



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

31.12.2001

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 31.12.2001

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 Einleitung | 1 |
| 1.1 Segmentierung | 1 |
| 1.1.1 Semantische Segmentierung | 1 |
| 1.1.2 Instanz-Segmentierung | 2 |
| 1.1.3 Panoptische Segmentierung | 2 |
| 1.2 Ziele und Anforderungen | 2 |
| 2 Literatur | 3 |
| 2.1 Verwandte Arbeiten | 3 |
| 2.1.1 PointNet | 3 |
| 2.1.2 UPSNet | 4 |
| 3 Theorie | 7 |
| 3.1 DeepLab | 7 |
| 3.1.1 Convolutional Neural Networks | 7 |
| 3.1.2 Anpassungen für Semantische Segmentierung | 8 |
| 3.1.3 Atrous Convolution | 9 |
| 3.1.4 Atrous Spatial Pyramid Pooling | 11 |
| 3.1.5 Conditional Random Fields | 11 |
| 3.1.6 Residual Networks | 12 |
| 3.2 Kamerakalibrierung | 12 |
| 4 Arbeitsmethodik und Entwicklung | 15 |
| 4.1 Integration von DeepLab | 15 |
| 4.2 Backbones | 17 |
| 4.2.1 Xception | 18 |
| 4.2.2 MobileNetV2 | 18 |
| 4.3 Segmentierung von Punktwolken der KITTI-Daten | 19 |
| 5 Datensätze | 21 |
| 5.1 Cityscapes | 21 |
| 5.2 KITTI | 21 |

| | |
|---|-----------|
| 6 Experimente | 23 |
| 6.1 Technische Daten des für die Experimente verwendeten Rechners | 23 |
| 6.2 Backbones | 23 |
| 6.2.1 MobileNetV2 | 23 |
| 6.2.2 Xception65 | 28 |
| 6.3 Verfeinerung mit KITTI | 29 |
| 6.4 Aufgetretene Probleme und Lösungen | 31 |
| 6.4.1 False Positives | 31 |
| 6.4.2 Overfitting | 31 |
| 7 Zusammenfassung | 35 |
| Literaturverzeichnis | 37 |

1 Einleitung

Für zahlreiche Entwicklungsthemen der heutigen Zeit, wie beispielsweise autonomes Fahren, 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 zu segmentieren.

1.1 Segmentierung

Segmentierung bezeichnet einen Vorgang, bei dem ein Bild nach bestimmten Heterogenitätskriterien 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, wie in [ups] beschrieben, semantische Segmentierung, Instanz-Segmentierung und panoptische Segmentierung. Für weitere Informationen siehe [GW08]

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 jeden prozessierten Bild gleich.

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.

1.1.3 Panoptische Segmentierung

Die panoptische Segmentierung stellt eine Kombination der vorherigen Segmentationsarten 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.

1.2 Ziele und Anforderungen

Ziel der Arbeit ist es, ein System zu entwickeln, das mit Hilfe von neuronalen Netzen ein Bild semantisch segmentiert und aufgrund der so entstandene Labels auf Pixelebene eine Punktfolge derselben Szene segmentiert. Der Anwendungsbereich des Systems soll autonomes Fahren sein, weshalb Entwicklung und Experimente mit Datensätzen für diesen durchgeführt werden. Konkret soll die Ausgabe des Systems Algorithmen für Einfädelvorgänge an Kreuzungen verbessern. Besondere Wichtigkeit kommt daher der Erkennung von Fahrzeugen, Personen und Straßen zu. Eine Kernanforderung ist dabei Echtzeitfähigkeit. Optimierung der Laufzeit ist also essentiell. Weiterhin soll das System transportabel, leicht zu verwenden, benutzerfreundlich und ressourcenschonend sein.

Die Entwicklung erfolgt in Python mit CUDA-Unterstützung unter Verwendung des von Google entwickelten Framework DeepLab, das zur Zeit der Entstehung dieser Arbeit als State-of-the-Art angesehen wird.

2 Literatur

2.1 Verwandte Arbeiten

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

2.1.1 PointNet

Das 2017 in [pnet] vorgestellte PointNet ist ein neuronales Netzwerk zum Auswerten von Punktwolken. Das Netz bietet dabei sowohl eine Architektur zur Klassifizierung als auch eine zur Segmentierung von Punktwolken. Der Strukturelle Aufbau ist in Abbildung 2.1 dargestellt. Problematisch an der Auswertung von Punktwolken mit 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. Soll das Netzwerk eine Segmentierung der Punktwolke vornehmen, wird der globale Feature Vektor mit dem Eingabevektor kombiniert, um einen Vektor zu erzeugen, der sowohl globale als auch lokale Features repräsentiert. Anschließend kann ein Label für jeden Punkt geschätzt werden.

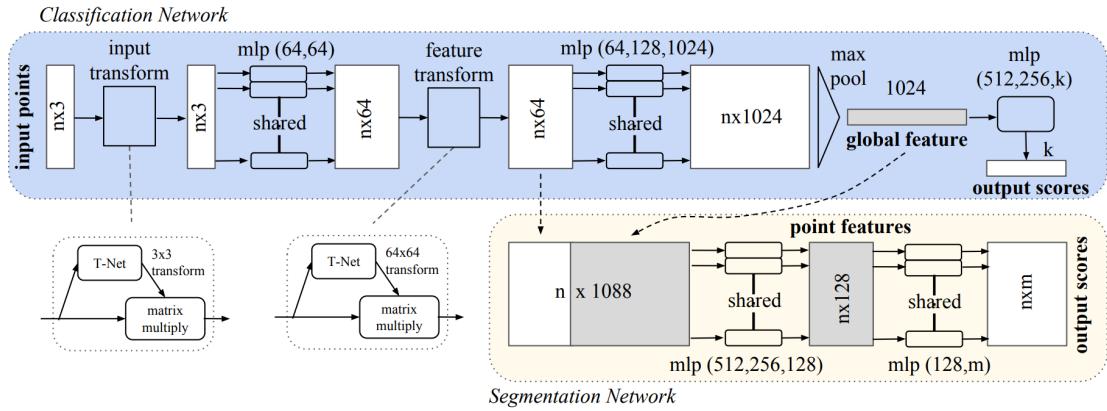


Abbildung 2.1: Funktionsweise von PointNet

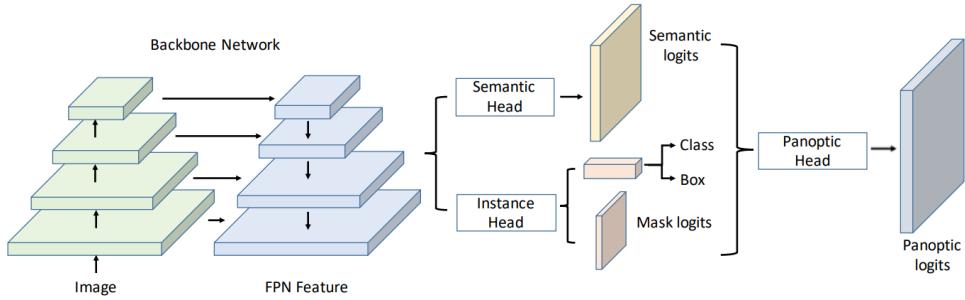


Abbildung 2.2: Architektur von UPSNet

2.1.2 UPSNet

Das 2019 in [ups] 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 2.2 dargestellt.

