

Das Feld der digitalen Bildverarbeitung umfasst nach [gonzalez2008digital] die Verbesserung der Bildinformation für menschliche Betrachter, sowie die Verarbeitung von Daten aus Bildern zur Übertragung oder autonomen maschinellen Erkennung durch digitale Rechner.

Der erste bekannte Einsatz digitaler Bildverarbeitung fand 1964 im Zuge des Raumfahrprogramms der USA statt, wobei ein Computer eingesetzt wurde, um Störungen in Bilder von der Mondoberfläche zu Korrigieren. Eine weitere Nennenswerte Entwicklung auf dem Gebiet ist die Erstellung von Röntgenbildern im Jahr 1979. Die Bedeutung digitaler Bildverarbeitung in der Geschichte steht in einem proportionalen Verhältnis zu den Möglichkeiten der Datenübertragung und -speicherung. Analog dazu erreicht sie ihren Durchbruch mit dem Aufkommen des World Wide Web um 1989. Zu den Aufgabe digitaler Bildverarbeitung gehören Bildbearbeitung, Bildaufbereitung, Bildkompression, Bildsegmentierung, Repräsentation und Objekterkennung. Die letzten drei Kategorien können einem Unterbereich der Digitalen Bildverarbeitung, dem maschinellen Sehen [Forsyth:2002:CVM:580035] zugewiesen werden.

2.1.1 Maschinelles Sehen

Maschinelles Sehen ist der Bereich der digitalen Bildverarbeitung, der sich mit der automatischen Analyse und Auswertung von Bilddaten mit Hilfe von Computern befasst. Zu den häufigsten Einsatzgebieten zählen die Automation, beispielsweise in Landwirtschaft [MANH2001139] und Industrie [Baird:1977:IST:1622943.1622976], und Mensch-Maschine-Schnittstellen, wie zum Beispiel in wahrnehmungsgesteuerten Benutzerschnittstellen [Bradska98computervision] und Augmented Reality [FeaturePointExt]. Natürlich spielt maschinelles Sehen auch eine wichtige Rolle beim autonomen Fahren [DICKMANNS1987221]. Das maschinelle Sehen kann unterteilt werden in:

Bildsegmentierung Bei der Segmentierung wird versucht, Punkte oder Bereiche von besonderem Interesse in einem Bild zu finden, um das Bild maschinell erkennbar zu machen. Für mehr Informationen siehe Abschnitt ??.

Representation und Beschreibung Dieser Bereich der Bildverarbeitung beschäftigt sich mit der Anfertigung mathematischer Beschreibungen von Bildern wie zum Beispiel die statistische Verteilung gerichteter Linien. Ein häufiges Mittel zur Beschreibung von Bildern ist „Scale Invariant Feature Transform“ (SIFT) [Lowe:1999:ORL:850924.851523]. Dabei handelt es sich um einen Algorithmus, der durch eine Reihe an Filter-Operationen aus einem Bild eine Menge von lokalen Feature-Vektoren berechnet, die in gewissen Maße invariant gegenüber Translation, Rotation, Größe und Helligkeit sind.

Objekterkennung Ziel der Objekterkennung ist es, Objekte auf Bildern zu detektieren, sodass dargestellte Szenen maschinell erkannt werden können. Übliche Methoden sind Bildvergleiche und Deep Learning. Ein konkretes Beispiel für einen Objekterkennungs-Algorithmus ist „You Only Look One“ (YOLO) [Redmon_2016_CVPR], das Bounding Boxes und Klassifizierungen für ein Bild mit Hilfe eines Convolutional Neural Networks [Goodfellow-et-al-2016] berechnet.

2.2 Digitale Bildsegmentierung

Die automatische Segmentierung digitaler Bilder zählt zu den kompliziertesten Problemen der digitalen Bildverarbeitung und ist, wie in [DBLP:journals/corr/abs-1904-09172] erläutert, unerlässlich für das Ziel der Objekterkennung. Aufgrund der Komplexität der Aufgabe verwenden viele moderne Methoden Neuronale Netze zu diesem Zweck. Als besonders geeignet gelten auf Convolutional Neural Networks basierende Architekturen. Nach wie vor kommen auch klassische Methoden der Bildsegmentierung zum Einsatz.

2.2.1 Klassische Methoden der Segmentierung

Als klassisch werden im Zuge dieser Arbeit alle Methoden der Segmentierung bezeichnet, die keine Prinzipien der Neuroinformatik ausnutzen und stattdessen auf

den Grundlagen von Diskontinuität und Ähnlichkeit basieren. Dazu gehören nach [gonzalez2008digital]:

Detektion von Diskontinuitäten Auch das Detektieren von Punkten, Linien und Kanten, die unter dem Begriff Diskontinuitäten zusammengefasst werden können, gehört zum Bereich der Bildsegmentierung. Um das zu erreichen werden im Bild nach Stellen gesucht, an denen sich der Farbwert der Pixel räumlich rapide ändert. Im einfachsten Fall kann dies, wie in [edgeseg], mittels Faltung mit einem Filter bewerkstelligt werden, der auf starke Änderungen des Farbwertes in eine bestimmte Richtung reagiert.

Segmentierung nach Schwellwerten Die trivialste Methode der Segmentierung ist die nach Schwellwerten. Dabei werden Pixel oder Superpixel anhand ihres Farbwertes eingeordnet. Ein Beispiel stellt [679444] dar.

Regionen-basierte Segmentierung Bei Regionen-basierter Segmentierung werden räumliche Homogenitätskriterien ausgenutzt, um ein Bild in Bereiche einzuteilen. Die zwei verbreitetsten klassischen Ansätze dafür sind „Region Growing“ [295913] und „Region Splitting and Merging“ [CHEN1991457]. Beim *Region Growing* werden eine Anzahl von Pixel im Bild ausgewählt. Die an diese angrenzenden Pixel werden untersucht und der Region hinzugefügt, wenn sie die entsprechenden Bedingungen erfüllen. Dann werden die Nachbarpixel der hinzugefügten Pixel untersucht und auf diese Weise die Region rekursiv aufgebaut.

Beim *Region Splitting and Merging* wird das Bild in Bereiche aufgeteilt bis alle damit entstandenen Regionen intern die Homogenitätsbedingungen erfüllen (Splitting). Danach werden benachbarte Regionen auf Ähnlichkeit untersucht und gegebenenfalls zusammengeführt (Merging).

Wasserscheide-Verfahren Das in [736688] beschrieben Wasserscheide(Watershed)-Verfahren nutzt lokale Minima und Maxima aus, um Regionen zu bilden. Das (Graustufen-)Bild wird dazu als dreidimensional betrachtet, wobei der Farbwert als Höhe interpretiert wird. Man stelle sich vor, die so entstandene Struktur werde ausgehend von den lokalen Minima geflutet und dort abgegrenzt, wo sich zwei Wasserflächen treffen würden. Auf diese Weise entstehen Regionen im Bild anhand der Lage von eindimensionalen lokalen Maxima, die zweidimensionale miteinander verbinden.

Segmentierung anhand von Bewegung Bei der Verarbeitung von Videodaten kann auch die zwischen den einzelnen Bildern stattfindende Bewegung zur Segmentierung ausgenutzt werden, wie beispielsweise in [1199481]. Dazu gibt es zahlreiche Ansätze. Die einfachsten Fälle sind Verfahren, bei denen Bilder pixelweise mitein-

ander Verglichen werden, um ein Differenzbild zu erstellen, wie es beispielsweise bei Background Subtraction der Fall ist. Dabei geht der Algorithmus davon aus, dass die aufnehmende Kamera statisch und der Hintergrund damit unbeweglich ist. Pixel deren Farbwerte sich rapide ändern werden dementsprechend als Vordergrundobjekte gewertet.

2.2.2 Segmentierung mit Neuronalen Netzen

Wie bereits erwähnt sind Neuronale Netze ein beliebtes Mittel zur digitalen Bildsegmentierung. Auf einige Ansätze in diesem Gebiet soll hier eingegangen werden. Oft ist es sinnvoll, Methoden die Machine Learning verwenden danach einzuteilen, wie intensiv der Lernvorgang von einer Person überwacht werden muss. An dieser Stelle wird unterschieden zwischen unüberwachtem, schwach überwachtem und überwachtem Lernen.

Unüberwachtes Lernen findet Anwendung im Bereich der Objekt-Segmentierung. Der Lern-Prozess konzentriert sich dabei meistens auf Bewegungen in Videodaten. Ein Beispiel stellt die Architektur aus [DBLP:journals/corr/abs-1805-07780] dar, die Arcade-Spiele durch Reinforcement Learning lernt. Das Netz erhält zwei Frames als Eingabe und berechnet für jeden eine bestimmte Anzahl Objektmasken und entsprechende Translationen, sowie die Kamerabewegung. Beim Lern-Vorgang wird der Optische Fluss anhand der Ergebnisse ermittelt und eine Fehlerfunktion danach berechnet, wie genau der erste Frame aus dem zweitem und dem Optischen Fluss hervorgeht.

Es existieren Ansätze zur Bildsegmentierung mit Neuronalen Netzen, die als schwach überwachtes Lernen bezeichnet werden können. Das Netz erhält dabei Bilder, von denen bekannt ist, ob sich ein bestimmtes Objekt darin befindet oder nicht. Die Idee ist, Objekte die in beiden Arten von Bildern vorkommen als Hintergrund zu erkennen und zu ignorieren. In [10.1007/978-3-642-33863-2_20] wird ein Verfahren vorgestellt, dass mit großen Mengen zufällig ausgewählter, eventuell verrauschter Videos der Plattform YouTube arbeitet. Dabei wird ein Bild nach klassischen Segmentierungsverfahren in Regionen unterteilt. Die einzelnen Regionen werden von einem Netz danach bewertet, mit welcher Wahrscheinlichkeit sie zu dem gelernten Objekt gehören und anhand der Segmente mit hohen Wahrscheinlichkeiten wird, ebenfalls mit klassischen Methoden, eine Maske erstellt, die das gesamt Objekt umfassen soll.

Wenn an ein Verfahren hohe Anforderungen gestellt werden, wie bei der panoptischen oder Instanz-Segmentierung ist eine überwachte Lern-Methode oft unerlässlich. Für jedes Bild eines Trainingssatzes muss dafür eine Ground Truth zur Verfügung stehen, die das gewünschte Ergebnis des Netzes darstellt. Für Beispiele für diese Art von Verfahren siehe Abschnitt ??.

2.3 Arbeiten über Segmentierung durch überwachtes Lernen

In diesem Abschnitt wird auf vorhandene Arbeiten eingegangen, die sich mit der Problematik der Segmentierung von Bildern oder Punktwolken mit neuronalen Netzen befassen.

2.3.1 PointNet

Das 2017 in [PointNet] vorgestellte PointNet ist ein neuronales Netzwerk zum Auswerten von Punktwolken. Das Netz bietet dabei sowohl eine Architektur zur Klassifizierung als auch eine zur Segmentierung. Der Strukturelle Aufbau ist in Abbildung ?? dargestellt. Problematisch an der Auswertung von Punktwolken mit

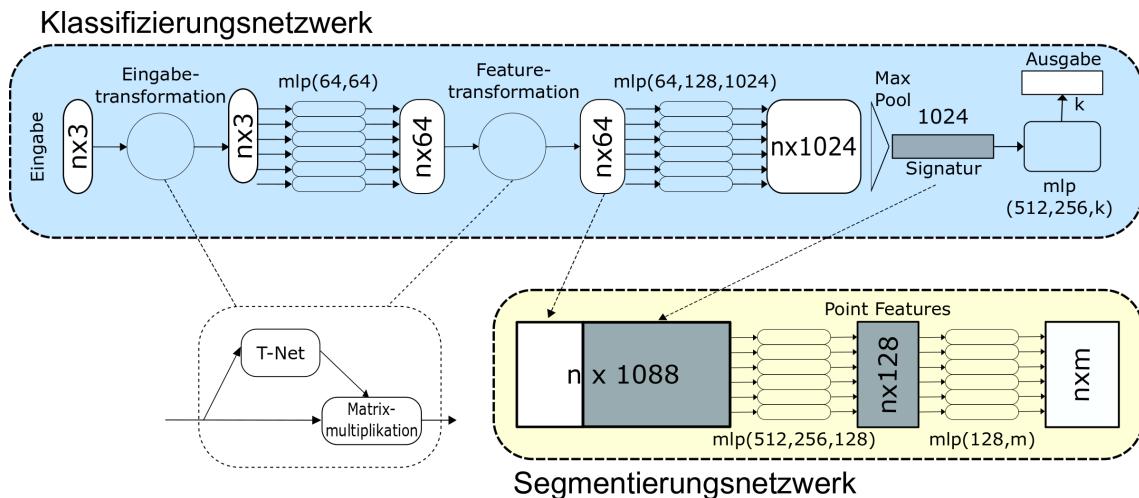


Abbildung 2.1: Architektur von PointNet nach [PointNet]. Aus der Eingabe, die aus einer im Allgemeinen ungeordneten Menge an Punkten besteht, wird durch eine Reihe von Transformationen mittels Neuronaler Netze eine globale Signatur erstellt. Zu Klassifizierungsaufgaben kann dieser globale Feature-Vektor von einem weiteren Netz direkt weiterverarbeitet werden. Um eine Segmentierung durchzuführen, wird aus den lokalen und globalen Informationen ein neuer Vektor gebildet, der dann von einem Netz ausgewertet werden kann.

Technologien der Neuroinformatik ist vor allem, dass die Daten im Allgemeinen

ungeordnet sind. Das Netzwerk erzeugt darum aus einem Eingabevektor, der aus einer Menge von Koordinaten gebildet wird zunächst durch Feature Transformation eine globale Signatur, also einen Feature-Vektor, der unabhängig von der Reihenfolge der Eingabegrößen ist. PointNet erreicht dies durch den Einsatz von Max-Pooling. Um Invarianz bezüglich bestimmter räumlicher Transformationen wie z.B. Rotation zu erreichen, wird bei der Erstellung dieses globalen Feature-Vektor mit einem kleinen neuronalen Netz, dem „T-Net“, eine Transformationsmatrix angenähert und auf die Eingabedaten angewandt. Mit der so berechneten Signatur kann ein weiteres Netz, in diesem Fall ein MLP, trainiert werden, das diese klassifiziert. Da der globale Feature-Vektor keine Ortsinformationen enthält, ist es nicht möglich, damit eine Segmentierung durchzuführen. Soll das Netzwerk also für diesen Zweck verwendet werden, wird der globale Feature-Vektor mit dem Eingabevektor kombiniert, um einen Vektor zu erzeugen, der sowohl globale als auch lokale Eigenschaften repräsentiert. Anschließend kann ein Label für jeden Punkt geschätzt werden.

2.3.2 UPSNet

Das 2019 in [UPSNet] vorgestellte UPSNet (Unified Panoptic Segmentation Network) ist ein neuronales Netz für panoptische Segmentierung. Dazu führt das Netzwerk parallel eine semantische Segmentierung und eine Instanzsegmentierung des Eingabebildes durch und erstellt mit den kombinierten Ausgaben beider Methoden einen Tensor von Wahrscheinlichkeiten für jede Klasse und Instanz. Aus diesem Tensor wird in einem letzten Schritt ein Ausgabebild erzeugt. Der Aufbau des Netzwerks ist in Abbildung ?? dargestellt.

UPSNets verwendet als Backbone das in [MaskRCNN] beschriebene Mask R-CNN, das sich aus dem ResNet [DBLP:journals/corr/HeZRS15] und dem Feature Pyramid Network [FPN] ableitet. Mask R-CNN führt eine Instanz-Segmentierung des Bildes durch und erzeugt parallel dazu eine Maske für jedes erkannte Objekt. Die Ausgabe des Backbones wird von zwei leichtgewichtigen Netzen, dem „Semantic Segmentation Head“, der semantisch segmentiert und dem „Instance Segmentation Head“, der eine Instanzsegmentierung durchführt unabhängig voneinander weiterverarbeitet. Die Implementierung eines einzelnen Backbones spart Rechenzeit und Speicherplatz gegenüber Architekturen mit zwei getrennten Netzen. Die daraus entstandenen Ergebnisse werden von dem „Panoptic Segmentation Head“ anhand einer Heuristik ausgewertet, um die Netzwerkausgabe zu erstellen.

Der „Instance Segmentation Head“ folgt dem Konzept von Mask R-CNN und erzeugt eine Anzahl von Bounding Boxes und Masken. Der „Semantic Segmentation Head“ ist ein CNN, das einen vom Backbone erzeugten Feature-Vektor als Eingabe erhält und mittels Soft-Max die Klasse jedes Pixels schätzt. Der „Panoptic Segmentation Head“ ermittelt zuerst die Anzahl von im Bild vorhandener Instanzen und erstellt einen

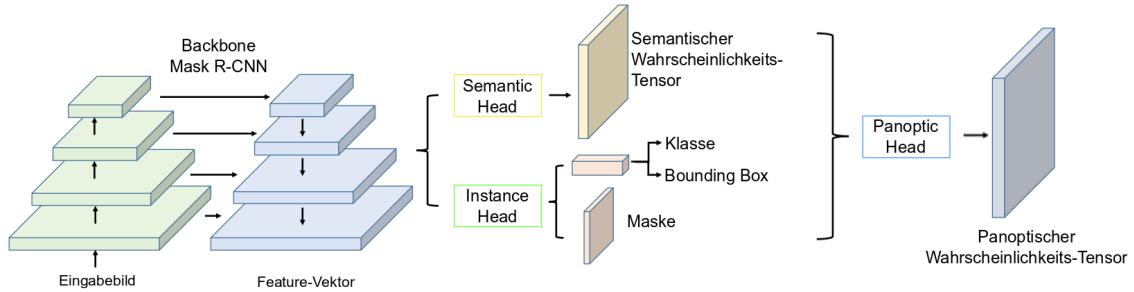


Abbildung 2.2: Architektur von UPSNet nach [UPSNet]. Ein Mask R-CNN fungiert als Backbone des Netzes. Es erstellt einen Feature Vektor aufgrund der Eingabe, der dann von zwei leichtgewichtigen Netzen, dem "Semantic Head" und dem "Instance Head" verarbeitet wird. Die so entstandenen Ausgaben werden kombiniert und in dem "Panoptic Head" anhand einer Heuristik ausgewertet.

