Bachelorarbeit

Daniel Leonid Riege

Segmentierung von Straßenmarkierungen durch maschinelles Lernen für die Miniaturautonomie

Fakultät Technik und Informatik
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

Daniel Leonid Riege

Segmentierung von Straßenmarkierungen durch maschinelles Lernen für die Miniaturautonomie

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Informatik Technischer Systeme*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Stephan Pareigis
Zweitgutachter: Prof. Dr. Tim Tiedemann

Eingereicht am: 25. Februar 2022

Daniel Leonid Riege

Thema der Arbeit

Segmentierung von Straßenmarkierungen durch maschinelles Lernen für die Miniaturautonomie

Stichworte

Bildsegmentierung, maschinelles Lernen, Miniaturautonomie, Linienerkennung

Kurzzusammenfassung

Erkennung von Straßenmarkierungen durch Bildsegmentierung mit neuronalen Netzen ist noch immer ein aktuelles Thema, da das Verhalten dieser Blackboxes schwer zu erfoschen ist. In dieser Arbeit werden bereits bekannte Ansätze verwendet, um neuronale Netze für solche Erkennungen zu trainieren und in der Miniaturautonmie zu untersuchen. Dabei werden UNet Architekturen durch verschieden vortrainierte CNNs abgewandelt und miteinander verglichen. Für die Anwendung in der Miniaturautonmie wurden neue Datensätze im Miniatur Wunderland Hamburg aufgenommen und annotiert. Die Netze wurden ebenfalls anhand ihrere Inferenzzeit verglichen, um geeignete Netze für Echtzeitsysteme zu finden. In einigen Experimenten konnte gezeigt werden, wie diese, durch überwachtes Lernen, trainierten neuronalen Netze sich in unbekannten Umgebungen verhalten. Dabei wurde entdeckt, dass manche Architekturen bei bestimmten Bildsequenzen ein Anomalieverhalten aufwiessen. Außerdem entstanden deutliche Unterschiede in der Genauigkeit bei einem erneuten Training der Netze, indem sich lediglich die Initialgewichte unterschieden haben.

Daniel Leonid Riege

Title of Thesis

Segmentation of road markings through machine learning for miniature autonomy

Keywords

image segmentation, machine learning, miniature autonomy, lane detection

Abstract

Road marking detection through image segmentation with neural nets is still an important topic, because the behaviour of such blackboxes is not an easy research. Known approaches are used in this thesis, to train neural nets for this kind of detection and to investigate them in the field of miniature autonomy. For this UNet architectures are modified and compared using different pre-trained CNNs. To be used in miniature autonomy, new datasets were recorded in the Miniatur Wunderland Hamburg and were annotated afterwards. The nets were compared by their inference time as well, to find the best suited one for a real time system. In some experiments it was shown, how these supervised trained neural nets performed in unkown environments. It was discovered that some architectures have anomalies in certain image sequences. Furthermore, clear differences in the accuracy of once more trained nets, with the only difference being the initial weights, was found.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungen

CM4 Compute Module 4.

CNN Convolutional Neural Net.

FC Fully Connected.

FN False Negative.

FP False Positive.

FPS Frames per Second.

HAW Hochschule für Angewandte Wissenschaften.

IoU Intersection over Union.

TN True Negative.

TP True Positive.

1 Einleitung

Autonomes Fahren beschäftigt sich mit der Aufgabe Fahrzeuge im Straßverkehr ohne menschliche Hilfe zu steuern. Dabei müssen die Fahrzeuge in der Lage sein die Umgebung richtig und zuverlässig wahrzunehmen. Dazu gehört auch die Erkennung von Straßenmarkierungen, da diese maßgeblich den Bewegungsraum des autonomen Fahrzeuges definieren. Anhand bestimmter Markierungen gibt es gewisse Verhaltensregeln, daher ist es wichtig, dass Straßenmarkierungen nicht nur erkannt, sondern auch klassifiziert werden.

Um solche Softwaresysteme für das autonome Fahren zu testen, erfordert es teure Hardware und eine sichere Umgebung. Aus diesem Grund ist der Einsatz von Simulationen üblich. Eine andere Form wären Miniaturmodelle, da diese kostengünstig sind und trotzdem eine realitätsnähere Umgebung bieten als Simulationen. An der Hochschule für Angewandte Wissenschaften (HAW) Hamburg wird an einer solchen Modellwelt geforscht. Die Miniaturanlage wurde im Maßstab 1:87 errichtet, da dies ein gängiger Maßstab ist und viele Modellbauprodukte vorhanden sind. Die HAW kooperiert in diesem Zusammenhang mit dem Miniatur Wunderland Hamburg, da diese eine 1499 Quadratmeter große Anlage, ebenfalls in 1:87, aufgebaut haben. Durch Fahrzeuge, die mit Hilfe eines Magnetstreifens in der Fahrbahn geleitet werden, ist es zudem möglich verschiedenste Szenarien zu testen.