UPSNets verwendet als Backbone das in [rcnn] beschriebene Mask R-CNN, das sich aus dem ResNet 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 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.

3 Theorie

3.1 DeepLab

DeepLab ist ein von Google entwickeltes, 2015 in [dl1] vorgestelltes Modell für semantische Segmentierung. Bei der in [dl2] vorgestellten Methode wird ein Deep Convolutional Neural Network (DCNN) zum Erzeugen einer Score Map benutzt, die anschließend mit einem Conditional Random Field (CRF) zur endgültigen Ausgabe weiterverarbeitet wird. Das Verfahren wird in Abbildung 3.1 grob dargestellt.

3.1.1 Convolutional Neural Networks

Wie in [GBC16] 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. 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 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, wie bei einer Matrixmultiplikation. 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

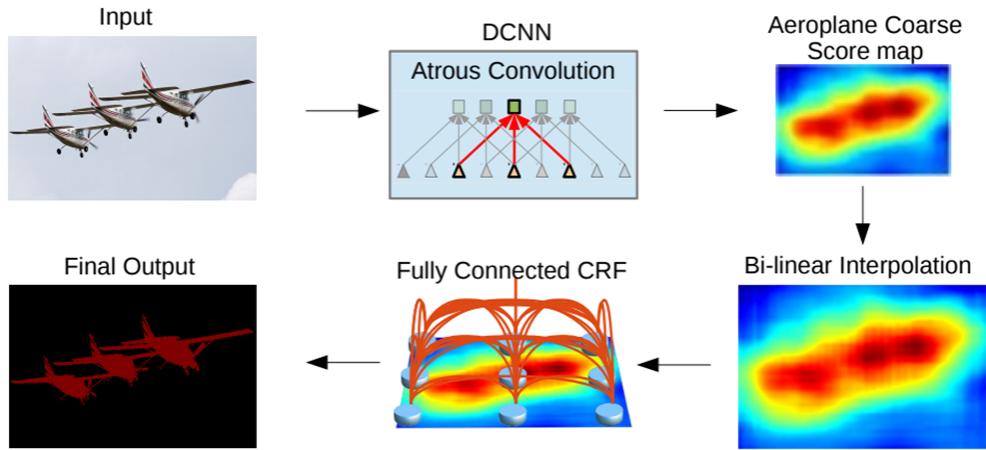


Abbildung 3.1: Grundsätzliche Funktionsweise von DeepLab

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 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 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.

3.1.2 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 Klassifizierungsaufgaben 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 verrauchte 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 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. 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.

3.1.3 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 3.2 illustriert.

Die Abstände der zu berücksichtigenden Werte in der Matrix wird dabei durch die so genannte Dilation Rate bzw. Erweiterungsrate (kurz Rate) festgelegt. Das Tatsächliche Sichtfeld des Filters wird also festgelegt 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. Die Vorteile der Verwendung von Atrous Convolution für Bildsegmentierung sind in Abbildung 3.3 dargestellt.