Tensor aus den von den beiden vorherigen Köpfen errechneten Wahrscheinlichkeiten und führt eine Soft-Max Berechnung durch. Aus dem Resultat wird anhand einer Heuristik für jeden Pixel entschieden, ob er einer Instanz eineszählbaren Objekts und welcher Klasse er angehört. Es ist dabei möglich, dass Pixel als unbekannt klassifiziert werden, was den IoU der Ergebnisse durch die Verminderung von False Positives verbessert.

2.3.3 DeepLab

Das in [DeepLab1] und [DeepLab2] vorgestellte DeepLab ist ein Framework zur semantischen Bildsegmentierung, das auf Deep Convolutional Neural Networks und Fully Connected Conditional Random Fields [CRF] basiert. Die derzeit aktuelle Version ist DeepLabV3+ [DBLP:journals/corr/abs-1802-02611]. Für weitere Informationen siehe Abschnitt ??.

2.4 Projektive Geometrie

Projektive Geometrie entstammt, wie in [doi:10.1142/9789812384737_0010] erklärt, der Geometrie von Perspektive, die erstmals im 16. Jahrhundert von Malern der italienischen Renaissance untersucht wurde und der Photogrammetrie, die im Zuge der Entdeckung der Fotografie im 19. Jahrhundert entstand. Die Forschung in diesem

Bereich gilt seit dem Beginn des 20. Jahrhunderts als abgeschlossen.

In [**Richter-Gebert:2011:PPG:2031391**] wird (reelle) projektive Geometrie als Erweiterung der (herkömmlichen) euklidischen Geometrie durch das Hinzufügen von unendlich weit entfernten Punkten beschrieben. Es heißt darin weiterhin, eine Anzahl paralleler Linien schneidet sich an einem Punkt im Unendlichen und alle Punkte im Unendlichen liegen auf einer Linie im Unendlichen. Auf diese Weise wird vermieden, den Spezialfall von parallelen Linien berücksichtigen zu müssen. Realisiert wird dies durch die Verwendung von homogenen Koordinaten, wie in [**Hartley**] erklärt. Mehr dazu in Abschnitt ??.

Ein Anwendungsbereich projektiver Geometrie aus dem Bereich des maschinellen Sehens, der, auch durch die Thematik des autonomen Fahrens, derzeit von Interesse ist, ist die Rekonstruktion dreidimensionaler Szenen aus mehreren Bildern mit unterschiedlicher Perspektive, die in [**Fusiello06elements of**] und [**mohr:inria-00548361**] behandelt wird. Des weiteren findet projektive Geometrie Anwendung bei der Kameralkalibrierung, auf die in Abschnitt ?? weiter eingegangen wird.

3 Grundlagentheorie

3.1 Convolutional Neural Networks

Wie in [Goodfellow-et-al-2016] beschrieben, handelt es sich bei Convolutional Neural Networks (CNNs) um Neuronale Netze, die in mindestens einer Verarbeitungsschicht Faltung an Stelle von Matrixmultiplikation als mathematische Operation durchführen. Der Begriff Faltung bezieht sich dabei nicht auf die streng mathematischen Definition. In der Regel wird eine Variation eingesetzt. So wird in Faltungsschichten neben der Größe des Kernels oft noch eine Schrittgröße festgelegt, die angibt, ob und wie viele Werte der Ausgabematrix bei der Berechnung übersprungen werden. Bei einer Schrittgröße von zwei etwa wird nur jeder zweite Wert des Faltungs-Ergebnisses berechnet, was in einer entsprechend kleineren Ausgabematrix resultiert. Verwendet das Netz ausschließlich Faltung spricht man von einem Fully Convolution Neural Network. CNNs eignen sich zur Anwendung auf rasterförmige Datenstrukturen und werden aufgrund ihrer im Folgenden beschriebenen Eigenschaften häufig zur Bildverarbeitung eingesetzt.

Ein Vorteil von Faltung gegenüber Matrixmultiplikation ist, dass die Größe der Eingabematrix variabel ist. Im Fall von Bildbearbeitung bedeutet das, dass ein Fully Convolution Neural Network Bilder unabhängig von deren Größe und Auflösung verarbeiten kann. Es ist zu beachten, dass damit nicht Größeninvarianz erreicht wird. Bei der Faltung einer Matrix mit einem Kernel ist jeder Wert des Ergebnisses nur abhängig von bestimmten Werten der Eingabematrix, nicht unbedingt von allen. Für semantische Segmentierung bedeutet das, dass der Ausgabewert für einen Pixel nur von Pixeln in einem begrenzten Bereich des Eingabebildes, dem Sichtfeld, bestimmt wird. Durch Verknüpfung mehrerer Faltungsschichten wird dieses Sichtfeld vergrößert. Außerdem wird jeder Wert der Eingabematrix auf dieselbe Weise verarbeitet. Damit werden die Ergebnisse der Faltungsschichten in einem Netzwerk Equivariant gegenüber Translation. Das bedeutet, wenn die Eingabe verschoben ist, tritt die gleichen Verschiebung in der Ausgabe auf. Die Größe des Faltungskernels kann theoretisch frei gewählt werden und ist im Fall von Bildverarbeitung in der Regel vernachlässigbar klein verglichen mit den Eingabedaten, was CNNs deutlich effizienter im Bezug auf Laufzeit und besonders Speicherbedarf macht.

Üblicherweise wird in CNNs eine Pooling genannte Operation eingesetzt. Beim Pooling beziehungsweise Downsampling wird aus einer Matrix eine andere, meistens kleinere erstellt, die eine Zusammenfassung der Originalmatrix darstellt. Es gibt verschiedene Arten von Pooling. Häufig verwendet wird so genanntes Max-Pooling, bei dem jeder Eintrag der Ausgabematrix das Maximum eines rechteckigen Bereichs der Eingabematrix ist. Durch Pooling soll das Netz Resistenter gegenüber kleinen Änderungen der Eingabedaten werden und die Größe für weitere Verarbeitungsschichten verringert werden, um die Laufzeit zu verbessern. Typischerweise folgt eine Pooling-Schicht auf eine oder mehrere Faltungsschichten.

In der Regel enthalten CNNs auch "Fully Connected Layers", bei denen, wie in einem klassischen Netzwerk, jeder Wert der Ausgabe von jedem Wert der Eingabe abhängt. Diese Schichten werden oft aus MLPs aufgebaut und befinden sich meistens am Ende des Netzes.

3.2 Atrous Convolution

Atrous Convolution, auch Dilated Convolution genannt, beschreibt eine Technik bei der eine Matrix mit einem spärlich bestückten Kernel gefaltet wird, wie in Abbildung 1.2 illustriert.

Die Abstände der zu berücksichtigenden Werte in der Matrix wird dabei durch die so genannte Dilation Rate bzw. Erweiterungsrate festgelegt. Das Tatsächliche Sichtfeld des Filters wird also durch die Größe des Kernels und die Rate bestimmt.

ein Filter mit einem Kernel der Größe 3x3 und einer Rate von 2, was dem Einfügen einer leeren Zeilen und Spalte zwischen den Werten entspricht, hat demnach ein Sichtfeld der Größe 5x5. Dadurch wird das effektive Sichtfeld des Filters erhöht und es kann eine höhere Auflösung bei gleichen Rechenaufwand erreicht werden. Vor allem kann Downsampling damit vermieden werden. Die Vorteile der Verwendung von Atrous Convolution für Bildsegmentierung sind in Abbildung 1.3 dargestellt.

3.3 Atrous Spatial Pyramid Pooling

Beim Atrous Spatial Pyramid Pooling werden, wie in [DeepLab2] beschrieben, mehrere parallele Convolutional Layers, die Atrous Convolutional Layers mit unterschiedlicher Erweiterungsrate verwenden, in das DCNN eingebaut. Aus den Ergebnissen der Verarbeitungszweige wird ein Tensor gebildet, der anschließend einer Dimension-

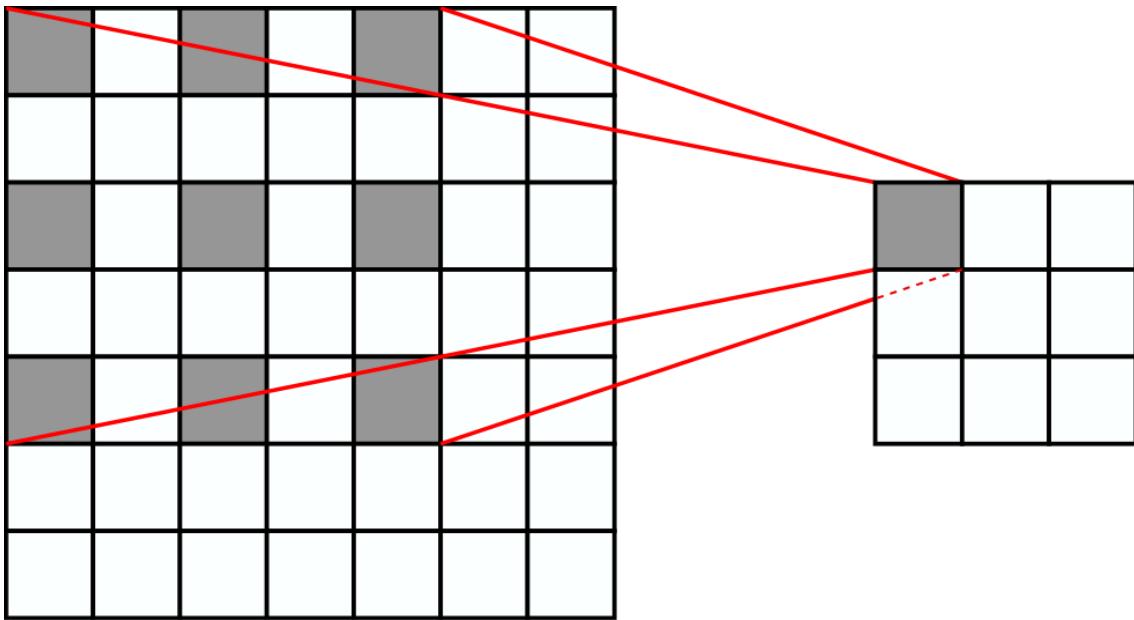


Abbildung 3.1: Prinzip von Atrous Convolution. Das Vorgehen bei der Faltung ist beispielhaft bei einer Erweiterungsrate von zwei dargestellt. Der Wert an der eingefärbten Stelle der Ausgabematrix (rechts) hängt von denen an den eingefärbten Stellen der Eingabematrix (links) ab, statt, wie bei einer herkömmlichen zweidimensionalen Faltung, von benachbarten. Die roten Strahlen markieren die Ecken des Sichtfelds.

übergreifenden 1×1 Faltung unterzogen wird, um die endgültigen Wahrscheinlichkeiten zu berechnen. Das Prinzip ist in Abbildung 1.4 dargestellt. Durch dieses Vorgehen soll Größeninvarianz erreicht werden.

3.4 Conditional Random Fields

Ein Conditional Random Field (CRF) ist ein Modell, das eine Datensequenz erhält und eine Sequenz gleicher Länge und Art ausgibt. CRFs werden zur Segmentierung und zum Labeln genutzt. Das Modell setzt, wie in [CrfIntro] und [McCallum:2002:EIF:2100584.2100633] beschrieben, überwachtes Lernen ein, um Parameter für eine Distribution zu bestimmen, die die Verteilung $d(X|Y)$ beschreibt, wobei X und Y Zufallsvariablen sind. X beschreibt die beobachtete Eingabesequenz, Y die zu bestimmende Ausgabesequenz.

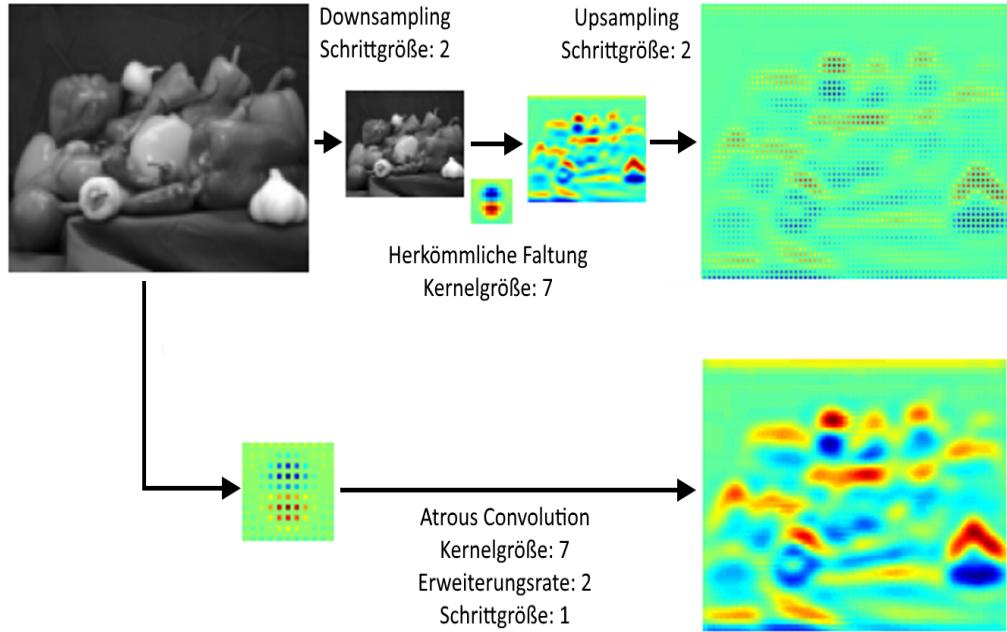


Abbildung 3.2: Beispielhaft dargestellte Vorteile von Atrous Convolution nach [DeepLab2]. In der oberen Reihe ist das übliche Vorgehen bei CNNs dargestellt, in dem durch Down- und Upsampling die Effizienz verbessert wird. Der Vorgang entspricht einer Faltung mit erhöhter Schrittgröße. Unten dargestellt ist die Anwendung von Atrous Convolution mit einer Schrittgröße von eins und Erweiterungsrate zwei. Ein Vergleich der Ergebnisse der beiden Algorithmen zeigt, dass die Ausgabe von Atrous Convolution eine höhere Auflösung aufweist.

Man betrachte eine Familie von Distributionen $p(z)$ und eine Menge von Feature-Funktionen ϕ_1, \dots, ϕ_N , die alle Daten ausdrücken, die in dem Modell berücksichtigt werden sollen. Es soll nun eine Sammlung von Parametern ω gefunden werden, sodass durch $p(y|x, \omega)$ die reale Verteilung $d(y|x)$ möglichst gut angenähert wird. p soll dabei so gewählt werden, dass dessen Entropie $H(p)$ maximal ist. Modelliert man, was in der Regel der Fall ist, p als Markov-Kette erster Ordnung, heißt, man nimmt an, dass vergangene Zustände den Ausgangszustand nicht beeinflussen, hat die Distribution mit maximaler Entropie die Form:

$$p(z) = \frac{1}{Z} \exp\left(\sum_i \omega_i \phi_i(z)\right) \quad (3.1)$$

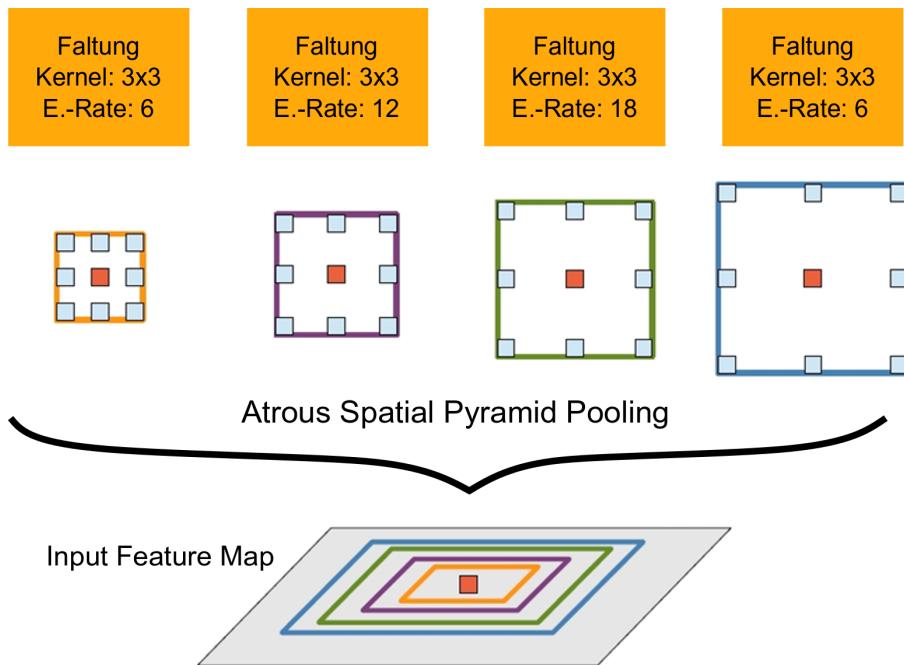


Abbildung 3.3: Prinzip von Atrous Spatial Pyramid Pooling wie in [DeepLab2]. Die Eingabe wird mit verschiedenen Filtern mit unterschiedlicher Erweiterungsrate gefaltet, was dazu führt, dass die Stellen in den Ausgabematrizen verschiedene große Sichtfelder aufweisen. Die so entstandene Ergebnisse werden zu einer Feature Map kombiniert.

mit einem festen Normalisierungsfaktor Z .

Idealerweise wird für eine Eingabesequenz x eine Ausgabesequenz y so gewählt, dass $p(y|x)$ nach dem berechneten Random Field $p(X|Y)$ maximal wird. Eine naheliegende Möglichkeit ist das „Ausprobieren“ aller möglichen y . Allerdings wäre das für lange Sequenzen zu aufwändig, weshalb in praktischen Anwendungen andere Optimierungsalgorithmen eingesetzt werden. Modelliert man, wie zuvor, p als Markov-Kette, ist eine effiziente Berechnung auf Grundlage des Viterbi-Algorithmus [Ryan:1993:VA:901051] möglich.