Das Problem, welches sich durch einen solch kleinen Maßstab entwickelt, ist die nötige Hardware. Nils Schönherr entwickelt in seiner Arbeit "Kamera-basierte Minimalautonomie"ein solches Miniaturfahrzeug, welches mit einem ESP32 Microcontroller ausgestattet ist [15]. Die Arbeit "Hardwareplattformen für autonome Straßenfahrzeuge im Maßstab 1:87"von Markus Kasten beschäftigt sich ebenfalls mit der Entwicklung eines 1:87 Fahrzeuges [9]. In den Autos sind Recheneinheiten, wie das Raspberry Pi Compute Modul 4 mit zwei Google Coral TPUs, verbaut. So muss für Lösungen, wie das Erkennen von Straßenmarkierungen, auf ressourcensparende Verfahren zurückgegriffen werden.

Für das Erkennen von Straßenmarkierungen gibt es bereits einige Verfahren. Darunter auch der Einsatz von neuronalen Netzen, welche mit Hilfe des maschinellen Lernens trainiert werden können, um diese Straßenmarkierungen in komplexen Umgebungen zu erkennen. Allerdings sind neuronale Netze sogenannte Blackboxes, da man die Entscheidungen des Netzes nicht nachvollziehen kann. Daher sind verschiedene Test- und Evaluierungsmethoden ausschlaggebend, um neuronale Netze sicher in autonomen System verwenden zu können. Bei solchen Testmethoden sind vor allem unkown unkowns interessant, da diese Fälle beschreiben von denen wir noch nicht wissen, ob wir sie wissen. Für solche Fragestellungen soll die Miniaturautonomie dienen.

1.1 Zielsetzung

Im Rahmen dieser Arbeit werden Architekturen neuronaler Netze für die Bildsegmentierung vorgestellt. Diese neuronalen Netze sollen anhand verschiedener Verfahren trainiert und getestet werden.

Es gibt im Bereich des maschinellen Lernens für Bildsegmentierung viele verschiedene Architekturen, die verwendet werden können. Da nicht alle im Rahmen dieser Arbeit getestet werden können, wird sich auf UNet Architekturen beschränkt [13]. Es werden dennoch verschiedene UNet Architekturen vorgestellt, welche jeweils andere Verfahren nutzen, um diese miteinander vergleichen zu können. Es wird beim Entwurf immer darauf geachtet, dass diese neuronalen Netze Anwendung in der Miniaturautonomie Verwendung finden sollen und somit die nötige Speichergröße und Anzahl der Operationen beachtet wird.

Um solche neuronalen Netze zu trainieren, gibt es bereits einige mögliche Fehlerfunktionen. In den Experimenten sollen alle Architekturen mit den ausgewählten Fehlerfunktion gegenübergestellt und verglichen werden. Zudem sollen die neuronalen Netze in Richtung ihrer Genauigkeit optimiert werden. Auf Optimierungen durch bestimmte Aktivierungsfunktionen etc. wird aufgrund des zeitlichen Rahmens verzichtet.

Um die Genauigkeit bestimmen zu können werden verschiedene Metriken verwendet. Dabei werden die vom Netz generierten Bildsegmentierungen mit den tatsächlichen Bildsegmentierungen verglichen. Außerdem sollen die Netze anhand ihrer Zuverlässigkeit bewertet werden. Verschiedene Verfahren werden hierfür vorgestellt und durchgeführt.

Für die Eignung in der Miniaturautonomie soll außerdem die Inferenzzeit der einzelnen neuronalen Netze untersucht werden. Als Referenz soll hierbei das CM4 Fahrzeug von Markus Kasten dienen [9], welches mit Beschleunigern für neuronale Netze ausgestatt ist.

1.2 Forschungsstand

Neben der Bildsegmentierungen gibt es auch andere Verfahren, um Straßenmarkierungen zu erkennen. So können in nicht komplexen Umgebungen auch algorithmisch gefundene Merkmale dazu dienen, Polynome für Straßenmarkierungen zu erzeugen [1, 8]. Allerdings finden solche Verfahren eher Anwendung auf Autobahnen, bei denen es keine kreuzende oder stark gekrümmte Linien gibt. Auch werden Straßenmarkierungen wie Fußgängerüberwege ignoriert.