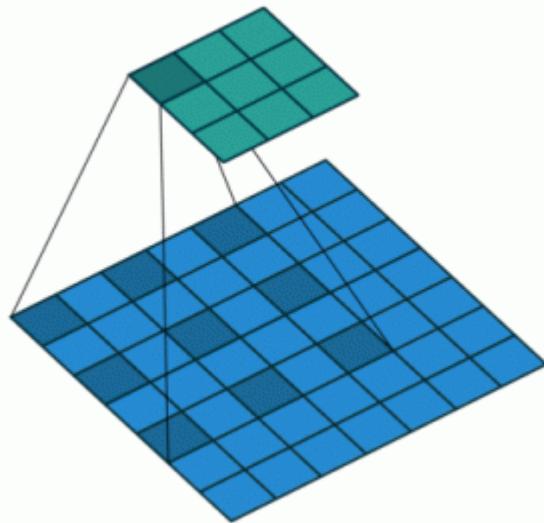


Abbildung 3.2: Prinzip von Atrous Convolution

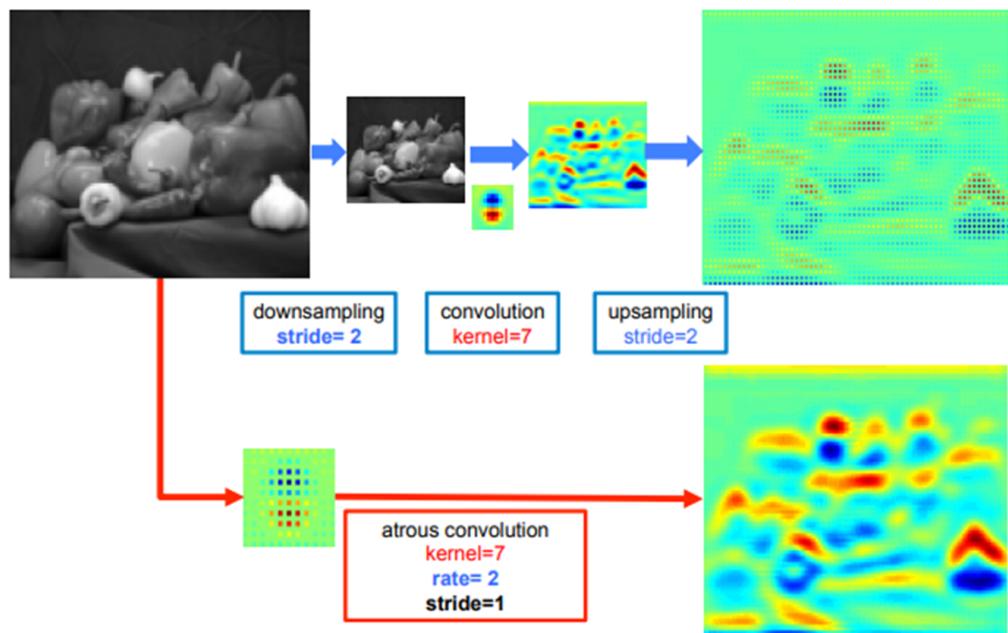


Abbildung 3.3: Beispielhaft dargestellte Vorteile von Atrous Convolution

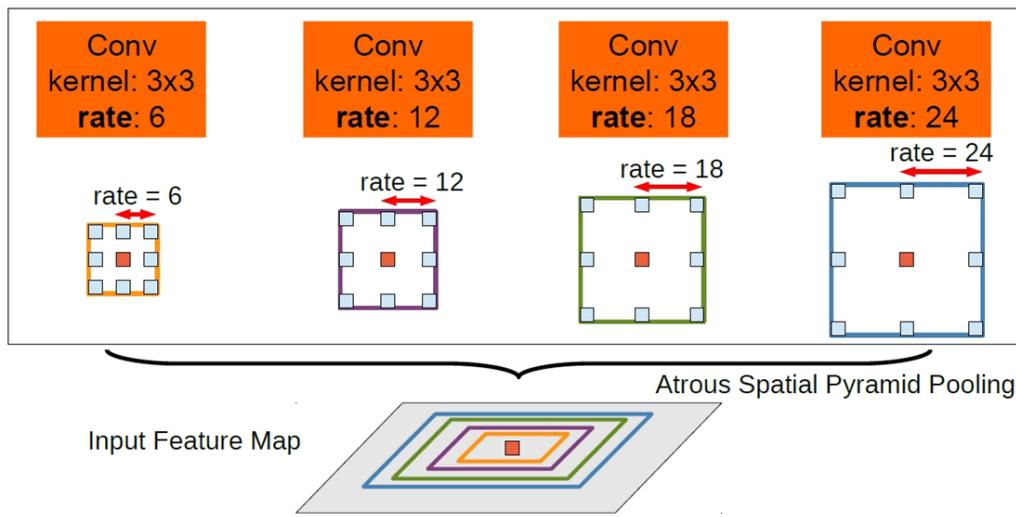


Abbildung 3.4: Beispielhaft dargestellte Vorteile von Atrous Convolution

3.1.4 Atrous Spatial Pyramid Pooling

Beim Atrous Spatial Pyramid Pooling werden mehrere parallele Convolutional Layers, die Atrous Convolutional Layers mit unterschiedlicher Rate verwenden, in das DCNN eingebaut. Aus den Ergebnissen der Verarbeitungszweige wird ein Tensor gebildet, der anschließend einer Dimension-übergreifenden 1x1 Faltung unterzogen wird, um die endgültigen Wahrscheinlichkeiten zu berechnen. Das Prinzip ist in Abbildung 3.4 dargestellt.

Durch dieses Vorgehen soll Größeninvarianz erreicht werden.

3.1.5 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 [crf] beschrieben, überwachtes Lernen ein, um Parameter für eine Funktion zu bestimmen, die die Verteilung $p(X|Y)$ beschreibt, wobei X und Y Zufallsvariablen sind. X beschreibt die beobachtete Eingabesequenz, Y die zu bestimmende Ausgabesequenz. 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. Allerdings wäre das für lange Sequenzen zu aufwändig, weshalb in praktischen Anwendungen andere Optimierungsalgorithmen eingesetzt werden. Wird für die Berechnung eines Wertes der Ausgabe alle Einträge der Eingabe betrachtet,

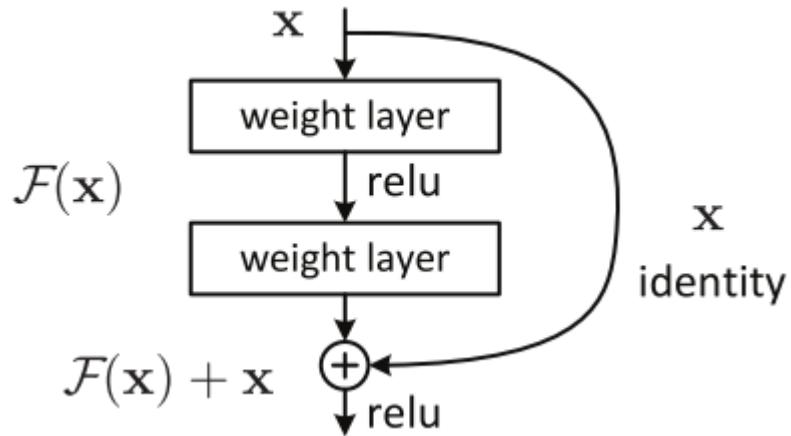


Abbildung 3.5: Prinzip von Residual Learning

spricht man von einem Fully Connected Conditional Random Field. Für weitere Informationen und eine präzise Definition siehe [LMP01].

3.1.6 Residual Networks

Ein Residual Neural Networks (ResNet) ist ein neuronales Netz, das das in [HZRS15] vorgestellte Residual Learning implementiert. Dabei werden, wie in 3.5 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. Die Überlegung dabei ist, dass weitere Schichten die Resultate nicht verschlechtern können, wenn die Ausgabe vorheriger Schichten noch unverändert vorhanden ist. Experimente bestätigen diese These. Residual Learning ist heute ein gebräuchliches Mittel beim Einsatz von vielschichtigen Netzwerken.

3.2 Kamerakalibrierung

Eine Kamera ermöglicht es, ein dreidimensionales Objekt auf eine zweidimensionale Ebene zu projizieren. Diese Projektion kann, wie in [HZ03] beschrieben, mit dem

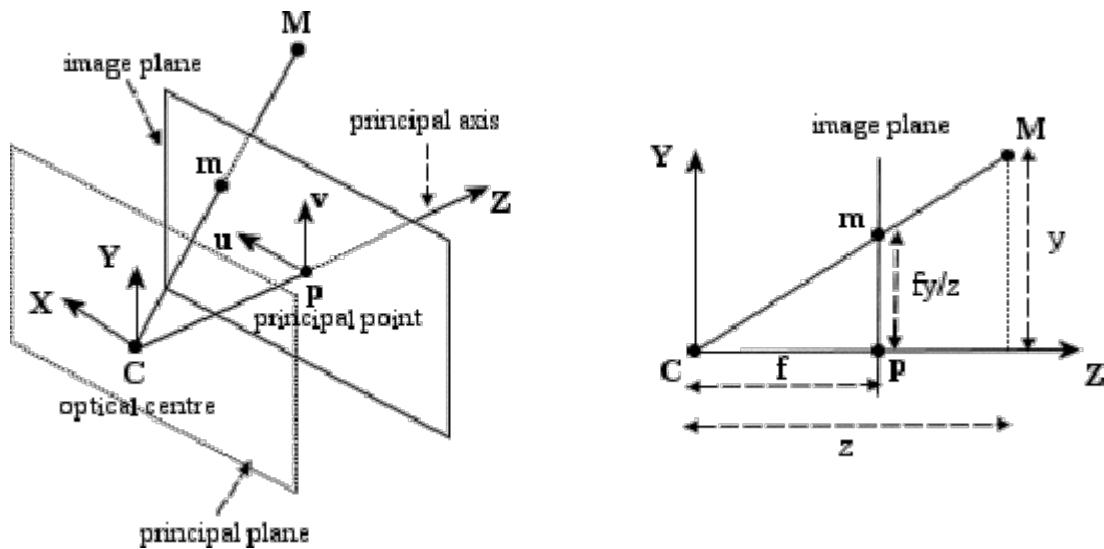


Abbildung 3.6: Prinzip einer Lochkamera zur Projektion vom dreidimensionalen Raum in den zweidimensionalen Raum [Fus06]

in Abbildung 3.6 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 über Entfernung, Längen, Winkel, Verhältnisse und dementsprechend Formen nicht erhalten bleiben.

Dass eine Kamera kalibriert ist, bedeutet im Praktischen Sinn, dass eine Matrix P bekannt ist, für die gilt: $m = PM$. M bezeichnet dabei die homogenen Koordinaten eines Punktes im dreidimensionalen Raum und m diejenigen von dessen Projektion auf der Bildebene. Für weitere Informationen über homogene Koordinaten siehe [HZ03]. 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 und umgekehrt. Offensichtlich kann damit berechnet werden, wo ein mit einer kalibrierten Kamera aufgenommener Punkt

im Bild einer anderen erscheint.