Der Lernprozess für CRFs beruht auf Gradient Descend. Da die Wahrscheinlichkeitsfunktion bei eindeutigen Trainingsdaten wegen der Eigenschaften der Exponentialfunktion konvex ist, ist ein lokales Minimum dabei garantiert ein globales. Das Berechnen des Gradienten ist allerdings nicht-trivial und ineffizient, weshalb auf iterative und stochastische Lernalgorithmen, wie das Newtonverfahren zurückgegriffen wird. Wird für die Berechnung eines Wertes der Ausgabe alle Einträge der Eingabe betrachtet, spricht man von einem Fully Connected Conditional Random Field. Für weitere Informationen und eine präzise Definition siehe [CRF].

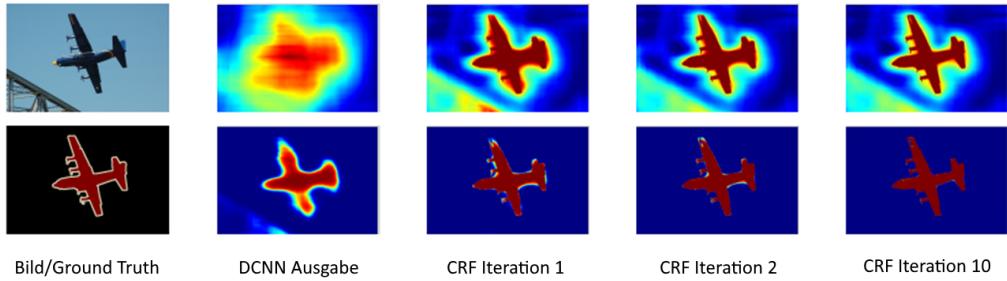


Abbildung 3.4: CRFs können eingesetzt werden, um die Ergebnisse von CNNs zu raffinieren, wie in [DeepLab1] gezeigt. In der oberen Reihe werden die Score Maps, die Ergebnisse vor Anwendung einer Soft-Max-Funktion, gezeigt, in der unteren die Ausgabe der Soft-Max-Funktion.

3.5 Residual Networks

Ein Residual Neural Networks (ResNet) ist ein neuronales Netz, das das in [DBLP:journals/corr/Han15] vorgestellte Residual Learning implementiert. Dabei werden, wie in ?? dargestellt, „Abkürzungen“ in das Netz eingebaut, über die die Ausgabewerte einer Schicht eine oder mehrere nachfolgende Schichten überspringen und eine tiefere Schicht unverändert erreichen und mit den Ergebnissen der übersprungenen Schichten addiert werden. Mit ResNets wird ein Problem von Deep Neural Networks gelöst, bei dem der Trainingsfehler durch Hinzufügen zusätzlicher Verarbeitungsschichten vergrößert wird. Wenn nun eine Anzahl von Schichten größer als eins, nicht unbedingt das gesamte Netz, eine beliebige stetige Funktion $H(x)$ approximieren kann, so kann angenommen werden, dass dies auch für eine Funktion $H(x) - x$ gilt. Anstatt zu erwarten, dass $H(x)$ gelernt wird, wird beim Residual Learning explizit eine Funktion $F(x) := H(x) - x$ approximiert. Die ursprüngliche Verarbeitungsfunktion wird dafür zu $F(x) + x$ abgewandelt. Die Überlegung dabei ist, dass weitere Schichten die Resultate nicht verschlechtern können, wenn die Eingabe vorheriger Schichten noch unverändert vorhanden ist. Diese These wird durch die Ergebnisse der in [DBLP:journals/corr/HeZRS15] dargelegten Experimenten gestützt. Residual Learning ist heute eine weit verbreitete Technologie beim Einsatz von Deep Learning.

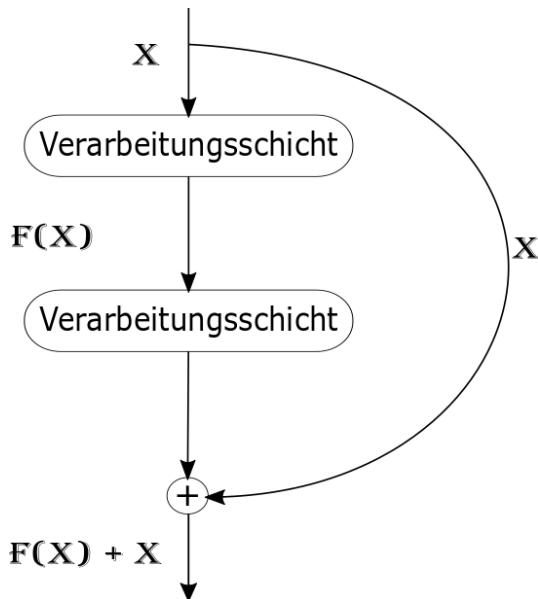


Abbildung 3.5: Prinzip von Residual Learning nach [DBLP:journals/corr/HeZRS15]. Über spezielle Abkürzungs-Schichten werden Daten unverändert in weiter darunter liegende Schichten geleitet und auf deren Ausgabe addiert.

3.6 Kamerakalibrierung

Eine Kamera ermöglicht es, ein dreidimensionales Objekt auf eine zweidimensionale Ebene zu projizieren. Diese Projektion kann, wie in [Hartley] beschrieben, mit dem in Abbildung ?? dargestellten Modell angenähert werden, in dem von Punkten im dreidimensionalen Raum ein Strahl durch einen bestimmten Fixpunkt, das Projektionszentrum, verläuft und dabei eine vorgegebene Ebene, die Bildebene, schneidet. Der Schnittpunkt dieses Strahls mit der Bildebene entspricht dem projizierten Punkt auf dem entstehenden Bild.

Offensichtlich werden alle Punkte im dreidimensionalen Raum, die auf einem Strahl durch das Projektionszentrum liegen, auf denselben Punkt in der Bildebene projiziert. Das Bild lässt sich also praktisch als eine Menge von Strahlen auffassen.

Ist eine Kamera kalibriert, ist es möglich, aus zwei Punkten auf einem damit aufgenommenen Bild den Winkel zwischen den beiden Strahlen zu bestimmen, durch die sie entstanden sind. Analog dazu kann beispielsweise aus dem Bild auf die Größe einer fotografierten Fläche geschlossen werden oder bestimmt werden, ob eine Ellipse auf dem Bild die Projektion eines Kreises ist. Um solche Berechnungen anzustellen sind zusätzliche Informationen notwendig, da bei der Kameraprojektion Informationen

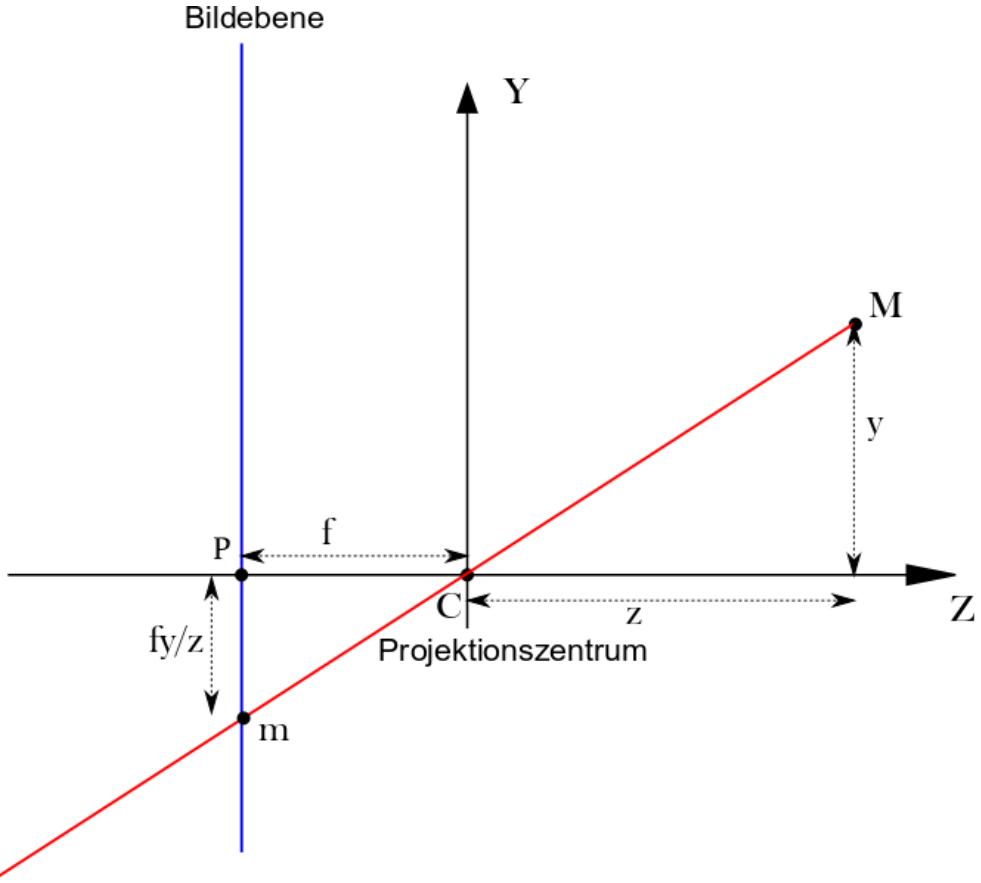


Abbildung 3.6: Prinzip einer Lochkamera zur Projektion vom dreidimensionalen in den zweidimensionalen Raum nach [Fusiello06elementsof]. Von einem realen Punkt M verläuft ein Strahl durch das Projektionszentrum C und schneidet die Bildebene im Bildpunkt m . Das Koordinatensystem ist so gewählt, dass sich C im Ursprung befindet und die **Y**-Achse parallel zur Bildebene verläuft. Die **Z**-Achse schneidet die Bildebene im Bildzentrum P . Hat M die Entfernung z auf der **Z**-Achse und y auf der **Y**-Achse und der Abstand zwischen C und der Bildebene ist f , so beträgt die Entfernung von m zur **Z**-Achse $\frac{fy}{z}$.

über Entfernungen, Längen, Winkel, Verhältnisse und dementsprechend Formen nicht erhalten bleiben.

Dass eine Kamera kalibriert ist, bedeutet im Praktischen Sinn, dass eine Matrix P berechenbar ist, für die gilt:

$$zm = PM. \quad (3.2)$$

M bezeichnet dabei die homogenen Koordinaten eines Punktes im dreidimensionalen Raum und m diejenigen von dessen Projektion auf der Bildebene, z ist ein reeller Faktor ungleich Null. Homogene Koordinaten unterscheiden sich von Euklidischen durch einen Zusätzlichen Parameter, der oft mit T bezeichnet wird. Eine Koordinate im zweidimensionalen Raum hat also die Form: $(X, Y, Z, T)^T$. Es gilt dabei, dass der Punkt $(x, y, 1)$ in homogenen Koordinaten äquivalent ist zum Punkt (x, y) in euklidischen Koordinaten. Es gilt also:

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \hat{=} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \quad (3.3)$$

Für homogene Koordinaten mit $T \neq 0$ verhält es sich für alle $k \neq 0$ so, dass:

$$\begin{pmatrix} x \\ y \\ \omega \end{pmatrix} = \begin{pmatrix} kx \\ ky \\ k\omega \end{pmatrix} \quad (3.4)$$

Eine Umrechnung in euklidische Koordinaten erfolgt folglich durch:

$$\begin{pmatrix} x \\ y \\ \omega \end{pmatrix} \hat{=} \begin{pmatrix} \frac{x}{\omega} \\ \frac{y}{\omega} \\ 1 \end{pmatrix} \quad (3.5)$$

Ein Punkt, in dem $T = 0$ gilt, bezeichnet man als „Punkt im Unendlichen“. Was hier beispielhaft für zwei Dimensionen dargestellt ist, kann offensichtlich für beliebig viele Dimensionen erweitert werden. Für weitere Informationen über homogene Koordinaten siehe [Hartley].

Kenntnis über die so genannte Projektions- oder Kameramatrix P ermöglicht also die Berechnung der zum Projektionszentrum relativen Koordinaten des projizierten Punktes aus denen des aufgenommenen Punktes im dreidimensionalen Raum. Außerdem kann damit berechnet werden, wo ein mit einer kalibrierten Kamera aufgenommener Punkt im Bild einer anderen erscheint. Zum Ermitteln der Projektionsmatrix sind, wie in [Fusiello06elements] beschrieben, folgende Informationen notwendig:

- Die Entfernung f zwischen dem Projektionszentrum und der Bildebene.
- Die Koordinaten des Bildzentrums $P = (p_x, p_y)$.
- Höhe s_x und Breite s_y der Pixel.
- Der Scherungswinkel Θ zwischen den Achsen, der für Gewöhnlich $\frac{\pi}{2}$ beträgt.

Diese Informationen lassen sich in der so genannten Kalibrierungsmatrix K folgendermaßen Zusammenfassen:

$$K = \begin{bmatrix} \frac{f}{s_x} & \frac{f}{s_x} \cot \Theta & p_x \\ 0 & \frac{f}{s_y} & p_y \\ 0 & 0 & 1 \end{bmatrix} \quad (3.6)$$

Ist die Kalibrierungsmatrix bekannt, muss noch die Position der Kamera durch die Rotationsmatrix R und den Translationsvektor t ausgedrückt werden. Dann kann die Projektionsmatrix berechnet werden mittels:

$$P = K [R|t] \quad (3.7)$$

4 Arbeitsmethodik und Entwicklung

4.1 Algorithmus

Es gibt mehrere Ansätze zum Segmentieren von Punktwellen mit Neuronalen Netzen, die unterschiedliche Leistungen bezüglich Laufzeit und Qualität erzielen. In dieser Arbeit wird ein möglichst einfacher und zeiteffizienter Algorithmus angewandt. Dabei wird, wie in Abbildung ?? dargestellt, zuerst ein Eingabebild mit Hilfe neuronaler Netze semantisch segmentiert. Dazu wird an dieser Stelle DeepLabV3+ [[DBLP:journals/corr/abs-1802-02611](#)] benutzt, da es mit seinen Ergebnissen auf populären Datensätzen (79.7% auf Pascal VOC 2012 [[Everingham10](#)] und 70.4% auf Cityscapes [[Cityscapes](#)] in mIOU-Metrik) laut [[DeepLab2](#)], als State-of-the-Art angesehen wird und gleichzeitig eine große Fülle an Informationen und öffentlichen Implementierungen zur Verfügung stehen.

Die durch die Segmentierung ermittelten Labels werden anschließend auf eine Punktwolke projiziert, die dieselbe Szene wie das segmentierte Bild darstellt. Dazu werden ein Bild und eine Punktwolke der zu segmentierenden Szene, sowie eine Projektionsmatrix benötigt. die verwendete Kamera muss also kalibriert sein. Offensichtlich ist es damit nur möglich, den Bereich der Punktwolke zu segmentieren, der im Bild zu sehen ist. Außerdem werden auf diese Weise keine in der Punktwolke vorhandenen Tiefen-Information ausgenutzt. Stattdessen können Farb-Informationen berücksichtigt werden.

4.2 DeepLab

DeepLab ist ein von Google entwickeltes, 2015 in [[DeepLab1](#)] vorgestelltes Modell für semantische Segmentierung. Bei der in [[DeepLab2](#)] vorgestellten Methode wird ein Deep Convolutional Neural Network (DCNN) zum Erzeugen einer Score Map

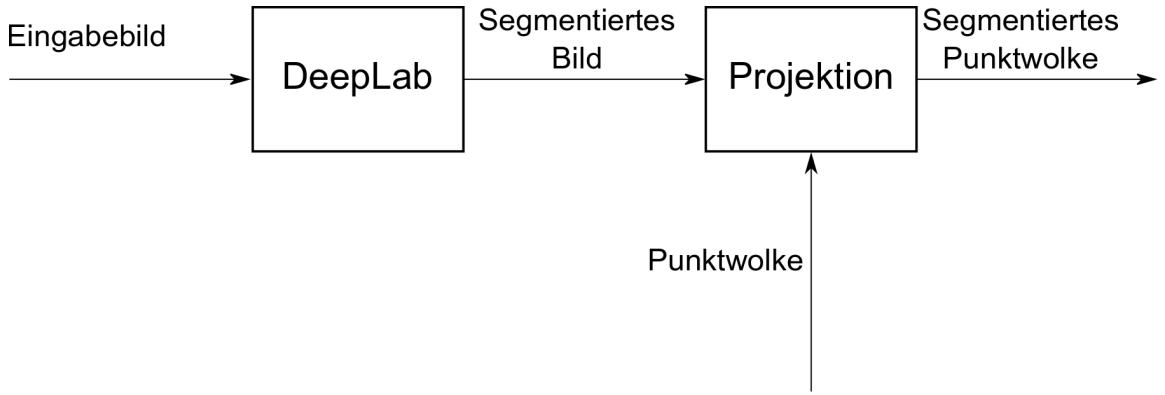


Abbildung 4.1: Schematische Arbeitsweise des Systems. Ein Eingabebild wird zunächst mit DeepLab segmentiert. Die so entstandenen Labels werden unter Verwendung einer Projektionsmatrix auf eine zum Eingabebild gehörende Punktwolke projiziert.

benutzt, die anschließend mit einem Conditional Random Field (CRF) zur endgültigen Ausgabe weiterverarbeitet wird. Das Verfahren wird in Abbildung 1.1 grob dargestellt.

4.2.1 Anpassungen für Semantische Segmentierung

Klassische DCNNs haben Eigenschaften, die sie für die Verwendung zur Bildsegmentierung nicht ideal machen.

- Der Einsatz von Downsampling führt zu verringriger Auflösung, die bei Klassierungsaufgaben nicht ins Gewicht fällt, für die Segmentierung aber essentiell ist.
- Neuronale Netze sind in der Regel gut geeignet, um Objekte unterschiedlicher Größe zu erkennen, wenn solche in der Lernphase präsentiert werden. Die Eigenschaften der Faltung, insbesondere dem begrenzten Sichtbereich beim Berechnen eines einzelnen Pixels ist allerdings für diese Problematik ungünstig.
- Der wiederholte Einsatz von Convolutional Layers führen zu einem Verlust an Ortsinformation. Infolgedessen produzieren DCNNs bei Segmentierungsaufgaben verschwommene, oft verrauschte Ergebnisse ohne klare Kanten.