Verfahren, die 2017 in der Tusimple Lane Detection Challenge gute Ergebnisse erzielten, greifen auf neuronale Netze zurück [11]. Diese Verfahren führen eine Instanzsegmentierung durch, indem die Linien mit Klassen, wie links-linke Linie, linke Linie, rechte Linie etc. definiert werden. Somit kann allerdings ein anschließendes Clustern vermieden werden [10]. Eine andere Methode, die auf die selbe Klassendefinition setzt, macht die Auflösung der Segmentierungen gröber, um schnellere Inferenzzeiten zu garantieren [12]. In einer anderen Arbeit werden mehrere aufeinanderfolgende Frames für die Segmentierung verwendet, was zu einer robusteren Linienerkennung führt [20]. Durch diese Instanzsegmentierung wird jedoch nicht nach Art der Straßenlinie klassifiziert. Eine Unterscheidung zwischen durchgezogener oder gestrichelter Linie findet nicht statt.

Für eine semantische Segmentierung gibt es Architekturen wie UNet, welches aus dem Bereich der medizinischen Bilder stammt [13]. An dieser Architektur wird in dieser Arbeit angesetzt, indem verschiedene vortrainierte Netze als Encoder verwendet werden [4, 5, 17, 16]. In einer anderen Arbeit wird ein ähnlicher Ansatz verfolgt. Dort wird, wie bei UNet, eine Encoder-Decoder Architektur aufgesetzt, welche ebenfalls ein vortrainiertes Netz als Encoder verwendet [2]. Allerdings unterscheidet sich diese vorgestellte SegNet Architektur von denen, die in dieser Arbeit vorgestellt werden, dadurch, dass die Skip Connections, wie in UNet beschrieben, beibehalten werden.

2 Grundlagen

In diesem Kapitel wird kurz erläutert, was Segmentierungsnetze sind und wie sie sich von herkömmlichen Klassifikationsnetzen unterscheiden. Zudem wird die Nomenklatur für die verwendeten Metriken und Fehlerfunktionen festgelegt.

2.1 Segmentierungsnetze

Bei einer Segmentierungen geht es darum, dass einzelne Pixel in einem Bild nach inhaltlich zusammenhängenden Regionen zusammengefasst werden. Segmentierungsnetze sind demnach neuronale Netze, welche eine pixel-weise Klassifikation durchführen. Dabei kommen, wie bei Klassifikationsnetzen auch üblich, convolutional neural nets (CNN) zum Einsatz. Die Eingabe erfolgt durch ein Bild und die Ausgabe ist ebenfalls ein Bild bzw. eine Reihe von Bildern mit meist gleicher Auflösung. Diese Ausgaben sind Masken. Die Anzahl der Masken hängt mit der Anzahl der möglichen Klassen zusammen. Für jede Klasse gibt es somit eine Maske. Eine Maske beschreibt für jeden einzelnen Pixel eine Wahrscheinlichkeit, dass dieser Pixel zu der jeweiligen Klasse gehört.

Somit ergibt sich ein Zusammenhang zu Klassifikationsnetzen. Bei einer Klassifikation dient als Eingabe ebenfalls ein Bild. Allerdings ist die Ausgabe eine diskrete Verteilung (Bei einer 1 aus n Klassifikation). Diese Verteilung ordnet dem ganzen Bild eine Klasse zu. Bei der Segmentierung handelt es sich im groben auch um eine solche Klassifikation, allerdings auf Pixelebene. Es wird für jeden Pixel eine Klassifikation durchgeführt. Somit gibt es auch für jeden Pixel eine eigene diskrete Verteilung. Ist das Eingangsbild also 10×10 Pixel groß und bei der Segmentierung soll in 4 Klassen unterschieden werden, so gibt es 100 diskrete Verteilungen mit 4 Wahrscheinlichkeitsvariablen.

Diese diskreten Verteilungen kann man so anordnen, dass daraus wieder ein Bild entsteht. So wird für jede Klasse, für die es in dem Beispiel 100 Wahrscheinlichkeitsvariablen gibt, eine Maske erzeugt. Die Auflösung entspricht dem Eingangsbild. Demnach werden die

Wahrscheinlichkeitsvariablen in gleicher Relation angeordnet und es entsteht ein Graustufenbild, bei dem jeder Pixel die Wahrscheinlichkeit angibt, dass dieser Pixel zu dieser Klasse gehört. Da es 4 Wahrscheinlichkeitsvariablen pro Verteilung gibt, wird es 4 Graustufenbilder geben. Also insgesamt wäre die Ausgabe ein $10 \times 10 \times 4$ Bild. Jeder Kanal im Bild spiegelt eine Klasse wieder.