4 Arbeitsmethodik und Entwicklung

4.1 Integration von DeepLab

Als Basis für die Integration von DeepLab dient der Entwicklungsstand vom 17.Mai 2019 einer öffentlich zugänglichen Implementierung in Python mit dem Pytorch-Framework mit dem Titel „pytorch segmentation“, erstellt von Hiroki Taniai.

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-Models 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. 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 erforderte 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 gelabelten 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 4.1 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

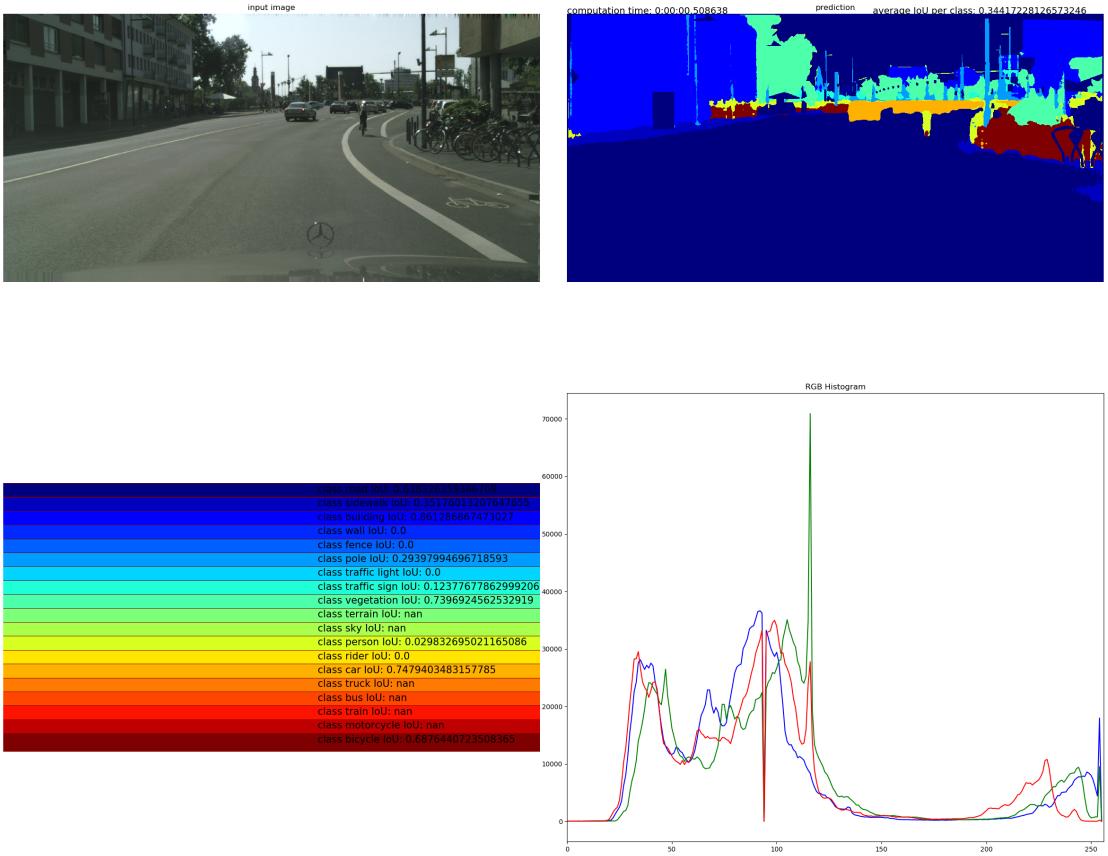


Abbildung 4.1: Beispiel für die Ausgabe des Evaluierungs-Skripts

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 mit einer Änderung in den Nomenklatur-Regeln 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 ignorierendes Label (255 in Cityscapes)
- Optimierer
- 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.

4.2 Backbones

Die hier verwendete Implementierung von DeepLab umfasst zwei Backbones: Xception65 und MobileNetV2.

4.2.1 Xception

Das in [xce] 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 4.2 dargestellt.

Das vorgestellte Netz hat insgesamt 36 Faltungsschichten, die in 14 Module gegliedert

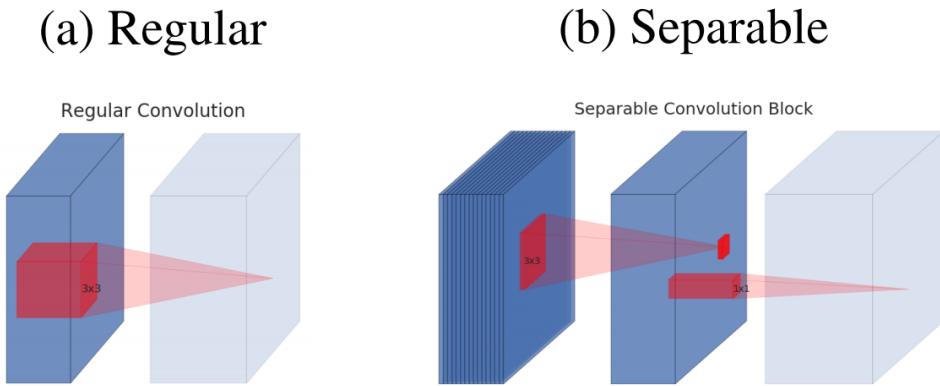


Abbildung 4.2: Schematische Darstellung des Prinzip von Depthwise Separable Convolution zur Tensorverarbeitung [mn2]

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.

4.2.2 MobileNetV2

Bei MobileNetV2 [mn2] 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.

4.3 Segmentierung von Punktwolken der KITTI-Daten

Der KITTI-Datensatz bietet sowohl Aufnahmen von kalibrierten Farbkameras mit der Projektionsmatrix als auch Laserscans in Form von Punktwolken 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 in diesem Projekt.

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. Das macht es einfach, die mit DeepLab gewonnenen Labels auf die Punktwolken zu projizieren. Zu beachten ist, dass dabei nur Punkte beachtet werden können, die sich im Sichtfeld des Bildes befinden. Für die Visualisierung der Ergebnisse wird eine Python-Integration von Mayavi verwendet.

5 Datensätze

5.1 Cityscapes

Cityscapes ist ein öffentlich zugänglicher Datensatz für Segmentierung. 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 war, bei einer Frame-Rate von 17Hz. Die Bilder des Datensatzes sind kalibriert, Bayer-gefiltert und rektifiziert. Für weitere Informationen siehe [COR⁺16].

5.2 KITTI

Das in [kit] 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. 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 [kit].

Im Gegensatz zum Cityscapes-Datensatz, der sich auf Straßenverkehr spezialisiert, wird im KITTI-Datensatz versucht, eine möglichst große Szenenvielfalt anzubieten.

6 Experimente

6.1 Technische Daten des für die Experimente verwendeten Rechners

- Prozessor: Intel(R) Core(TM) i7-7700HQ CPU @ 2.860GHz
- Grafikkarte: NVIDIA GeForce GTX 1070
- Arbeitsspeicher: 16GB RAM

6.2 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 [lov] 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.