Um diese Probleme zu lösen, erhält das von DeepLab verwendete DCNN einige Anpassungen. Zunächst werden alle Fully Connected Layers durch Convolutional Layers ersetzt, um ein Fully Convolutional Network zu bilden. Noch dazu wird anstatt

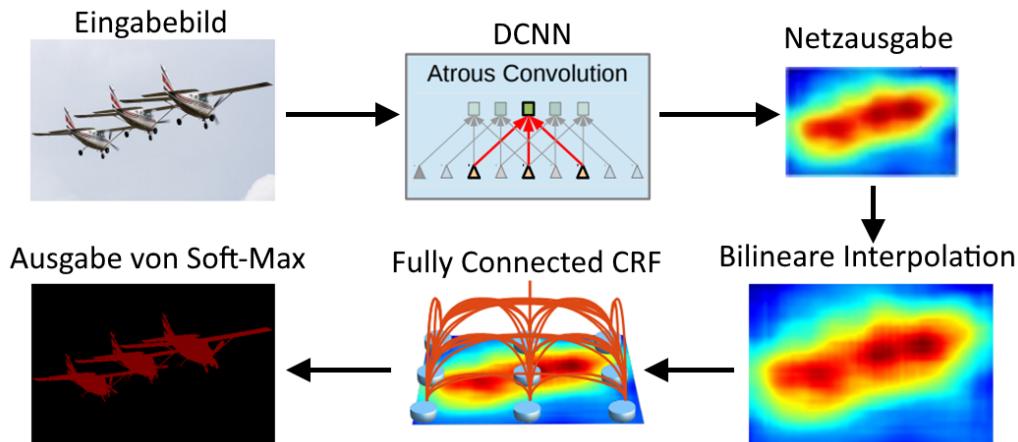


Abbildung 4.2: Arbeitsfluss von DeepLab nach [DeepLab2]. Das Eingabebild wird von einem DCNN segmentiert. Das Ergebnis wird durch Bilineare Interpolation auf die Größe der Eingabe vergrößert und in einem Fully Connected CRF raffiniert.

von Pooling Layers in den unteren Schichten Atrous Convolution eingesetzt, womit die Auflösung der Ausgabe erhöht wird. In den höheren Schichten werden auch hier Pooling Layers eingesetzt, um Speicherbedarf und Rechenzeit zu verbessern. Um Größeninvarianz zu erreichen wird bei den unteren Schichten Atrous Spatial Pyramid Pooling verwendet und um die Ergebnisse, vor allem an den Kanten von Objekten zu verbessern und Rauschen zu reduzieren, wird die Ausgabe des DCNN mit einem Fully Connected CRF weiterverarbeitet.

4.3 Anwendung von DeepLab

Als Basis für die Integration von DeepLab dient eine öffentlich zugänglichen Implementierung in Python mit dem Pytorch-Framework.

Das Projekt implementiert die zwei Netzwerke Xception65 und MobileNetV2. Es wird die Möglichkeit angeboten, vor-trainierte Encoder zu verwenden, die in dieser Arbeit aber nicht genutzt wird, da derzeit Kompatibilitätsprobleme mit den bereitgestellten vor-trainierten MobileNetV2-Modellen auftreten. Neben den bereits beschriebenen Technologien werden die klassischeren Konzepte der Neuroinformatik Dropout, L2-Regularisierung und Momentum verwendet.

Bezüglich der Entwicklung ist das erste Ziel die Verwendung dieses Projekts mit einem vor-trainierten Model zur Evaluierung frei gewählter Bilder. Ein Model bezeichnet im Rahmen dieser Arbeit eine Datei, in der vom Netz gelernte Parameter gespeichert sind

und bei Bedarf geladen werden können. Die ursprüngliche Implementierung erwartet Test- und Trainingsdaten im Format von entweder Cityscapes oder Pascal. Der erste Schritt ist folglich die Erstellung eines Python-Skripts zur Evaluierung beliebiger Bilder. Dieses Skript soll Ausgaben im PNG-Format erzeugen, die das Originalbild, die Auswertung des Netzes, eine Legende, ein RGB-Histogramm und die erforderliche Rechendauer zeigen. Zur Erzeugung der Plots wird das Modul Matplotlib verwendet. Für den Fall, dass die Eingabebilder einen schwarzen Rand aufweisen, wird dieser Rand auf die segmentierten Bilder übertragen, wozu das Bildverarbeitungs-Framework OpenCV verwendet wird. Die Legende wird automatisch erstellt und richtet sich nach der Klasse mit dem höchsten Label, die in dem Eingabebild erkannt wurde. Sie zeigt außerdem den IoU-Wert jeder Klasse. Das RGB-Histogramm wird mit einer Funktion von OpenCV automatisch erstellt. Ein Beispiel ist in Abbildung ?? dargestellt.

In einem nächsten Schritt soll die Möglichkeit geschaffen werden, ein eigenes Model mit eigenen Trainingsdaten zu trainieren. Das bereits vorhandene Trainings-Skript kann dazu verwendet werden. Die Verwendung von Nebenläufigkeit beim Laden der Trainingsdaten führt allerdings unter Windows dazu, dass das Training nach einer Epoche abgebrochen wird. Auch hier wird erwartet, dass die Daten im Format von Cityscapes oder Pascal sind. Da hier vor allem der Cityscapes-Datensatz zum Trainieren verwendet wird und dieser einfach zu erweitern ist, wird das als sinnvoll eingeschätzt und beibehalten. Zum Beginnen eines Trainings muss eine Konfigurationsdatei im YAML-Format übergeben werden. Darin können folgende Trainingsparameter festgelegt werden:

- Encoder-Type
- Decoder-Typ
- Format des Datensatzes
- Zielgröße der Trainingsbilder
- Anzahl Trainingsepochen
- Batch-Größe
- Verwendung von FP16
- Fehler-Typ
- zu ignorierender Index (255 in Cityscapes)
- Optimierer

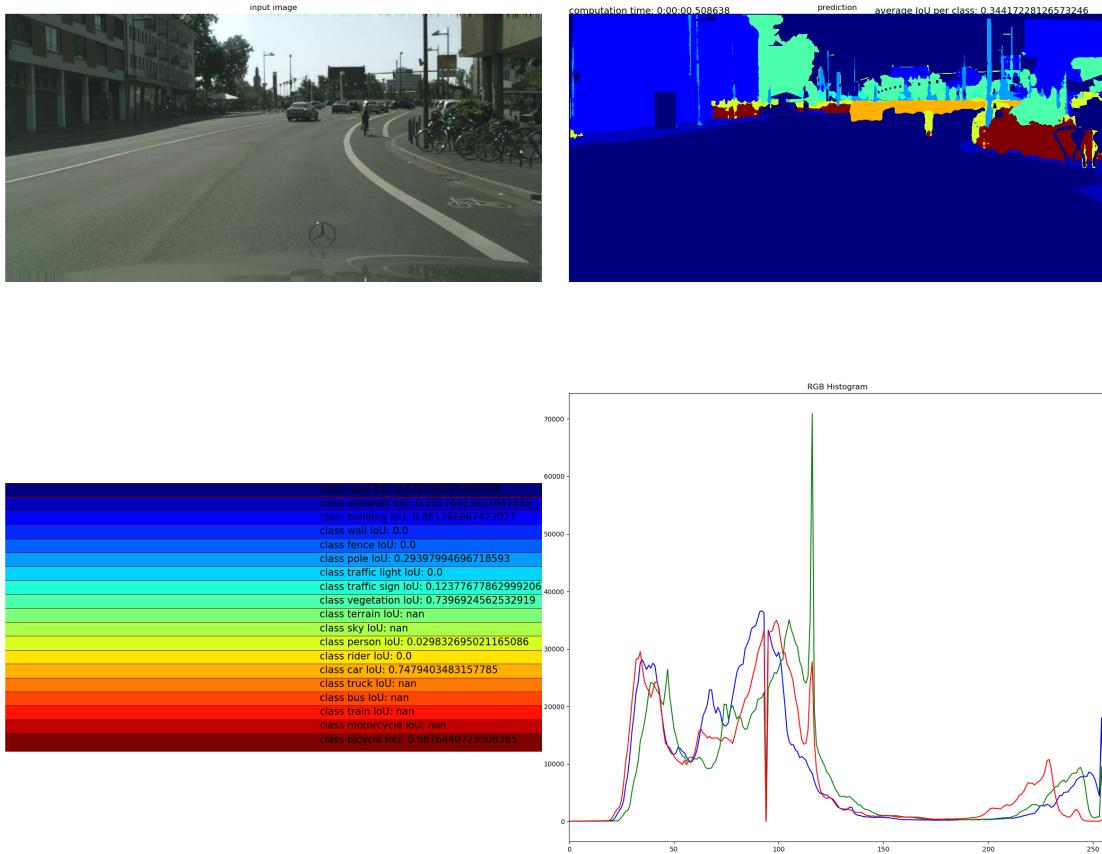


Abbildung 4.3: Beispiel für die Ausgabe des Evaluierungs-Skripts. Zu sehen ist das Eingabebild (oben links), das segmentierte Bild (oben rechts), eine Legende, die den IoU für jede Klasse enthält (unten links) und ein RGB-Farbhistogramm (unten rechts). Die Rechenzeit des Netzes und der durchschnittliche IoU werden über dem segmentierten Bild angezeigt.

- Grundlernrate

Außerdem kann der Pfad zu einem vor-trainierten Model angegeben werden und bestimmt werden, ob ein bereits existierendes Model weiter trainiert werden soll, was zum Nach-Training benutzt werden kann.

Als letztes muss noch ein Skript erstellt werden, das ein Model auf dem Cityscapes-Datensatz testet und in IoU-Metrik bewertet. Eine Funktion zur Berechnung des IoU ist bereits vorhanden, muss aber noch in einer für diese Arbeit sinnvolle Weise aufgerufen werden. Das dazu geschriebene Skript erwartet Evaluierungsdaten im

Cityscapes-Format, da die Experimente auf diesem Datensatz durchgeführt werden. Es wertet alle Bilder dieses Datensatzes mit dem zu testenden Model aus und berechnet den durchschnittlichen IoU für jede von dem Netz erkennbare Klassen, sowie die durchschnittliche Rechendauer pro Eingabebild. Zu beachten ist dabei, dass NaN-Werte bei der Berechnung speziell behandelt werden müssen.

Zusätzlich soll noch die Möglichkeit geschaffen werden, eigene Daten zu annotieren. Dazu wird der von der University of Oxford bereitgestellte VGG Image Annotator (VIA) genutzt. Dieser bietet die Möglichkeit, Polygone in ein Bild einzutragen und diese als CSS-Dateien zu exportieren. Ein Python-Skript, das OpenCV verwendet verarbeitet diese Daten dann zu Cityscapes-konformen PNG-Dateien, die zum Training oder zur Validierung verwendet werden können. Im Rahmen dieser Arbeit wird das allerdings nicht eingesetzt.

4.4 Backbones

Wie bereits erwähnt, wird der Begriff Backbone hier als Synonym für Feature Extractor verwendet und bezeichnet ein Netz, das ein Bild als Eingabe erhält und eine Feature Map erzeugt. Die hier verwendete Implementierung von DeepLab umfasst zwei Backbones: Xception65 und MobileNetV2.

4.4.1 Xception

Das in [Xception] präsentierte Xception-Netzwerk ist ein einfaches aber leistungsfähiges DCNN, das auf der Idee von Depthwise Separable Convolution basiert. Es wird also angenommen, dass Korrelationen, die mehrere Kanäle umfassen von räumlichen Korrelationen entkoppelt werden können. Die einzelnen Module des Netzes verarbeiten zunächst alle Kanäle separat und führen dann eine einfache Faltung über alle Dimensionen durch. Dadurch werden Laufzeit- und Speichereffizienz verbessert. Das Prinzip ist in Abbildung ?? dargestellt.

Das vorgestellte Netz hat insgesamt 36 Faltungsschichten, die in 14 Module gegliedert sind, die als ResNet miteinander verbunden sind. In jeder Schicht werden nach dem oben erklärten Prinzip mehrere Faltungen mit 3x3 Filtern parallel auf allen Kanälen durchgeführt. Der so entstandene Tensor wird anschließend mit einem 1x1 Filter gefaltet, der alle Dimensionen umfasst.

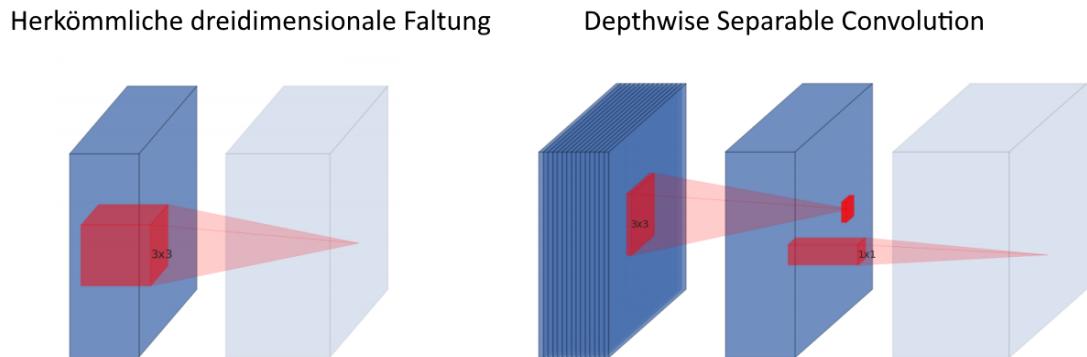


Abbildung 4.4: Schematische Darstellung des Prinzip von Depthwise Separable Convolution zur Tensorverarbeitung nach [MobileNetV2]. Statt einer Faltung mit einem 3x3-Kernel über alle Dimensionen, wird für jede Dimension eine Faltung mit einem 3x3-Kernel durchgeführt, gefolgt von einer Faltung mit einem 1x1-Kernel über alle Dimensionen.

4.4.2 MobileNetV2

Bei MobileNetV2 [MobileNetV2] handelt es sich um ein leichtgewichtiges DCNN für die Implementierung auf mobilen Geräten. Da Laufzeit und Ressourcenlastigkeit für das Ziel dieser Arbeit von kritischer Bedeutung sind, wird hier vorrangig mit diesem Backbone gearbeitet. Wie Xception verwendet es das Prinzip von Depthwise Separable Convolution und Residual Connections. Für den Entwurf des Netzes wird zusätzlich die Annahme gemacht, dass die entscheidenden Eigenschaften eines Eingabetensors in einem Subraum mit weniger Dimensionen zusammengefasst werden können.

Begründet auf diesen Konzepten ist das grundsätzliche Architekturelement von MobileNet der so genannte „Bottleneck Residual Block“. Ein- und Ausgabetensoren dieser Verarbeitungsschichten haben eine niedrigere Dimension als Tensoren dazwischen. Innerhalb des Blocks wird zuerst der Eingabetensor zu einem mit mehr Dimensionen erweitert. Der so entstandene Tensor wird dann nach dem Prinzip von Depthwise Separable Convolution zuerst Kanalweise, dann Kanalübergreifend mittels Faltungsoperationen und ReLU6 weiterverarbeitet, wobei die Dimension wieder reduziert wird. Das Ergebnis davon wird nach dem ResNet-Prinzip mit dem Eingabetensor verrechnet. Der Vorteil dieses Verfahrens liegt in einem geringeren Speicherbedarf. Insgesamt besteht das Netz aus einem Convolutional Layer mit 32 Filtern am Anfang, gefolgt von 19 Residual Bottleneck Layer und einigen weiteren Schichten am Ende, die von der zu erfüllenden Aufgabe abhängig sind.

4.5 Segmentierung von Punktwolken

Wie in Abschnitt ?? beschrieben, ist es durch Kenntnis einer Projektionsmatrix möglich zu berechnen, wo ein Punkt im dreidimensionalen Raum auf einem aufgenommenen Bild erscheint. Ist eine solche Matrix für die jeweilige Kamera bekannt, wird diese als kalibriert bezeichnet.

Zur Projektion der auf dem Bild erkannten Klassen auf die Punktfolge iteriert ein Algorithmus durch alle Punkte und berechnet durch Anwendung von Gleichung ?? einen Vektor bestehend aus den homogenen Koordinaten des projizierten Punktes. Die Komponenten dieses Vektors werden im Folgenden, analog zur Bezeichnung in Gleichung ??, mit x , y und ω bezeichnet. Für alle so berechneten Vektoren, deren ω -Komponente positiv ist, andernfalls befände sich der entsprechende Punkt in der Punktfolge hinter der Bildebene, werden deren euklidische Koordinaten mittels Gleichung ?? ermittelt und überprüft, ob sie sich im Bild befinden. Ist dies der Fall, wird dem entsprechenden Punkt in der Punktfolge das Label zugewiesen, das an den berechneten Koordinaten im Bild bei der Segmentierung erkannt wurde. Befindet sich der berechnete Punkt auf der Bildebene nicht innerhalb des Bildes, bedeutet das, dass sich der jeweilige Punkt im dreidimensionalen Raum nicht im Sichtfeld der Kamera befand und folglich nicht auf dem Bild erscheint. Offensichtlich kann der in dieser Arbeit vorgestellte Algorithmus keine Aussagen über solche Punkte machen.

Der KITTI-Datensatz [KITTI] bietet sowohl Aufnahmen von kalibrierten Farbkameras, als auch Laserscans in Form von Punktfolgen im Velodyne-Format und eine, wenn auch verhältnismäßig geringe, Menge an fein-annotierten Bildern für semantische Segmentierung. Das macht ihn zu einer logischen Wahl für die Generierung von Beispielergebnissen im Rahmen dieser Arbeit.

Für die Implementierung bietet sich das für KITTI entwickelte Python-Modul Pykitti an, das Funktionen zur Verwaltung der Bild- und Velodyne-Daten und zum Umrechnen der Koordinaten anhand der Projektionsmatrizen anbietet. Für die Visualisierung der Ergebnisse wird eine Python-Integration von Mayavi verwendet. Abbildung ?? zeigt ein Beispiel.