6.2.1 MobileNetV2

Da Echtzeitfähigkeit eine wichtige Rolle spielt, konzentrieren sich die Experimente auf das leichtgewichtige MobileNetV2, statt dem leistungsfähigeren Xception65.

Wie in Tabelle 6.1 und Abbildung 6.1 zu erkennen ist, verbessert sich die Bewertung

| 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 ≠ 0 | 0.4131 | 0.4094 | 0.4458 | 0.4222 | 0.4534 | 0.4503 |

Tabelle 6.1: Bewertung von Models in Intersection-over-union (IoU) Metrik in Abhängigkeit der Trainingsdauer. Berechnung des IoU durch:
 $\text{IoU} = \text{True Positives} / (\text{True Positives} + \text{False Positives} + \text{False Negatives})$

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 ist in Abbildung 6.2 dargestellt.

Die Models weisen durchwegs ihre höchsten Ergebnisse beim Erkennen der amorphen Klassen Straße, Gebäude, Vegetatio 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 in der Klasse Auto, die in dem Datensatz besonders häufig ist.

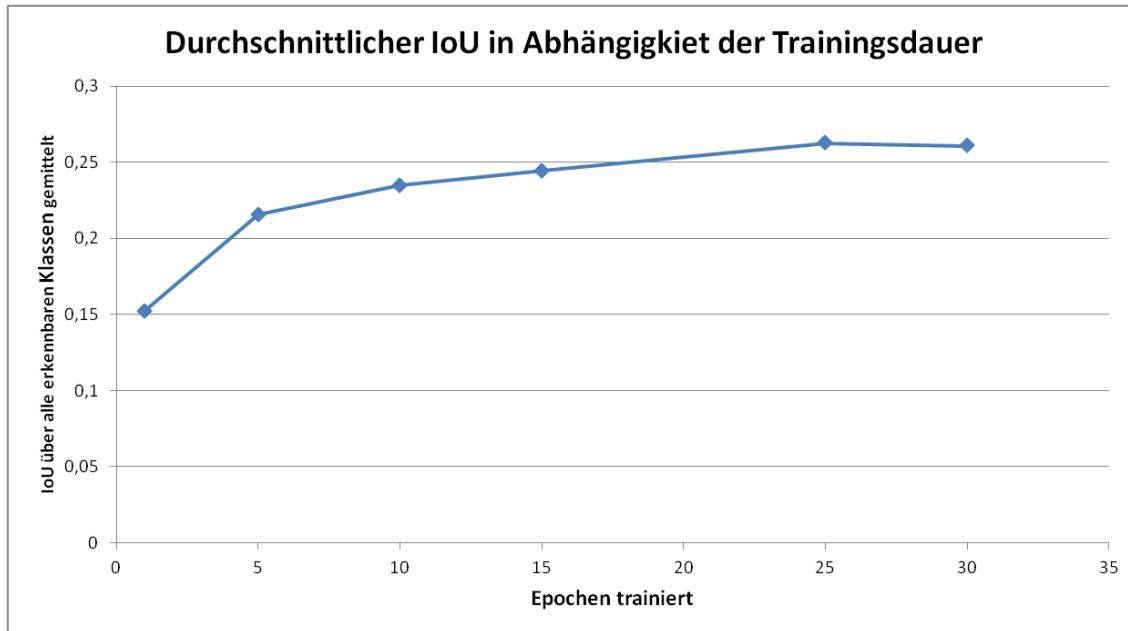


Abbildung 6.1: Durchschnittlicher IoU in Abhängigkeit der Anzahl trainierter Epochen

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

Die in Abbildung 6.2 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 Mauer und einem Gebäude zu unterscheiden und die häufiger auftretende Variante auswählt. Die Ergebnisse der Klasse Fahrrad lassen vermuten, dass ein längeres Training dieses Verhalts 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.

6.3 zeigt ausgewählte Ausgaben des für 25 Epochen lang trainierten Netzes, das die besten Ergebnisse in der IoU-Metrik liefert. Sie spiegeln die Bewertung in 6.1 mit hoher Genauigkeit bei großen und amorphen Objekten und niedriger bei kleineren,zählbaren. Besonders auffällig sind False Positives der Klasse Person, die das Modell 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

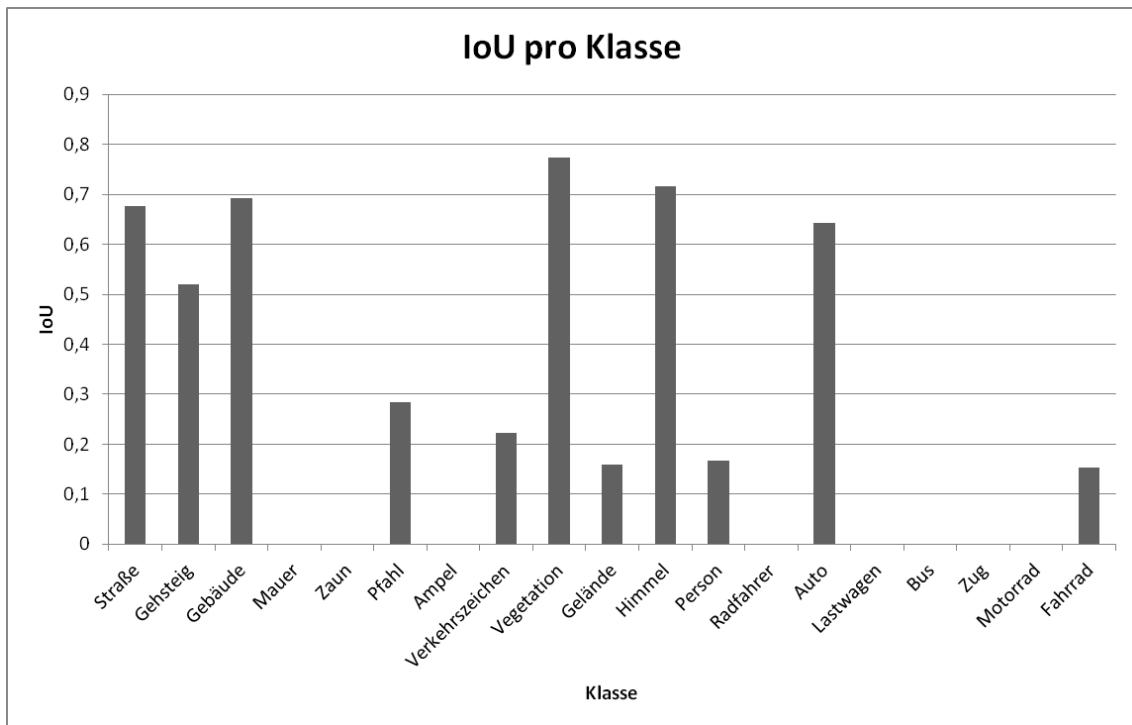


Abbildung 6.2: IoU für jede Klasse vom Test des für 25 Epochen trainierten Netzes

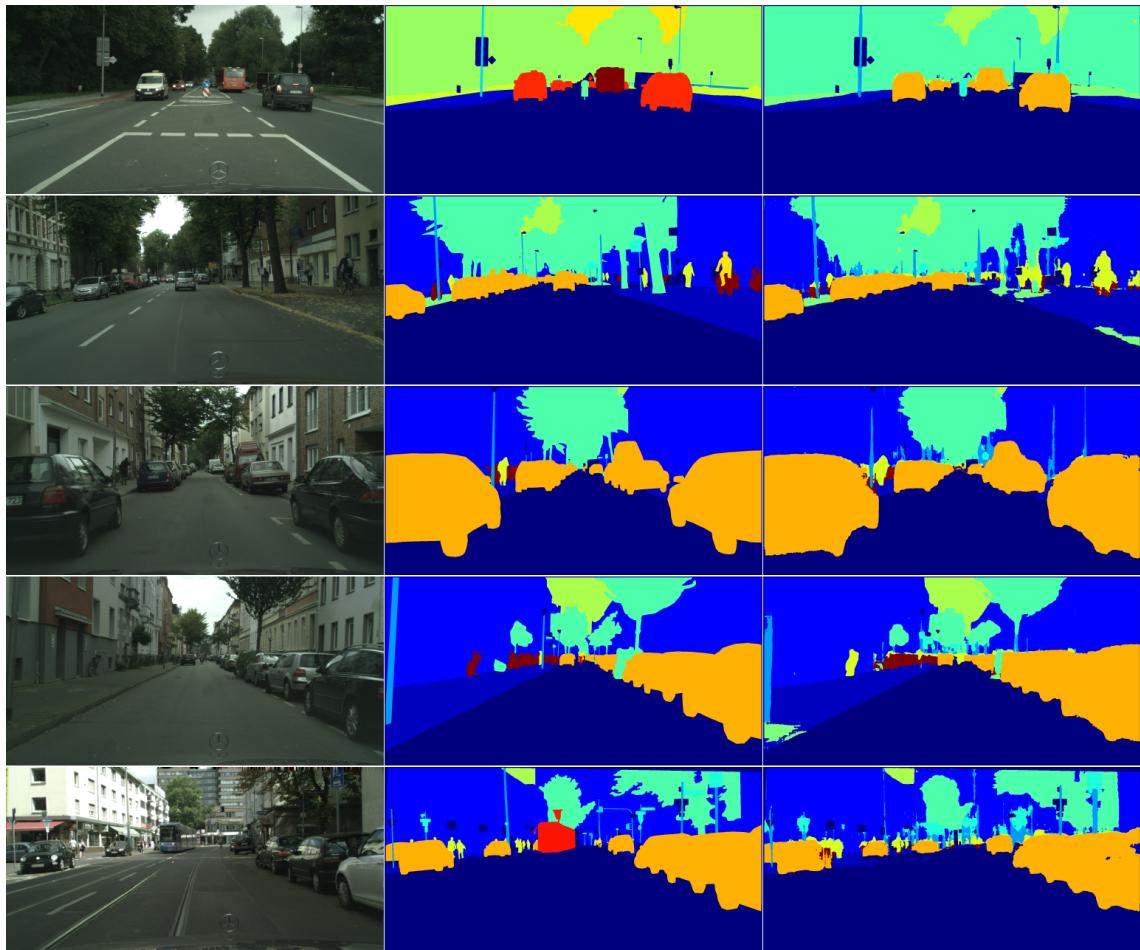


Abbildung 6.3: Beispiele für Ausgaben des Models (25 Epochen trainiert)
Von links nach rechts: Eingabebild, Ground Truth, Ausgabe von DeepLab

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.

6.2.2 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.

| | 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: Vergleich zwischen Xception65 und MobileNetV2 in IoU-Metrik und Verarbeitungszeit

Wie in Tabelle 6.2 und Abbildung 6.4 zu sehen ist, erzeugt das Model mit Xception65 durchwegs bessere Ergebnisse als das mit MobileNetV2, benötigt aber im Durchschnitt

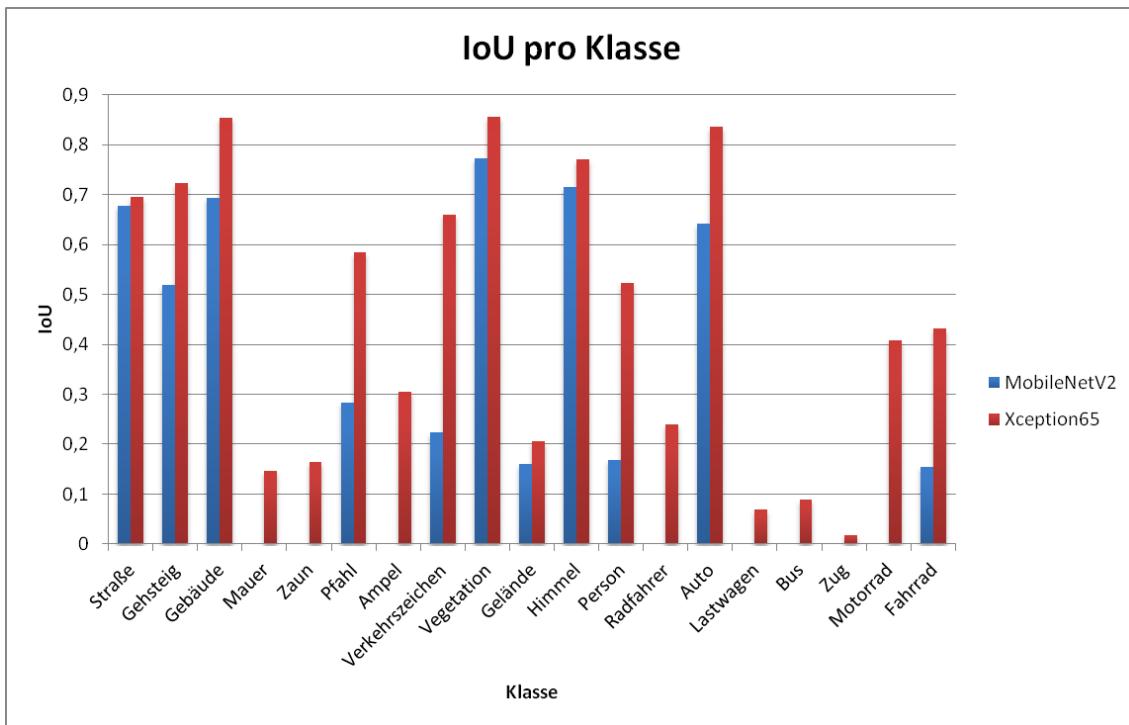


Abbildung 6.4: Vergleich Xception65 und MobileNetV2 in IoU-Metrik

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 Objekten auf wie Personen (311%) und Verkehrszeichen (295%).

Abbildung 6.5 zeigt beispielhafte Ergebnisse des Models aus dem Validierungs-Datensatz von Cityscapes.