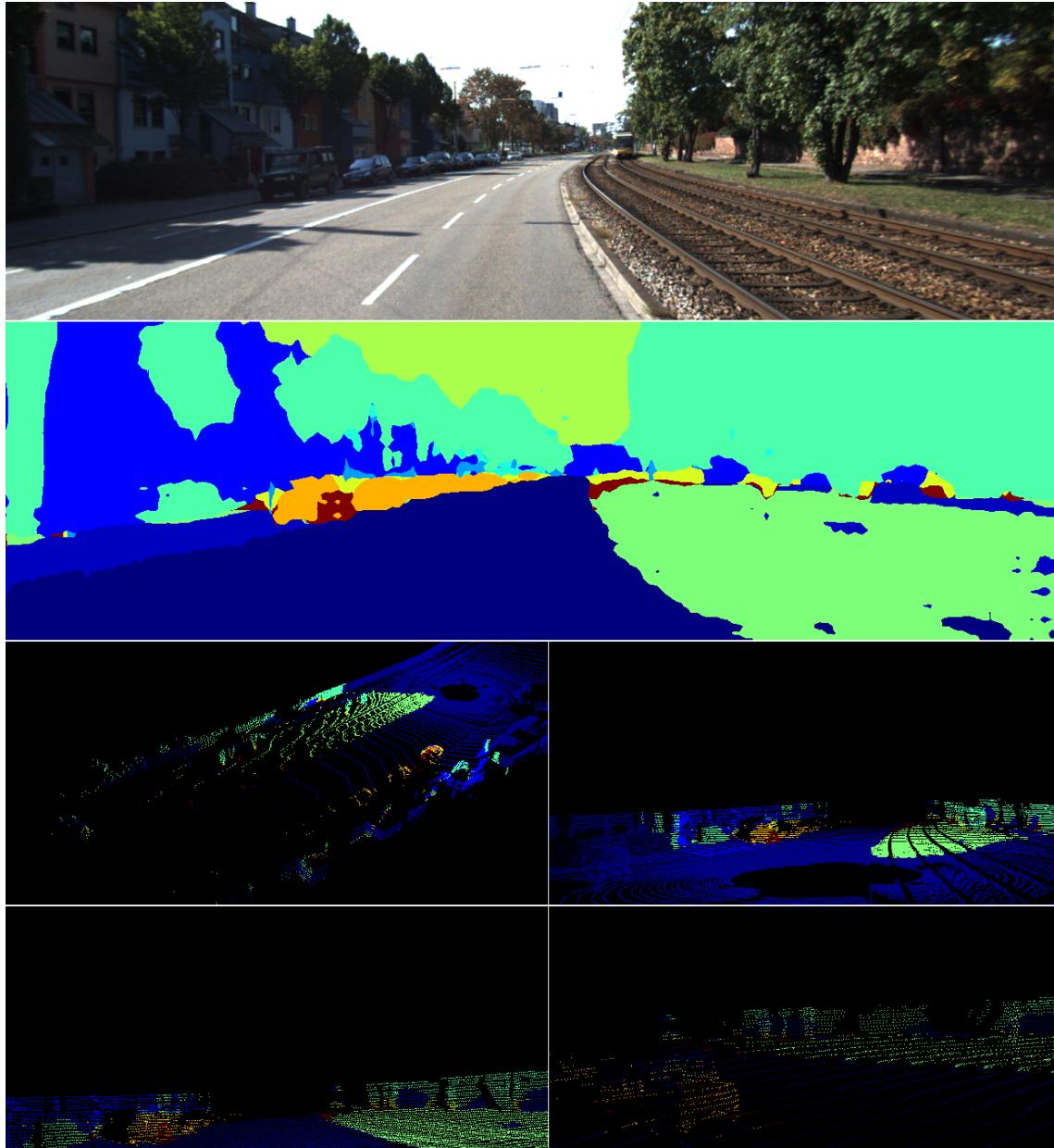


Abbildung 4.5: Beispiel für segmentierte Punktwolke aus dem KITTI-Datensatz. Zu sehen ist von oben nach unten: Originalbild, Ergebnis von DeepLab, segmentierte Punktwolke aus verschiedenen Perspektiven.

5 Datensätze

Methoden, die auf überwachtes Lernen zurückgreifen, erfordern in der Regel eine große Menge an Trainingsdaten, um verwendbare Resultate zu erzielen. Das Erstellen von Labeln für semantische Segmentierung ist besonders aufwändig, da jedem Pixel des Bildes eine Klasse zugeordnet werden muss. Deshalb werden in dieser Arbeit die öffentlich zugänglichen Datensätze Cityscapes und KITTI genutzt. In dem Kapitel soll näher auf verfügbare Datensätze eingegangen und erläutert werden, weshalb diese beiden ausgewählt wurden.

5.1 Cityscapes

Cityscapes ist ein öffentlich zugänglicher Datensatz ausgelegt für Bildsegmentierung zum autonomen Fahren. Er bietet 5000 fein und 20000 grob auf Pixel-Ebene annotierte Bilder für semantische oder Instanzsegmentierung. Der Satz an fein annotierten Aufnahmen, der in den Experimenten verwendet wird, ist unterteilt in einen Trainingssatz aus 2975 Bildern, einem Evaluierungssatz von 500 Bildern und einem Testsatz aus 1525 Bildern. Aufgenommen ist der Datensatz von einem Auto aus in 50 größtenteils deutschen Städten, jeweils am Tag bei sonnigem oder bewölktem Wetter um Frühling, Sommer und Herbst. Die Bilder zeigen ausschließlich Szenen, die sich auf vielbefahrenen Straßen abspielen.

Die Daten sind aufgenommen mit einer Stereo-Kamera, die hinter der Windschutzscheibe des Fahrzeugs angebracht ist, bei einer Frame-Rate von 17Hz. Die Bilder des Datensatzes sind kalibriert, Bayer-gefiltert und rektifiziert. Abbildung ?? zeigt Beispiele. Für weitere Informationen siehe [[Cityscapes](#)].

Die große Anzahl an qualitativ hochwertigen Bildern und Annotationen, sowie deren Verfügbarkeit machen Cityscapes zu einer logischen Wahl für diese Arbeit. 2975 Bilder scheinen eine vergleichbar kleine Datenmenge zu sein, doch die Information, die in einem Bild enthalten ist, ist ungleich größer als beispielsweise in für Klassifizierung annotierten Bildern.

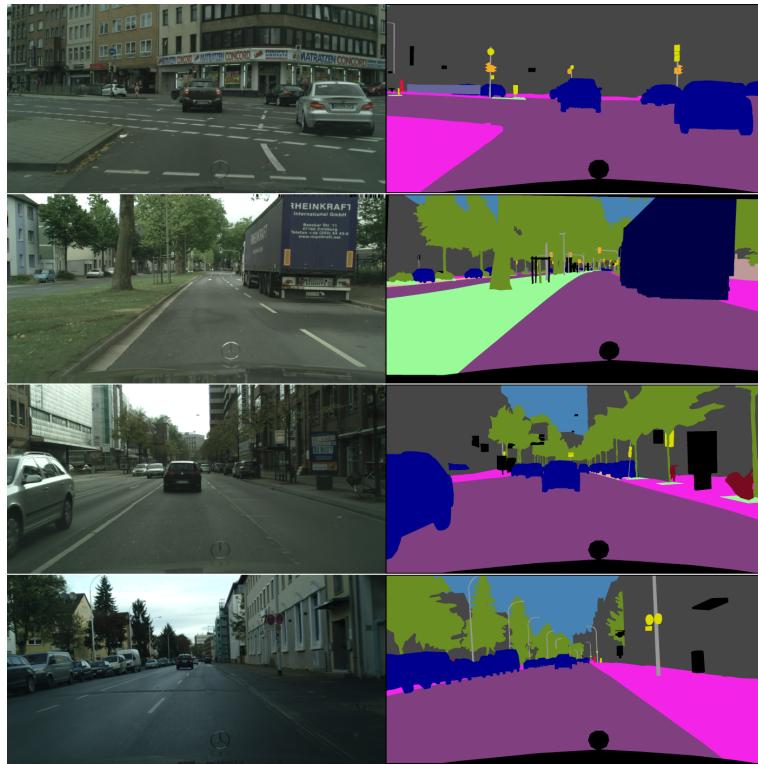


Abbildung 5.1: Hier zu sehen sind beispielhaft fein annotierte Trainingsdaten für semantische Segmentierung aus dem Cityscapes-Datensatz. Wie man sieht, werden die Bilder (links) von einer hinter der Windschutzscheibe platzierten Kamera aufgenommen. Die rechte Spalte zeigt die eingefärbte Ground Truth für semantische Segmentierung. Die schwarz markierten Bildflächen gehören keiner im Datensatz gelabelten Klasse an und werden während des Trainings ignoriert.

5.2 KITTI

Das in [KITTI] beschriebene KITTI ist ein Datensatz für Forschung in den Bereichen mobile Robotik und autonomes Fahren. Es werden darin Kamerabilder, Laserscans, GPS- und IMU-Daten zur Verfügung gestellt. Die Kamerabilder werden von zwei Stereo-Kamera-Rigs aufgenommen, eines für Farbaufnahmen, eines für Graustufenbilder und liegen sowohl als Rohdaten als auch rektifiziert vor. Die Laserscan-Daten sind im Velodyne LiDAR-Format gespeichert. Die Kalibrierungs-Matrizen sind im Rohdatensatz ebenfalls angegeben.

Der Datensatz umfasst insgesamt 6 Stunden an Aufnahmen mit zwischen 10Hz und 100Hz aus Karlsruhe. KITTI bietet außerdem 200 für Segmentierung annotierte Bilder.

Die Messgeräte sind auf einer mobilen Plattform auf einem Auto angebracht. Die Perspektive unterscheidet sich also geringfügig von der der Cityscapes-Daten. Für weitere Informationen über die verwendeten Messgeräte siehe [KITTI].

Im Gegensatz zum Cityscapes-Datensatz, der sich auf Straßenverkehr spezialisiert, wird im KITTI-Datensatz versucht, eine möglichst große Szenenvielfalt anzubieten. Beispiele für Bilder des Datensatzes sind in Abbildung ?? zu sehen. KITTI wird in dieser Arbeit verwendet, da darin, im Gegensatz zu Cityscapes, neben den Bildern Punktwolken und Projektionsmatrizen zur Verfügung gestellt werden. Die mit Cityscapes konformen Trainingsdaten für semantische Segmentierung erlauben noch dazu Experimente mit der Verfeinerung des Netzes durch Nachtraining und Auswertung der vom Netz produzierten Ergebnisse mit anderen Daten als dem Cityscapes-Datensatz.

5.3 COCO

Bei dem in [DBLP:journals/corr/LinMBHPRDZ14] von Microsoft vorgestellten COCO (Common Objects in Context) handelt es sich um einen umfangreichen Datensatz für Instanz-Segmentierung. Es werden darin 328.000 Bilder mit insgesamt 2,5 Millionen annotierten Instanzen geboten. Das Bildmaterial für COCO stammt aus dem Internet, wobei nicht-ikonische Bilder, die Objekte aus untypischen Perspektiven zeigen, bevorzugt wurden. Da sich die Annotationen aufzählbare Objekte beschränken, ist der COCO-Datensatz für den Zweck dieser Arbeit nicht geeignet.

5.4 Pascal VOC

Wie in [Everingham10] beschrieben, ist Pascal VOC (Visual Object Classes) eine Sammlung öffentlich zugänglicher Datensätze für Bildklassifizierung, Objekt-Detektion, Segmentierung und Personen-Layout (Detektierung von Körperteilen), sowie ein jährlicher Wettbewerb in diesen Bereichen im Zeitraum von 2007 bis 2012. Die Hauptdisziplinen sind dabei Bildklassifizierung und Objekt-Detektion. Der letzte Stand von Pascal aus dem Jahr 2012 umfasst insgesamt 11.530 Bilder mit 27.450 Region-of-Interest-annotierten Objekten und 6.929 Segmenten aus 20 verschiedenen Klassen. Die Bilder für die Datensätze stammen aus dem Internet und sind ohne Präferenz hinsichtlich Bildqualität, Perspektive, Beleuchtung und Ähnlichem ausgewählt. In [mottaghi_cvpr14] wird ein Datensatz vorgestellt, der den Anforderungen dieser Arbeit entspricht und eine valide Alternative zu Cityscapes darstellen könnte. Im gegebenen Kontext wird trotzdem Cityscapes verwendet wegen dessen stärkeren



Abbildung 5.2: Beispiele aus dem KITTI-Datensatz für semantische Segmentierung. Auf die Abbildung der Ground Truth wird verzichtet, da keine eingefärbte Version bereitgestellt wird und es sich nicht in erster Linie um einen Datensatz für Segmentierung handelt. Die Bilder sind von einer mobilen Plattform aufgenommen, die sich auf dem Fahrzeug befindet, weshalb die Perspektive sich von der der Cityscapes-Daten unterscheidet.

Praxisbezugs und konsequenteren Label-Politik. Dazu kommt, dass Pascal Daten aus allgemeinen Szenen anbietet, während sich Cityscapes auf den Straßenverkehr konzentriert und für dieses Thema leichter LiDAR-Daten gefunden werden können.

5.5 WildDash

WildDash [Zendel_2018_ECCV] ist ein Datensatz, der spezifisch für das Testen von Methoden zur Bildsegmentierung mit Szenen aus dem Straßenverkehr zusammengestellt ist. Ähnlich wie bei KITTI ist der Satz an annotierten Bildern nicht ausreichend, um einen adäquaten Trainingssatz zu bilden, ist aber konform mit Cityscapes und kann eine sinnvolle Erweiterung von dessen Trainingssatz darstellen.

Die Bilder in WildDash stammen von „YouTube-Autoren“ und sind nach speziellen Kriterien ausgewählt. Zu diesen gehört unter anderem, dass in den Daten Situationen gezeigt werden, die erfahrungsgemäß schwierig auszuwerten sind und potentiell Risiken darstellen, wie zum Beispiel Tiere auf der Fahrbahn und Tunnelausfahrten. Außerdem enthält der Datensatz negative Testfälle, also Bilder bei denen ein schlechtes Ergebnis erwartet wird, wie beispielsweise ein gedrehtes Bild.

Im Rahmen dieser Arbeit wird der KITTI-Datensatz als geeigneter angesehen, da darin auch Punktwolken gestellt und hauptsächlich städtische Szenen gezeigt werden. Außerdem erscheinen Testergebnisse auf dem WildDash-Datensatz weniger repräsentativ.

6 Experimente

In diesem Kapitel sollen die im Rahmen der Arbeit durchgeführten Experimente aufgeführt, sowie deren Ergebnisse dargelegt und diskutiert werden. Ziel der Experimente ist es, Praktiken zu ermitteln, die bei der Anwendung des entwickelten Systems vorteilhaft sind und mit denen die bestmöglichen Ergebnisse produziert werden.

Zunächst soll die Auswirkung des verwendeten Backbones auf die Leistungsfähigkeit des Netzes untersucht werden, wobei sich die Experimente auf MobileNetV2 konzentrieren. Dazu soll zunächst experimentell die ideale Trainingsdauer für die Verwendung von MobileNetV2 ermittelt werden. Anschließend daran wird das so erstellte Modell auf einem Evaluierungs-Datensatz ausgewertet. Die Ergebnisse eines Netzes, das Xception65 als Backbone verwendet sollen ebenfalls diskutiert und mit denen von MobileNetV2 bezüglich deren Qualität und notwendiger Rechenzeit verglichen werden. Als nächstes soll festgestellt werden, wie gut das Netz unter der Verwendung von MobileNetV2 Bilder aus einem anderen Datensatz verarbeitet und wie sich die Ergebnisse durch Anpassung des Trainingsverhaltens verbessern lassen. Als letztes sollen auf die in Abschnitt ?? beschriebene Weise segmentierte Punktwolken betrachtet und diskutiert werden. Dabei soll vor allem auf den Zusammenhang zwischen der Qualität der segmentierten Bilder und der der Punktwolken geachtet werden.

6.1 Backbones

Für die Durchführung folgender Experimente wird der fein annotierte Cityscapes-Datensatz verwendet. Die Netze werden auf dem aus 2975 Bildern bestehenden Trainingssatz trainiert und auf dem 500 Bilder fassenden Validierungssatz ausgewertet.

Für jede Trainingsepoke werden aus dem Trainingssatz 1487 Bilder zufällig ausgewählt. Als Grundlernrate wird 0.007 gewählt. Der Trainingsfehler wird über die in [Lovasz] beschriebene Lovasz-Funktion berechnet. Weitere Hyperparameter sind eine Dropout-Rate von 0.1, eine L2 Regularisierungsrate von $4 * 10^{-5}$ und ein Momentum-Faktor von 0.9.

Die Netzwerkausgaben werden mittels der im Bereich Bildsegmentierung üblichen

Intersection-over-union-Metrik (IoU) ausgewertet, die sich folgendermaßen berechnet:

$$IoU = \frac{TP}{TP + FP + FN} \quad (6.1)$$

Dabei steht TP für True Positives, also richtig erkannte Pixel, FP für False Positives und FN für False Negatives. Da in diesem System jedem Pixel eine gültige Klasse zugeordnet wird, sind FP und FN hier stets identisch.

6.1.1 MobileNetV2

Da Echtzeitfähigkeit eine wichtige Rolle spielt, konzentrieren sich die Experimente auf das leichtgewichtige MobileNetV2, statt dem leistungsfähigeren Xception65. Das erste Experiment beschäftigt sich mit dem Finden der idealen Trainingsdauer.