6.3 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, ist es nicht möglich, die

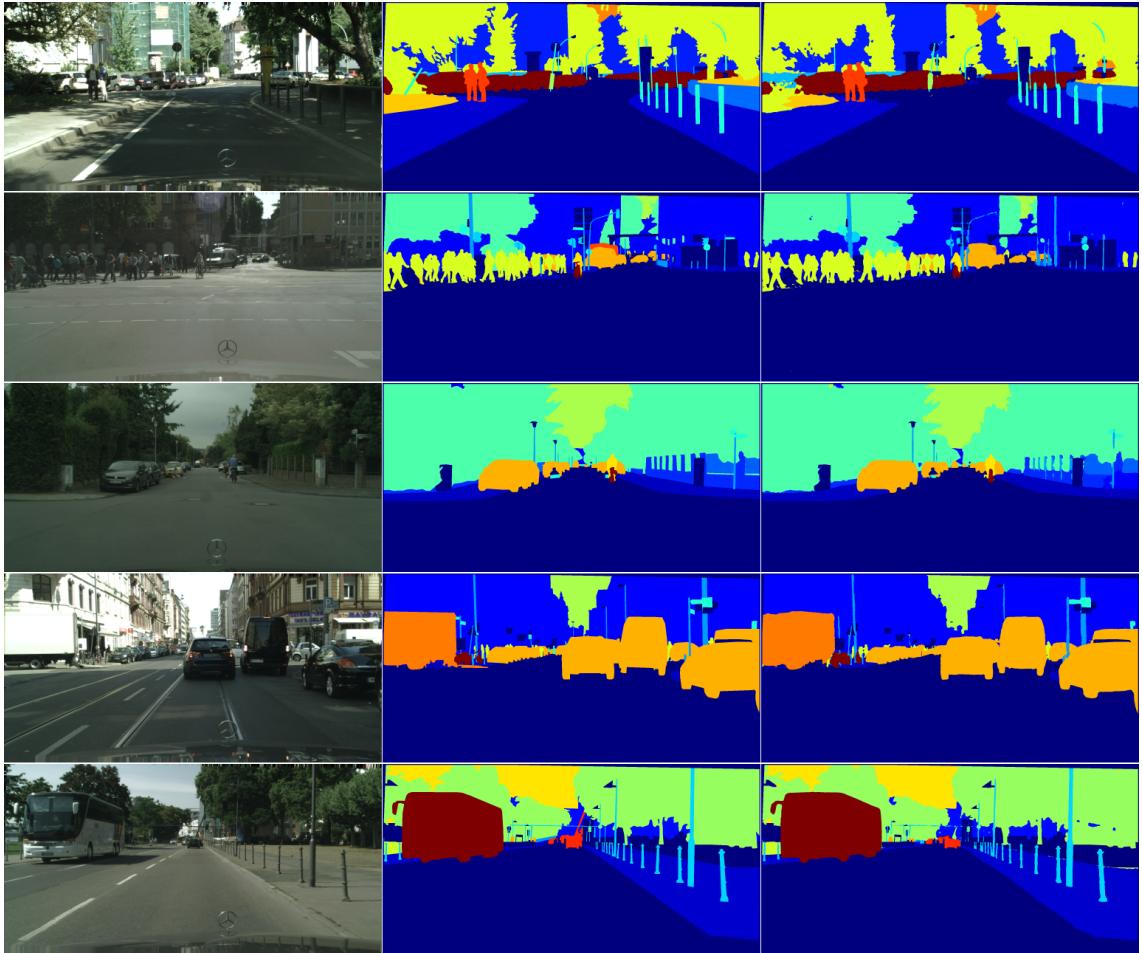


Abbildung 6.5: Beispiele für Ausgaben des Models

Von links nach rechts: Eingabebild, Ground Truth, Ausgabe von DeepLab

Ergebnisse mit der IoU-Metrik zu bewerten. Das Experiment begnügt sich deshalb mit einer empirischen Auswertung.

Abbildung 6.6 zeigt beispielhaft je ein Bild aus beiden Datensätzen und die Ausgaben 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 6.6 zeigt die Ausgabe eines Modells, 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 Verschlechte-

rung der Ergebnisse auf KITTI-Bildern und einer Verbesserung auf Cityscapes-Bildern.

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 klassifiziert wird. Es besteht kein erkennbarer Zusammenhang zwischen den Bildern, bei denen dieses Verhalten auftritt. Abbildung 6.7 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 Epoche mit allen verfügbaren Daten nachzutrainieren.

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

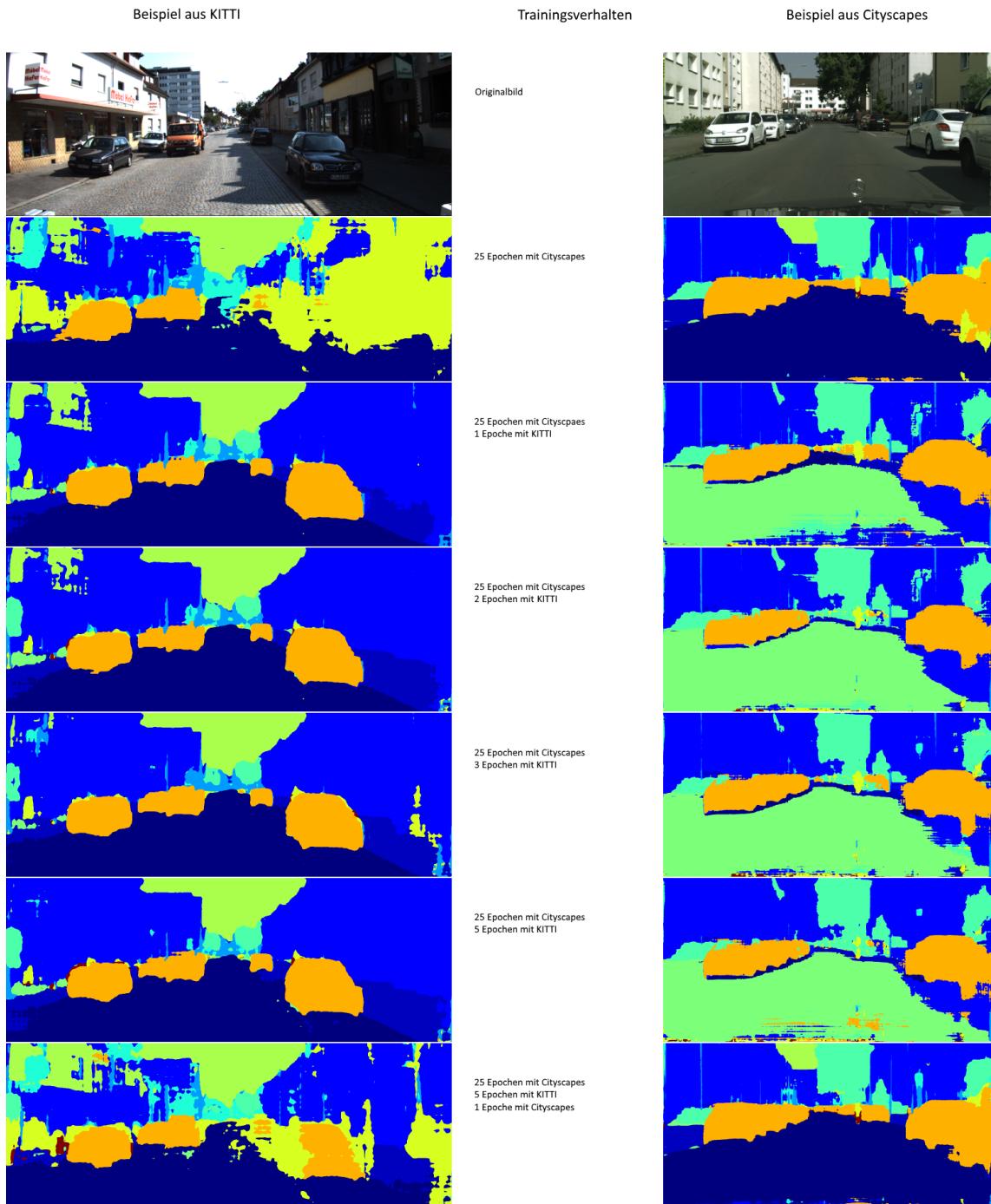


Abbildung 6.6: Beispiel für Ausgabe der verfeinerten Models

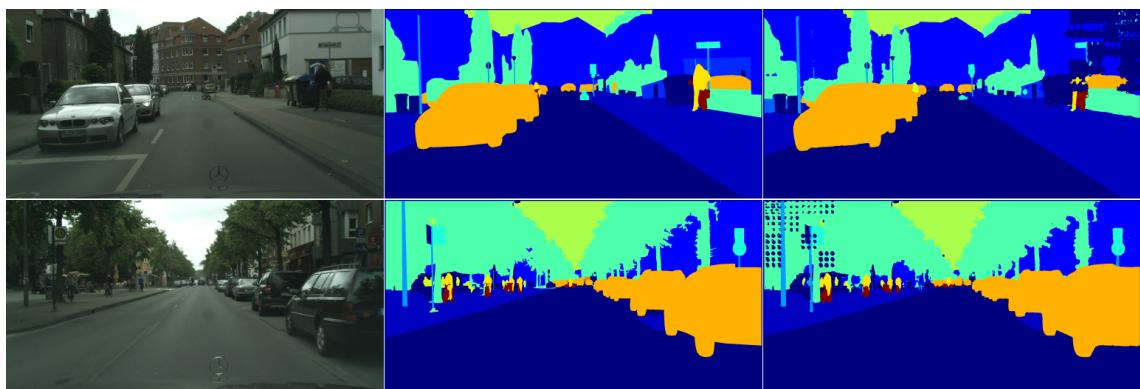


Abbildung 6.7: Beispiele für fehlerhafte Ausgaben des Models mit Xception65
Von links nach rechts: Eingabebild, Ground Truth, Ausgabe von DeepLab

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 bzw. 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 dienen die 5000 fein annotierten Bilder für Segmentierung von Cityscapes zu Training und Evaluierung des Netzes. Der KITTI-Datensatz liefert Laserscans bzw. 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. 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.

Literaturverzeichnis

- [COR⁺16] Cordts, Marius; Omran, Mohamed; Ramos, Sebastian u. a.: *The Cityscapes Dataset for Semantic Urban Scene Understanding*. In: *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.
- [crf] Sutton, Charles und McCallum, Andrew: *An Introduction to Conditional Random Fields*. URL: <https://arxiv.org/pdf/1011.4088.pdf> (zuletzt besucht am 02.09.2019).
- [dl1] Chen, Liang-Chieh; Papandreou, George; Kokkinos, Iasonas; Murphy, Kevin und Yuille, Alan L.: *Semantic Image Segmentation with Deep Convolutional Nets and Fully Connected CRFs*. URL: <https://arxiv.org/pdf/1412.7062.pdf> (zuletzt besucht am 24.07.2019).
- [dl2] Chen, Liang-Chieh; Papandreou, George; Kokkinos, Iasonas; Murphy, Kevin und Yuille, Alan L.: *DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs*. URL: <https://arxiv.org/pdf/1606.00915.pdf> (zuletzt besucht am 24.07.2019).
- [fpn] Lin, Tsung-Yi; Dollár, Piotr; Girshick, Ross B. u. a.: *Feature Pyramid Networks for Object Detection*. URL: <https://arxiv.org/pdf/1612.03144.pdf> (zuletzt besucht am 09.09.2019).
- [Fus06] Fusiello, Andrea: *Elements of Geometric Computer Vision*. 2006.
- [GBC16] Goodfellow, Ian; Bengio, Yoshua und Courville, Aaron: *Deep Learning*. <http://www.deeplearningbook.org>. MIT Press, 2016.
- [GW08] Gonzalez, Rafael C. und Woods, Richard E.: *Digital image processing*. Prentice Hall, Upper Saddle River, N.J., 2008.
- [HZ03] Hartley, Richard und Zisserman, Andrew: *Multiple View Geometry in Computer Vision*. 2. Auflage, Cambridge University Press, New York, NY, USA, 2003.
- [HZRS15] He, Kaiming; Zhang, Xiangyu; Ren, Shaoqing und Sun, Jian: *Deep Residual Learning for Image Recognition*. In: *CoRR*, Band abs/1512.03385, 2015.

- [kit] Geiger, Andreas; Lenz, Philip; Stiller, Christoph und Urtasun, Raquel: *Vision meets Robotics: The KITTI Dataset*. URL: <http://www.cvlibs.net/publications/Geiger2013IJRR.pdf> (zuletzt besucht am 14.08.2019).
- [LMP01] Lafferty, John D.; McCallum, Andrew und Pereira, Fernando C. N.: *Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data*. In: *Proceedings of the Eighteenth International Conference on Machine Learning*, Seiten 282–289. ICML '01, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001.
- [lov] Berman, Maxim; Triki, Amal Rannen und Blaschko, Matthew B.: *The Lovasz-Softmax loss: A tractable surrogate for the optimization of the 'intersection-over-union' measure in neural networks*. URL: https://zpzascal.net/cvpr2018/Berman_The_LovaSz-Softmax_Loss_CVPR_2018_paper.pdf (zuletzt besucht am 05.08.2019).
- [mn2] Sandler, Mark; Howard, Andrew; Zhu, Menglong; Zhmoginov, Andrey und Chen, Liang-Chieh: *MobileNetV2: Inverted Residuals and Linear Bottlenecks*. URL: <https://arxiv.org/pdf/1801.04381.pdf> (zuletzt besucht am 05.09.2019).
- [pnet] Qi, Charles R.; Su, Hao; Mo, Kaichun und Guibas, Leonidas J.: *PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation*. URL: <https://arxiv.org/pdf/1612.00593.pdf> (zuletzt besucht am 12.08.2019).
- [rcnn] He, Kaiming; Gkioxari, Georgia; Dollár, Piotr und Girshick, Ross: *Mask R-CNN*. URL: <https://arxiv.org/pdf/1703.06870.pdf> (zuletzt besucht am 12.08.2019).
- [ups] Xiong, Yuwen; Liao, Renjie; Zhao, Hengshuang u. a.: *UPSNet: A Unified Panoptic Segmentation Network*. URL: <https://arxiv.org/pdf/1901.03784.pdf> (zuletzt besucht am 24.07.2019).
- [xce] Chollet, François: *Xception: Deep Learning with Depthwise Separable Convolutions*. URL: <https://arxiv.org/pdf/1610.02357.pdf> (zuletzt besucht am 05.09.2019).