Epochen trainiert	1	5	10	15	25	30
IoU Straße	0.6260	0.6498	0.6727	0.6729	0.6774	0.6675
IoU Gehsteig	0.1835	0.3788	0.4330	0.4616	0.5193	0.5078
IoU Gebäude	0.5563	0.6402	0.6573	0.6771	0.6931	0.7068
IoU Mauer	0.0	0.0	0.0	0.0	0.0	0.0
IoU Zaun	0.0	0.0	0.0	0.0	0.0	0.0
IoU Pfahl	0.0791	0.1931	0.2159	0.2561	0.2835	0.2728
IoU Ampel	0.0	0.0	0.0	0.0	0.0	0.0
IoU Verkehrszeichen	0.0	0.1056	0.1785	0.1859	0.2233	0.2291
IoU Vegetation	0.5581	0.7153	0.7273	0.7518	0.7728	0.7663
IoU Gelände	0.0	0.0915	0.1274	0.1355	0.1593	0.1445
IoU Himmel	0.5353	0.6606	0.6867	0.6977	0.7153	0.7081
IoU Person	0.0	0.1160	0.1582	0.1458	0.1680	0.1863
IoU Radfahrer	0.0	0.0	0.0	0.0	0.0	0.0
IoU Auto	0.3537	0.5438	0.6014	0.5863	0.6419	0.6103
IoU Lastwagen	0.0	0.0	0.0	0.0	0.0	0.0
IoU Bus	0.0	0.0	0.0	0.0	0.0	0.0
IoU Zug	0.0	0.0	0.0	0.0	0.0	0.0
IoU Motorrad	0.0	0.0	0.0	0.0	0.0	0.0
IoU Fahrrad	0.0	0.0	0.0	0.0750	0.1340	0.1542
Durchschnitt IoU	0.1522	0.2155	0.2346	0.2444	0.2625	0.2607
Durchschnitt IoU $\neq 0$	0.4131	0.4094	0.4458	0.4222	0.4534	0.4503

Tabelle 6.1: Die Tabelle zeigt die Bewertung von Models in Intersection-over-union (IoU) Metrik in Abhängigkeit der Trainingsdauer.

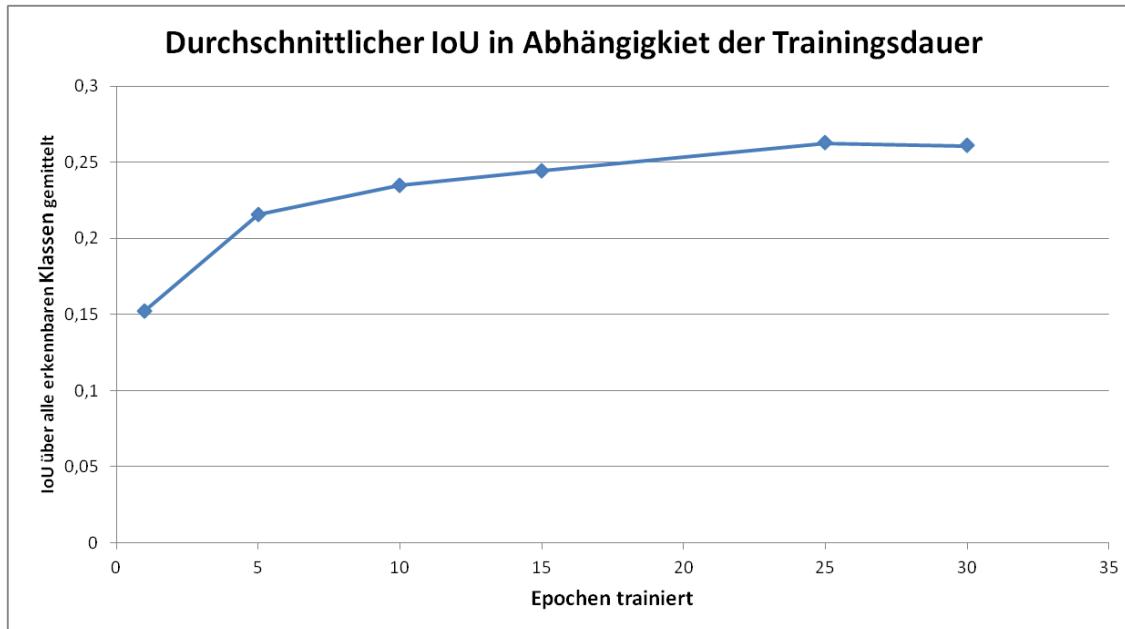


Abbildung 6.1: Durchschnittlicher IoU in Abhängigkeit der Anzahl trainierter Epochen. Der Graph erreicht sein Maximum bei 25 Epochen und fällt danach leicht ab, was auf Overfitting schließen lässt.

Wie in Tabelle ?? und Abbildung ?? zu erkennen ist, verbessert sich die Bewertung des Models bis zu einem Punkt, der in etwa bei Epoche 25 liegt und verschlechtert sich bei Verlängerung der Trainingsdauer wieder. Es tritt also trotz der L2 Regularisierung der Netzparameter und der Nutzung des Dropout-Verfahrens wahrscheinlich Overfitting auf. Im Folgenden wird das für 25 Epochen trainierte Netz verwendet. Die Bewertung dieses Models ist in Abbildung ?? dargestellt.

Die Modelle weisen durchwegs ihre höchsten Ergebnisse beim Erkennen der amorphen Klassen Straße, Gebäude, Vegetation und Himmel auf, wie bei einem semantischen Segmentierungsverfahren zu erwarten. Das niedrige Ergebnisse bei der Klasse Gelände lässt sich dadurch erklären, dass der Cityscapes-Datensatz in städtischen Umgebungen aufgenommen wird, wo diese Klasse selten auftritt. Vergleichsweise hoch ist auch der IoU-Wert für die Klasse Auto, die in dem Datensatz besonders häufig ist.

Die niedrigsten positiven IoU-Werte weisen die Ergebnisse bei kleineren,zählbaren Objekten wie Personen, Pfähle, Fahrräder und Verkehrszeichen auf.

Die in Abbildung ?? dargestellten Ergebnisse lassen außerdem erkennen, dass das Netz bestimmte Klassen praktisch nicht erkennt. Dies lässt vermuten, dass es nur eingeschränkt fähig ist, Bildsegmente anhand ihres Kontextes zu bewerten und beispielsweise zwischen einem Fußgänger und einem Fahrradfahrer oder zwischen einer

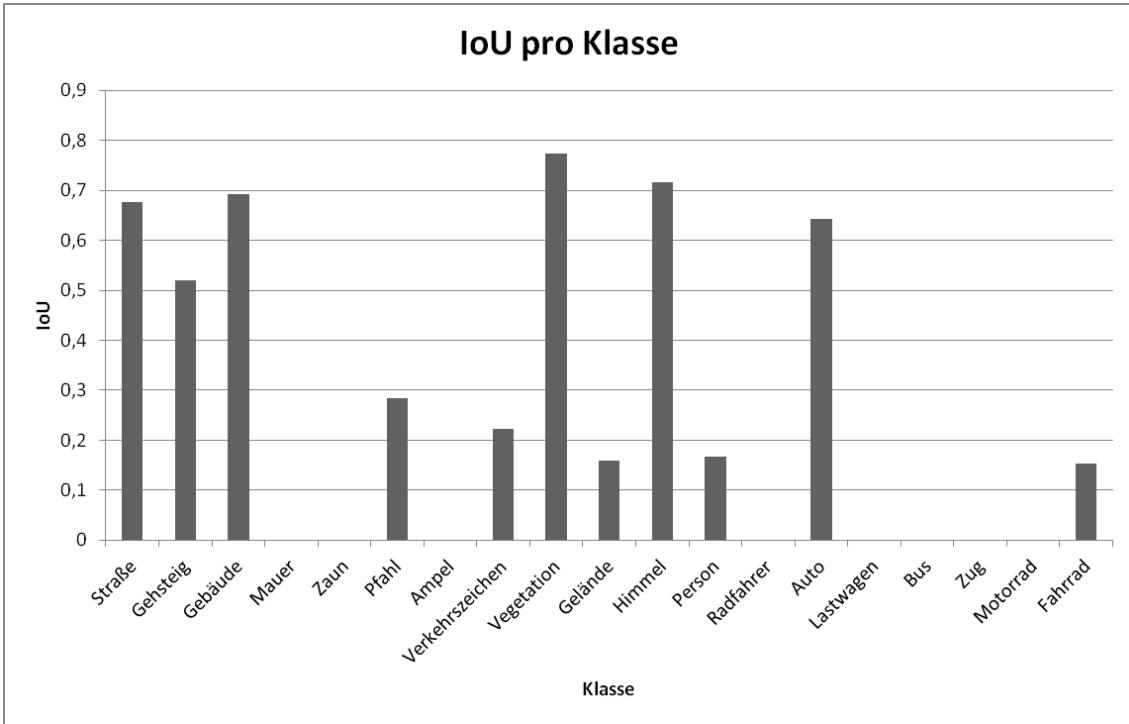


Abbildung 6.2: IoU für jede Klasse vom Test des für 25 Epochen trainierten Netzes. Das Diagramm zeigt gute Ergebnisse für amorphe Klassen und schlechtere für kleine Objekte. Zu sehen ist auch, dass einige Klassen, die in den Trainingsdaten selten vorkommen oder Ähnlichkeit mit anderen Klassen aufweisen, nicht erkannt werden.

Mauer und einem Gebäude zu unterscheiden und die häufiger auftretende Variante auswählt. Die Ergebnisse der Klasse Fahrrad lassen vermuten, dass ein längeres Training dieses Verhalten verbessern könnte. Da ein zu langes Training sich, wie vorher erwähnt, negativ auf den durchschnittlichen IoU auswirkt, wird in diesem Experiment aber davon abgesehen. Eine weitere Möglichkeit wäre, dem Trainingssatz mehr Daten hinzuzufügen, die vermehrt die entsprechenden Objekte enthalten.

Abbildung ?? zeigt ausgewählte Ausgaben des für 25 Epochen lang trainierten Netzes, das die besten Ergebnisse in der IoU-Metrik liefert. Sie spiegeln die Bewertung aus Tabelle ?? wieder, mit hoher Genauigkeit bei großen und amorphen Objekten und niedriger bei kleineren,zählbaren. Besonders auffällig sind False Positives der Klasse Person, die das Model oft an Fahrräder oder Bäume in der Nähe von Personen vergibt. Die Beispiele lassen erkennen, dass das Netz teilweise die Trainingsdaten memorisiert. So wird dem oberen Teil eines Fahrrades häufig die Klasse Person zugewiesen und

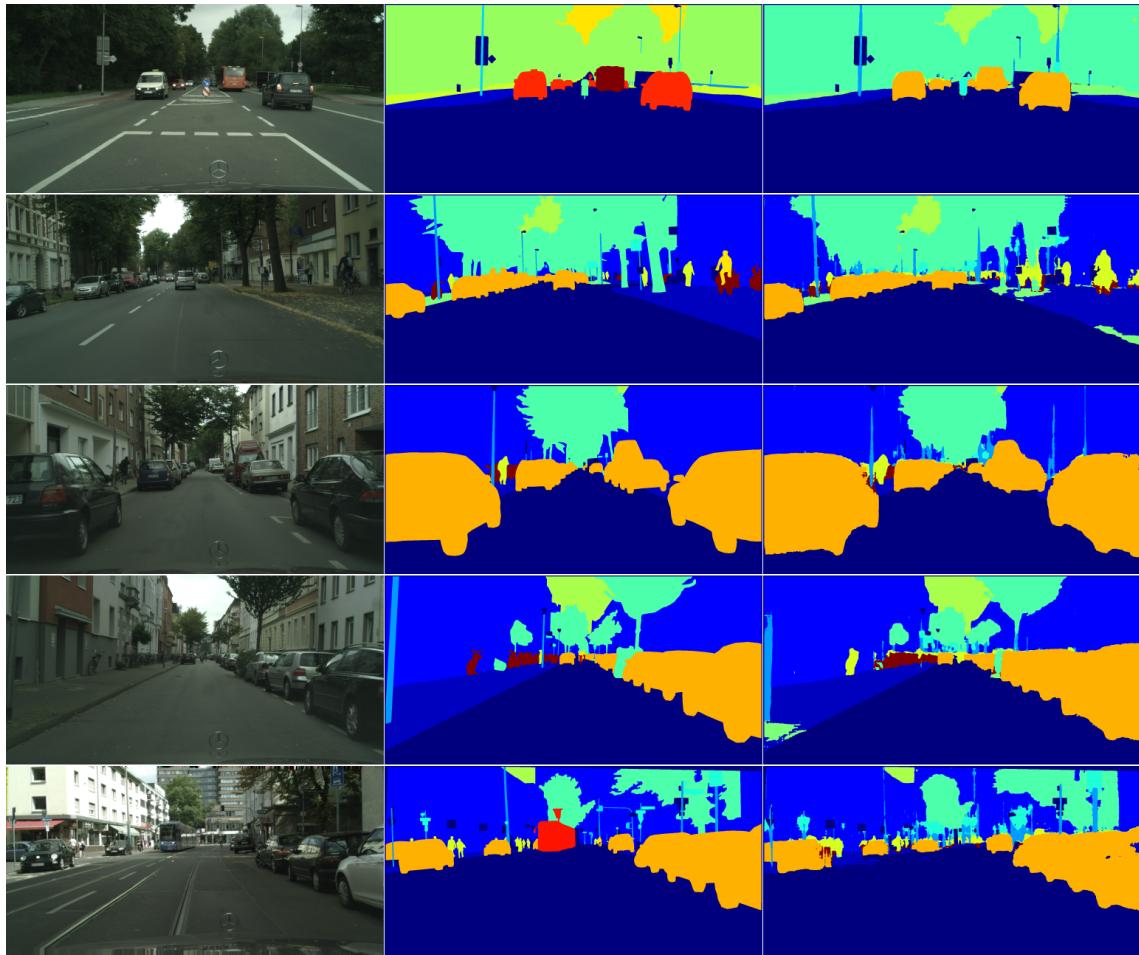


Abbildung 6.3: Zu sehen sind Beispiele für Ausgaben des Models mit der besten Bewertung im vorherigen Experiment (25 Epochen trainiert).
Von links nach rechts gezeigt sind Eingabebild, Ground Truth, Ausgabe von DeepLab.

umgekehrt der untere Teil einer Person als Fahrrad erkannt. Genauso werden Bereiche, die sich über einem als Straße erkannten Segments befinden tendenziell eher als Auto klassifiziert. Das lässt sich auf die Funktion des CRF zurückführen.

6.1.2 Xception65

Wir betrachten den Xception65-Backbone im Vergleich zu MobileNetV2. Auf ein Experiment mit unterschiedlichen Trainingsepochen wird an dieser Stelle aufgrund

der langen Trainingsdauer der Netze verzichtet. Das verwendete Model ist 60 Epochen lang trainiert mit denselben Hyperparametern wie die Models mit MobileNetV2. Da in diesem Experiment die für ein Bild nötige Verarbeitungszeit eine Rolle spielt, soll hier erwähnt werden, dass der Rechner auf dem es durchgeführt wird mit der Grafikkarte NVIDIA GeForce GTX 1070, auf der die Berechnungen durchgeführt werden, ausgestattet ist und über 16GB RAM verfügt.

	Xception65	MobileNetV2	Differenz
IoU Straße	0.6951	0.6774	0.0177
IoU Gehsteig	0.7238	0.5193	0.2045
IoU Gebäude	0.8533	0.6931	0.1602
IoU Mauer	0.1461	0.0	0.1461
IoU Zaun	0.1641	0.0	0.1641
IoU Pfahl	0.5843	0.2835	0.3008
IoU Ampel	0.3049	0.0	0.3049
IoU Verkehrszeichen	0.6592	0.2233	0.4359
IoU Vegetation	0.8558	0.7728	0.0830
IoU Gelände	0.2068	0.1593	0.0475
IoU Himmel	0.7707	0.7153	0.0554
IoU Person	0.5234	0.1680	0.3554
IoU Radfahrer	0.2386	0.0	0.2386
IoU Auto	0.8369	0.6419	0.1950
IoU Lastwagen	0.0691	0.0	0.0691
IoU Bus	0.0893	0.0	0.0893
IoU Zug	0.0185	0.0	0.0185
IoU Motorrad	0.0685	0.0	0.0685
IoU Fahrrad	0.4080	0.1340	0.4080
Durchschnitt IoU	0.4324	0.2625	0.2346
Durchschnittliche Verarbeitungszeit [ms]	764	387	377

Tabelle 6.2: Gezeigt ist ein Vergleich zwischen den Ergebnissen von DeepLab mit Xception65 und MobileNetV2 in IoU-Metrik und der Verarbeitungszeit.

Wie in Tabelle ?? und Abbildung ?? zu sehen ist, erzeugt das Model mit Xception65 durchwegs bessere Ergebnisse als das mit MobileNetV2, benötigt aber im Durchschnitt 97% mehr Zeit für die Verarbeitung eines Eingabebildes.

Durch die Verwendung von Xception65 ist das Model außerdem in der Lage, alle Klassen des Datensatzes zu erkennen, auch wenn die IoU-Werte für die von MobileNetV2 nicht erkannten Klassen vergleichsweise niedrig sind. Eine besonders große Steigerung der Bewertung im Vergleich mit MobileNetV2 weist das Netz bei kleineren, zählbaren

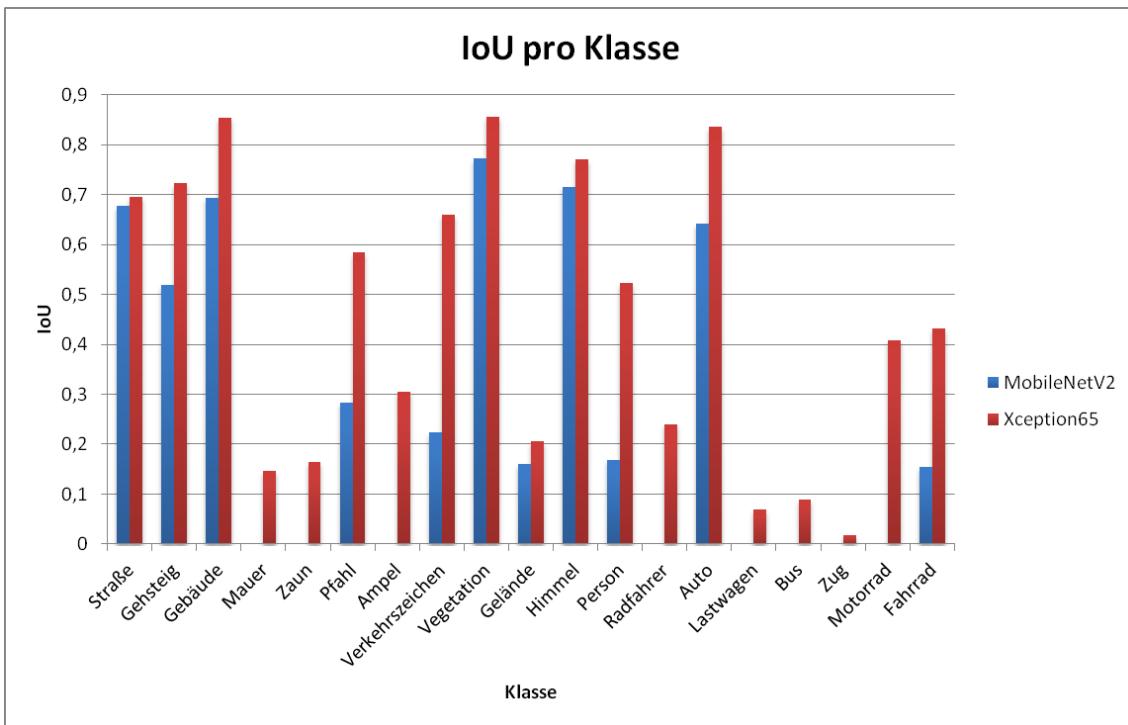


Abbildung 6.4: Der Vergleich von Xception65 und MobileNetV2 in IoU-Metrik zeigt, dass Xception die besten Ergebnisse bei denselben Klassen wie MobileNetV2 produziert, aber deutlich besser beim Erkennen von Objekten und Details ist.

Objekten auf wie Personen (311%) und Verkehrszeichen (295%) auf. Abbildung ?? zeigt beispielhafte Ergebnisse des Models aus dem Validierungs-Datensatz von Cityscapes.

6.2 Verfeinerung mit KITTI

In diesem Experiment wird das am besten bewertete, mit Cityscapes trainierte Model mit MobileNetV2 als Backbone mehrere Epochen mit den 200 Bildern umfassenden Trainingsdaten für semantische Segmentierung von KITTI nachtrainiert. Die Resultierenden Models werden anschließend mit anderen Bildern aus dem KITTI-Datensatz getestet. Da KITTI nur für diese 200 Bilder eine Ground Truth zur Verfügung stellt und ein Test auf den Trainingsdaten wenig aussagekräftig ist, begnügt sich das Experiment in diesem Fall mit einer empirischen Auswertung.

Abbildung ?? zeigt beispielhaft je ein Bild aus beiden Datensätzen und die Ausgaben

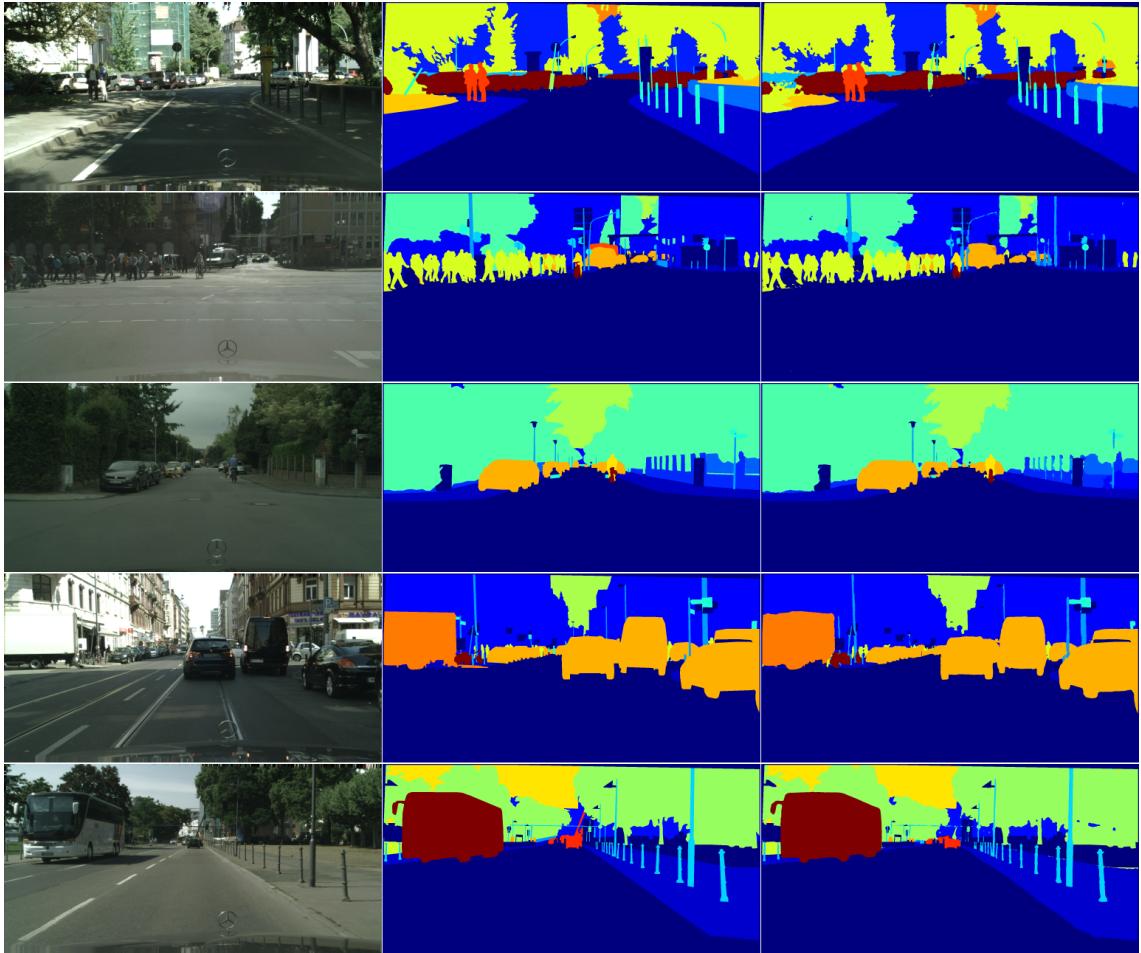


Abbildung 6.5: Gezeigt sind Beispiele für Ausgaben des Models, das Xception65 nutzt.

Von links nach rechts zu sehen ist Eingabebild, Ground Truth, Ausgabe von DeepLab.

von DeepLab für verschiedene lang nachtrainierte Modelle. Wie man sieht, tritt bei den Ergebnissen auf dem KITTI-Datensatz bereits nach einer Epoche Verfeinerung mit KITTI-Daten eine deutliche Verbesserung auf. Weitere Trainingsepochen verbessern die Ergebnisse weiter, aber weniger erheblich. Gleichzeitig verschlechtert sich die Ausgabe für ein Bild aus dem Cityscapes-Datensatz in ähnlichem Maße.

Das unterste Bild von Abbildung ?? zeigt die Ausgabe eines Models, das für 25 Epochen auf Cityscapes-Daten trainiert, für 5 Epochen mit KITTI-Daten verfeinert und anschließend für eine Epoche mit Cityscapes-Daten nachtrainiert ist. Wie man sieht, führt die Verfeinerung mit dem ursprünglichen Datensatz wieder zu einer Verschlechterung der Ergebnisse auf KITTI-Bildern und einer Verbesserung auf Cityscapes-Bildern.

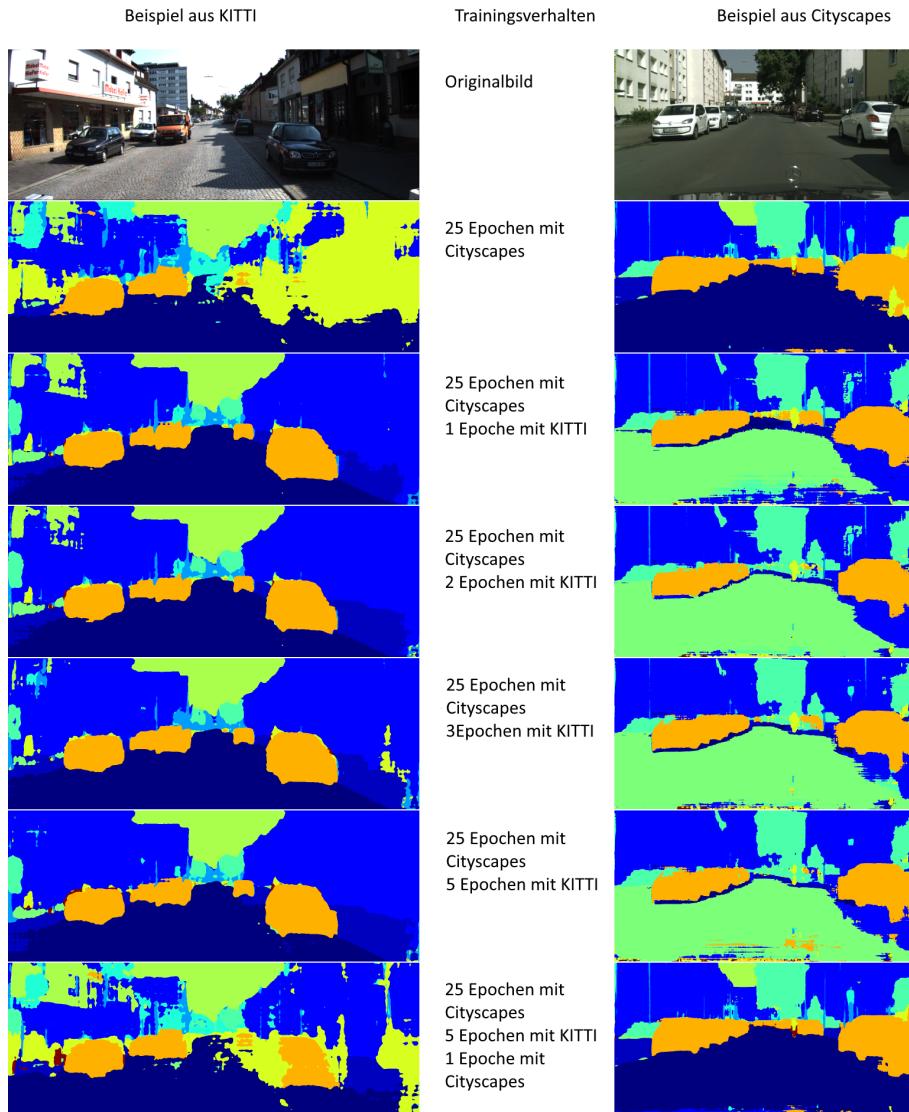


Abbildung 6.6: Beispiel für Ausgabe von mit KITTI-Daten verfeinerten Modellen. Die erste Epoche der Verfeinerung zeigt die größte Verbesserung auf den KITTI-Daten und gleichzeitig eine deutliche Verschlechterung bei Cityscapes-Bildern. Dieses Verhalten ist durch ein Nachtraining mit Cityscapes reversibel. Das Experiment zeigt, dass Overfitting ein Problem des Netzes ist.

Diese Beobachtungen bestärken die Vermutung, dass beim Trainieren des Netzes Overfitting auftritt.

Trainingsverhalten	Durchschnittlicher IoU auf Cityscapes-Bildern	Durchschnittlicher IoU auf KITTI-Bildern
25 Epochen mit Cityscapes-Daten (A)	0.2625	0.1681
24 Epochen mit ge- mischten Daten (B)	0.2605	0.2002
24 Epochen mit ge- mischten Daten + 1 Epoche mit KITTI- Daten (C)	0.2045	0.2350
24 Epochen mit ge- mischten Daten + 1 Epoche mit KITTI- Daten + 1 Epoche mit gemischten Daten (D)	0.2718	0.2105

Tabelle 6.3: Die Tabelle zeigt die Bewertung von Netzwerken mit unterschiedlich verfeinerten Trainingsdaten und -verhalten auf Evaluierungssätzen von Cityscapes und KITTI.

In einem nächsten Schritt soll untersucht werden, wie die Berücksichtigung der KITTI-Daten beim gesamten Lernprozess die Ergebnisse auf beiden Datensätzen beeinflusst. Dazu wird der annotierte KITTI-Datensatz geteilt in einen Trainings- und Evaluierungssatz zu je 100 Bildern. Dadurch wird es möglich, eine Aussagekräftige Bewertung in IoU-Metrik zu berechnen. Aufbauend auf dem vorherigen Experiment wird ein Model mit gemischten Daten für 24 Epochen trainiert (B) und ausgewertet. Danach wird es für eine Epoche ausschließlich mit KITTI-Daten verfeinert (C) und dann für eine weitere Epoche mit dem gesamten, gemischten Trainingssatz nachtrainiert (D). Damit ist das letzte Model 25 Epochen auf den gemischten Daten trainiert und hat damit genauso viele Lernschritte auf Cityscapes-Bilder absolviert wie das am besten Bewertete Model von Abschnitt ?? (A). Die Ergebnisse sind in Tabelle ?? eingetragen und in Abbildung ?? graphisch dargestellt.

Verglichen mit Model A zeigen die Ergebnisse bei Model B deutlich bessere Ergebnisse bei den KITTI-Daten, während die auf den Cityscapes-Daten etwa gleich sind. In Model C zeigt sich ein ähnliches Verhalten wie im vorherigen Experiment mit einer deutlichen Verbesserung bei Den KITTI-Daten und einer Verschlechterung bei den Cityscapes-Daten. Die Ergebnisse von Model D weisen auf beiden Datensätzen eine Verbesserung gegenüber denen von Model A auf, wobei der Unterschied bei den Bildern von KITTI deutlich größer ist. Allerdings verschlechtern sich die Ergebnisse von Model D bei den KITTI-Daten bezüglich Model C.

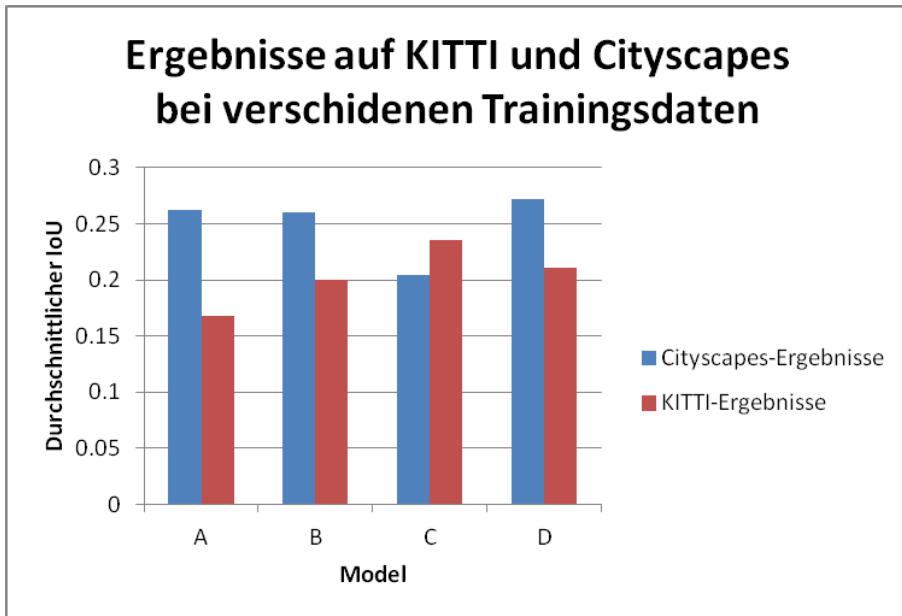


Abbildung 6.7: Graphische Darstellung der Ergebnisse von Tabelle ???. Besonders auffällig sind die Unterschiede bei den Ergebnissen auf den KITTI-Daten (rot) und der Einbruch bei denen der Cityscapes-Bilder (blau) in Model C.

Die Ergebnisse zeigen, dass das Hinzufügen zusätzlicher Trainingsdaten, was zu einer größeren Varietät darin führt, einen positiven Effekt auf die Leistung des Netzes hat. Abbildung ?? zeigt die Ausgabe der verschiedenen Models auf den Bildern vom vorherigen Versuch, einem aus dem KITTI-, einem aus dem Cityscapes-Datensatz. Bei den Ergebnissen des Cityscapes-Bildes zeigen die Models A, B und D ähnliche Resultate und Model C ein deutlich schlechteres, was die Bewertung aus Tabelle ?? widerspiegelt. Das Resultat von Model C ist jedoch augenscheinlich besser als die der mit KITTI verfeinerten Models in Abbildung ???. Bei dem Bild aus KITTI liefern Models B, C und D erwartungsgemäß deutlich bessere Ergebnisse als Model A. Das empirisch beste Resultat erzielt Model C, das aber das schlechteste auf dem Cityscapes-Bild produziert. Die Ergebnisse von Model D zeigen eine geringfügige, aber erkennbare Verbesserung gegenüber Model B. Dieses Experiment zeigt, dass eine hohe Varietät innerhalb der Trainingsdaten wünschenswert ist. Ist die Art der Daten bekannt, auf denen das Netz angewandt werden soll, ist es zudem ratsam, es mit annotierten Beispieldaten zu verfeinern.

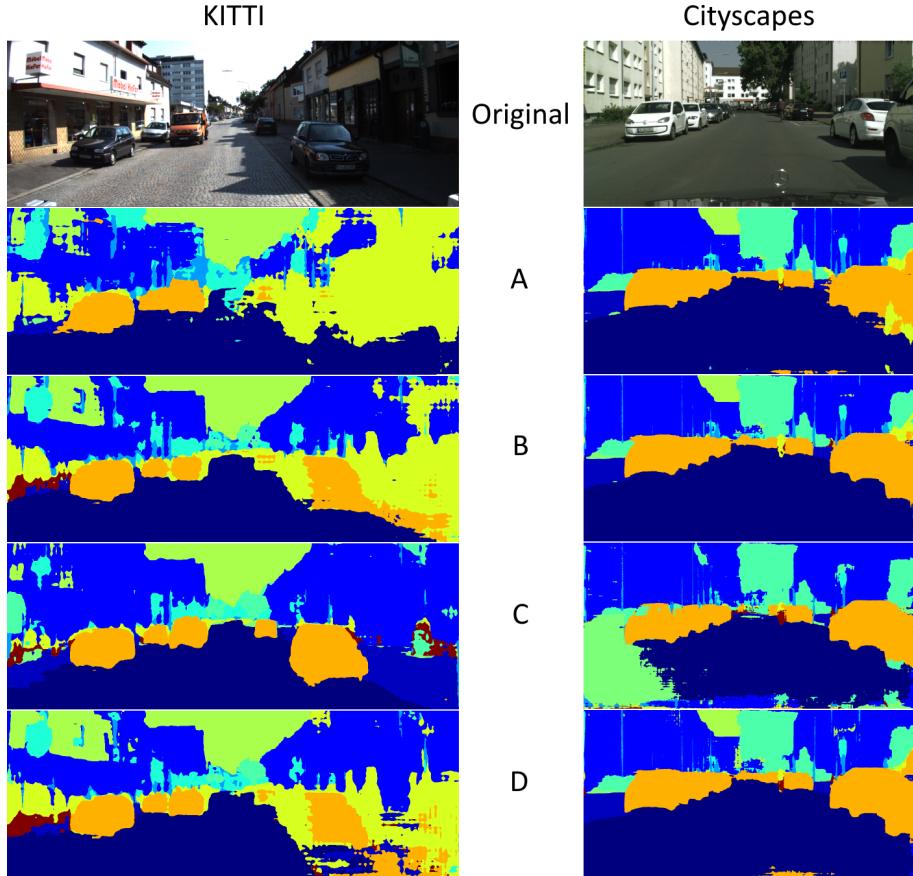


Abbildung 6.8: Hier sieht man beispielhaft Ergebnisse für ein Bild aus dem KITTI- und eines aus dem Cityscapes-Datensatz. Die Bilder sind dieselben wie in Abbildung ??, um einen Vergleich zu ermöglichen. Wie man sieht, zeigen die Models A, B und D ähnlich gute Ergebnisse auf dem Cityscapes-Bild. Auf dem KITTI-Bild erzielt die besten Ergebnisse das Model C, das die schlechtesten auf Cityscapes liefert. Auf dem KITTI-Bild erzeugen auch die Models B und D bessere Resultate als Model A, das komplett ohne KITTI-Daten trainiert ist.

6.3 Segmentierung von Punktwolken

In diesem Abschnitt sollen die Ergebnisse der Punktwolkensegmentierung diskutiert werden. Wie bereits erwähnt, werden die Experimente auf LiDAR-Daten des KITTI-Datensatzes durchgeführt. Da diese nicht annotiert sind, werden die Ergebnisse an dieser Stelle nur empirisch ausgewertet. In diesem Experiment wird das Model C aus

dem letzten Abschnitt verwendet, das 24 Epochen auf einer Mischung aus Cityscapes- und KITTI-Daten und eine Epoche nur auf KITTI-Daten trainiert ist und die höchste IoU-Bewertung auf den KITTI-Bildern erzielt. In Abbildung ?? und ?? sind zwei Beispiele für segmentierte Punktwolken analog zu Abbildung ?? gezeigt.

Offensichtlich ist die Genauigkeit der Punktwolkensegmentierung abhängig von der

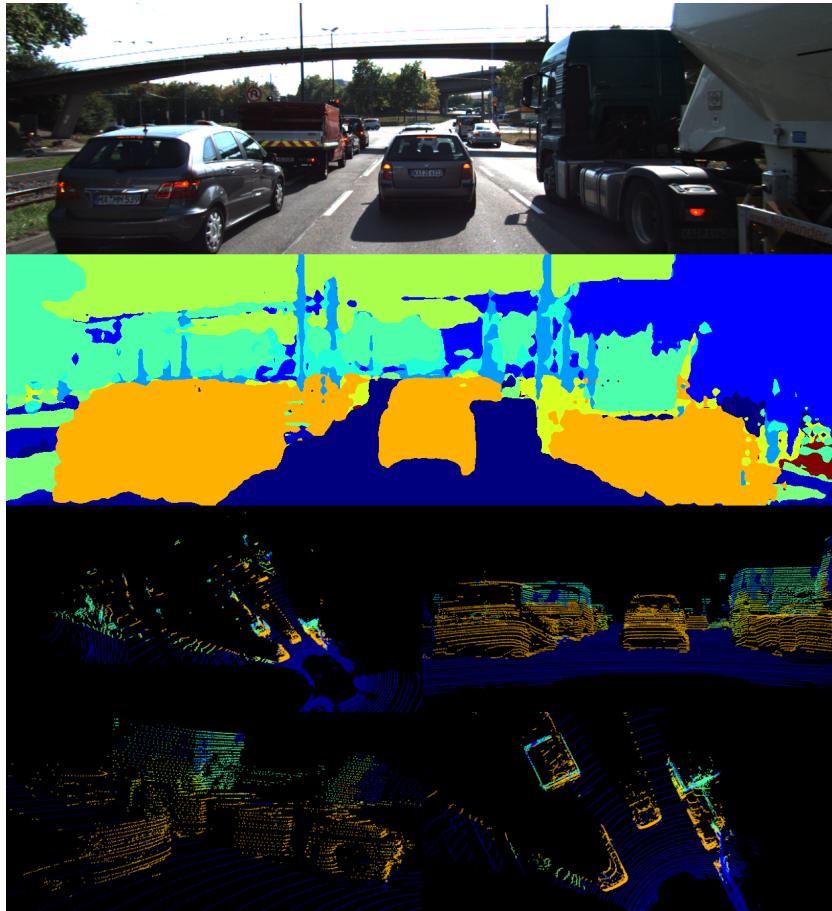


Abbildung 6.9: Beispiel für segmentierte Punktwolken. Zu sehen sind von oben nach unten: Originalbild, segmentiertes Bild, segmentierte Punktwolke aus vier verschiedenen Perspektiven. Gezeigt ist eine Szene, in der mehrere Fahrzeuge auf einer dreispurigen Straße an eine rote Ampel heranfahren. Im Hintergrund ist Vegetation. Die Segmentierung der Straße und Autos in der Punktwolke ist weitgehend Korrekt. Die Lastwagen werden im unteren Teil als Autos, im oberen als Gebäude oder Vegetation klassifiziert.

Qualität der Bildsegmentierung, was besonders in Abbildung ?? deutlich wird. Aufgrund der in Abschnitt ?? bestimmten Eigenschaft von MobileNetV2, nur häufig

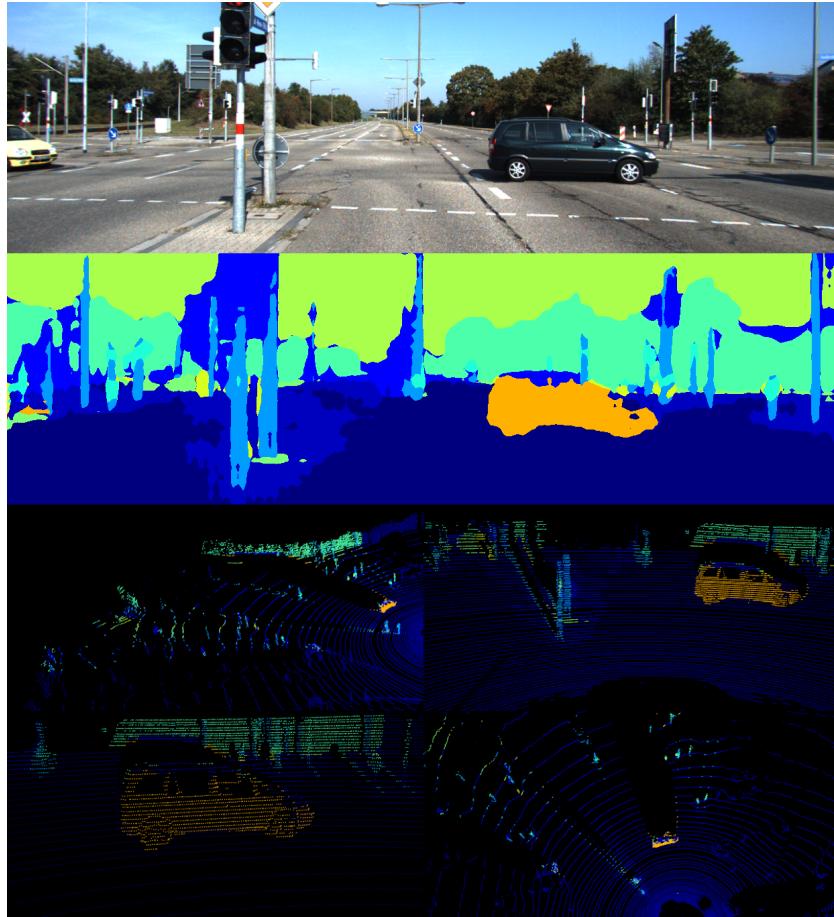


Abbildung 6.10: Beispiel für segmentierte Punktwolken. Zu sehen sind von oben nach unten: Originalbild, segmentiertes Bild, segmentierte Punktwolke aus vier verschiedenen Perspektiven. Gezeigt ist eine Szene an einer Kreuzung mit vielen Pfahl-artigen Objekten. Das filmende Fahrzeug hält an einer roten Ampel, während ein Auto die Kreuzung von links nach rechts durchfährt. Ein weiteres Auto fährt von links in die Bildfläche ein. Die Segmentierung des Autos auf dem Bild weist Fehler im oberen Bereich auf, die auch in der segmentierten Punktwolke wiederzufinden sind. Das nicht gut sichtbare Fahrzeug am linken Rand wird vom Netzwerk nicht korrekt erkannt. Die Pfähle werden in der Punktwolke gut segmentiert, obwohl sie im Bild nur ungenau erkannt werden.

vorkommende Klassen des Trainingssets zu erkennen, werden die Lastwagen im Bild im unteren Bereich als Autos und im oberen Bereich als Mischung aus Gebäuden und Vegetation erkannt, was dann in die Punktwolke projiziert wird.

Insgesamt zeigen die Beispiele gute Ergebnisse beizählbaren Objekten im Vordergrund, selbst wenn die Bildsegmentierung in diesen Bereichen nicht sehr präzise ist, wie in Abbildung ???. Die Segmentierung der Pfähle darin ist verschwommen und weist keine klaren Ränder auf. In der Punktwolke führt dies dazu, dass die Objekte selbst richtig und der Hintergrund hinter den Objekten falsch gelabelt wird.

Hinsichtlich dieser Eigenschaften wäre ein Netz wünschenswert, das die Klassezählbarer Objekte großzügig verteilt, wodurch diese in den Punktwolken richtig segmentiert werden. Die durch verschwommene Ränder im segmentierten Bild falsch erkannten Punkte in amorphen Objekten fallen im Vergleich zu der Menge an Punkten dieser Klasse nicht ins Gewicht. Dazu kommt, dass diese sich oft im Hintergrund befinden und für praktische Anwendungen von geringerem Interesse sind.

6.4 Aufgetretene Probleme und Lösungen

6.4.1 False Positives

Models, die MobileNetV2 benutzen neigen dazu, einige Klassen zum Großteil mit einer anderen zu klassifizieren. Beispiele dafür sind die bereits angesprochene Erkennung von Fahrrädern als Personen und die Klassifizierung großer Fahrzeuge als Gebäude. Bei Verwendung des Xception65-Backbones kommt es reproduzierbar bei bestimmten Bildern zu einem Phänomen, bei dem eine große Anzahl Pixel nahe einer der Ecken als Straße beziehungsweise die Klasse mit dem niedrigwertigsten Label klassifiziert wird. Es besteht kein erkennbarer Zusammenhang zwischen den Bildern, bei denen dieses Verhalten auftritt. Abbildung ?? zeigt Beispiele dafür.

6.4.2 Overfitting

Overfitting hat sich als eines der hauptsächlichen Probleme bei der Verwendung von MobileNetV2 herausgestellt. Schon kleine Änderungen in der Perspektive, wie sie beispielsweise im KITTI-Datensatz auftritt, verursachen eine spürbare Verschlechterung der Ergebnisse.

Um bessere Ergebnisse auf dem KITTI-Datensatz zu erzeugen, hat es sich als vorteilhaft herausgestellt, den Trainingssatz für semantische Segmentierung der KITTI-Daten beim Trainieren des Netzes mitzuberücksichtigen. Das Netz mit jenen Daten nachzutrainieren verbessert ebenfalls die Ergebnisse, kann aber wiederum zu Overfitting auf diesen Datensatz führen. Es kann hilfreich sein, nach dem Nachtraining noch eine

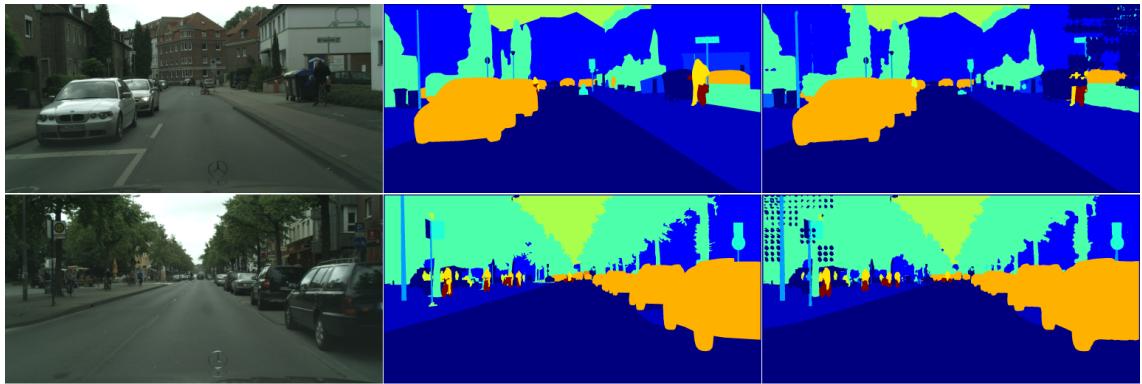


Abbildung 6.11: Hier zu sehen sind Beispiele für Ausgaben des Models mit Xception65, die das oben beschriebene Phänomen aufweisen. Von links nach rechts gezeigt sind Eingabebild, Ground Truth, Ausgabe von DeepLab. Das Verhalten ist auf den jeweiligen Bildern reproduzierbar.

Epoche mit allen verfügbaren Daten nachzutrainieren. Allgemein wirkt sich eine hohe Varietät innerhalb der Trainingsdaten positiv aus.

Weitere Möglichkeiten, gegen Overfitting vorzugehen, mit denen in dieser Arbeit aber nicht experimentiert wird, sind eine Erhöhung der Dropout-Rate und dem L2 Regularisierungsfaktor.

7 Zusammenfassung

Ziel der Arbeit war die Entwicklung eines Systems zur semantischen Segmentierung von Bildern und Punktwolken mit neuronalen Netzen. Es soll also jedem Pixel eines Bildes, beziehungsweise jedem Punkt einer Punktwolke eine Klasse zugeteilt werden, ohne zwischen Instanzen vonzählbaren Objekten zu unterscheiden wie das bei Instanz- oder panoptischer Segmentierung der Fall wäre.

Als Netzarchitektur wurde das von Google entwickelte DeepLab verwendet. Dabei handelt es sich um ein Deep Fully-Convolutional Neural Network, das durch das Adaptieren von Atrous Convolution, Atrous Spatial Pyramid Pooling un Conditional Random Fields auf den Bereich der semantischen Bildsegmentierung angepasst ist.

Wegen der Anforderungen an die Laufzeit wurde als Backbone das leichtgewichtige MobileNetV2 gewählt, das in Experimenten mit dem Leistungsfähigeren Xception65 verglichen wurde. Beide Netze sind als Residual Networks strukturiert, was bedeutet, dass über so genannte Rasidual Connections Daten über mehrere Verarbeitungsschichten hinweg unverändert weitergeleitet und auf die Ergebnisse der übersprungenen Schichten addiert werden. Dadurch wird verhindert, dass ein Hinzufügen weiterer Schichten die Ergebnisse verschlechtert. Eine weitere Technik, die beide Architekturen verwenden ist Depthwise Separable Convolution, bei der zuerst eine räumliche und dann eine dimensionsübergreifende Faltung durchgeführt wird. MobileNetV2 zeichnet sich durch die Verwendung von Bottleneck-Blöcken aus. Darin werden die Daten zuerst Expandiert, dann Komprimiert, um Speicherplatz zu sparen.

Da diese Arbeit das Ziel hat, Algorithmen für Autonomes Fahren zu unterstützen und verbessern, wurde die Implementierung mit Hilfe der Datensätze Cityscapes und KITTI ausgewertet, die explizit für diesen Bereich konzipiert sind. Dabei dienten die 5000 fein annotierten Bilder für Segmentierung von Cityscapes als hauptsächlicher Datensatz zu Training und Evaluierung des Netzes. Der KITTI-Datensatz liefert Laserscans beziehungsweise Punktwolken und Bilder, die mit kalibrierten Kameras aufgenommen wurden, was es ermöglicht, die auf den Bildern erkannten Labels durch Berücksichtigung der Projektionsmatrix auf die zugehörigen Punktwolken zu projizieren.

Die ideale Trainingsdauer von 25 Epochen wurde für ein Netz mit MobileNetV2 experimentell ermittelt. Die Ergebnisse der verschiedenen Models wurden dafür in IoU-Metrik bewertet. Das Netzwerk erzielte dabei die besten Ergebnisse beim Erkennen amorpher Objekte, schlechtere beim Erkennen von Details im Bild. Der

Vergleich mit einem Xception65-Model ergab, wie zu erwarten war, dass Xception bessere Ergebnisse bei längerer Rechenzeit liefert. Als größtes Problem stellte sich Overfitting heraus, wie ein Versuch mit Verfeinerung mit Trainingsdaten aus dem KITTI-Datensatz zeigt. Bereits nach einer Trainingsepoch mit einem vergleichsweise kleinen Datensatz verschlechterten sich die Ergebnisse auf dem Cityscapes-Datensatz merklich. Es stellte sich als vorteilhaft heraus, die KITTI-Daten beim Lernprozess miteinzubeziehen und so die Varietät in den Trainingsdaten zu erhöhen.

Zukünftige Experimente könnten zu Ziel haben, die Hyper-Parameter wie Dropout- und Regularisierungs-Rate zu anzupassen. Auch das Hinzufügen und Entfernen von Verarbeitungsschichten im Netzwerk könnte eine Möglichkeit sein, die Ergebnisse zu verbessern. Zur Verbesserung der Laufzeit könnten Bilder mit unterschiedlicher Auflösung getestet werden.