Deshalb können bei Segmentierungsnetzen die gleichen Techniken verwendet werden, wie bei Klassifikationen. Also zum Beispiel die Verwendung von Softmax in der Ausgabeschicht, um eine Wahrscheinlichkeitsverteilung, welche in Summe 1 ergibt, zu generieren. Aber auch die in Klassifikationsnetzen mit CNN üblichen Transfer Learning Modelle können verwendet werden. Der Unterschied besteht nur darin, dass es weitaus mehr Verteilungen gibt, als bei einer einfachen Klassifikation. Daher werden für den Klassifikationsteil ebenfalls oft Faltungslayer verwendet, da diese die Anzahl der nötigen Gewichte reduzieren können.

In den Segmentierungsnetzen gibt es zudem auch noch Unterschiede. So gibt es die semantische Segmentierung und die Instanzsegmentierung. In dieser Arbeit wird eine semantische Segmentierung angewendet. Das bedeutet, wie oben beschrieben, dass jeder Pixel einer Klasse zugeordnet wird. Die Instanzsegmentierung funktioniert so, dass jeder Erkennung eine Instanz zugeordnet wird. Gibt es in einem Bild also zum Beispiel mehrere Autos, wird jedes individuelle Auto einer eigenen Instanz zugeordnet.

Zusammengefasst kann man also sagen, dass neuroale Netze für die semantische Segmentierung den Klassifikationsnetzen ähnlich sind, mit dem Unterschied, dass es mehrere diskrete Verteilungen gibt. Aus diesen diskreten Verteilungen können wiederum Bilder erzeugt werden, welche die Informationen aus der Klassifikation beinhalten und somit jeden Pixel im Eingangsbild einer Klasse zuordnen.

2.2 Metriken

Als Grundlage für alle vorgestellten Metriken dienen die Klassifikationsleistungen, wie true positive (TP), false positive (FP), true negative (TN) und false negative (FN). Diese werden bei Klassifikation verwendet, indem die echten Daten mit dem vom Klassifikator erzeugten Daten verglichen werden. Diese Klassifikationsleistungen geben die

Anzahl der Klassifikation an, die für die jeweilige Leistung zutreffen. Bei einer Segmentierung sagt TP zum Beispiel aus, wie viele Klassifikationen der tatsächlichen Straßenmarkierungen im Bild auch als Straßenmarkierungen segmentiert wurden. Umgekehrt sagt FP dann aus, wie viele Klassifikationen der erkannten Straßenmarkierungen tatsächlich keine Straßenmarkierung sind. Die Werte der Metriken liegen alle im Interval von $[0, 1]$.

Insgesamt wird immer eine Abweichung vom tatsächlichem Wert zu vorhergesamtem Wert berechnet.

2.2.1 Precision

Die Precision setzt die richtig klassifizierten Pixel ins Verhältnis zu allen klassifizierten Pixel. Somit gibt die Precision an, wie viele der vom Netz klassifizierten Pixel relevant sind. Umgekehrt, wie viele unbrauchbare Pixel wurden klassifiziert.

$$precision = \frac{TP}{TP + FP} \tag{2.1}$$

2.2.2 Recall

Der Recall setzt die richtig klassifizierten Pixel ins Verhältnis zu allen möglich richtigen Pixeln. Somit gibt der Recall an, wie viele der möglich richtigen Pixel gefunden wurden.

$$recall = \frac{TP}{TP + FN} \tag{2.2}$$

2.2.3 F1 Score

Der F_1 Score, auch bekannt als Dice Koeffizient, ist das harmonische Mittel von Precision und Recall. Er misst in Richtung des Durchschnitts von Precision und Recall.

$$F_1 = \frac{TP}{\frac{1}{2}(FP + FN)} = \frac{2(precision \cdot recall)}{precision + recall} \tag{2.3}$$

2.2.4 IoU Score

Der Intersection over Union (IoU) Score misst das Verhätnis aus der Region, welche richtig klassifiziert wurde (TP/Intersection), zu der Region, bei der alle in Klassen auftreffende Pixel befinden (Union). Zu den in Klassen auftreffenden Pixel gehören sowohl die aus dem Netz klassifiziert wurden, als auch die tatsächlich klassifiziert sind. Der IoU Score misst im Vergleich zum F_1 Score eher in Richtung des worst case.

$$IoU = \frac{TP}{TP + FP + TN} = \frac{precision \cdot recall}{precision + recall - precision \cdot recall} \tag{2.4}$$

Sowohl F_1 als auch IoU Score sind sich einig, wenn ein Klassifikator schlecht funktioniert. Allerdings tendiert der IoU Score immer zu schlechteren Werten, wenn dieser über mehrere Inferenzen gemittelt wird, da dieser schlechte Klassfikationen mehr bestraft.

2.3 Fehlerfunktionen

Fehlerfunktionen dienen dazu, ähnlich wie Metriken, eine Abweichung zwischen tatsächlichem Wert zu vorhergesamtem Wert zu berechnen. Diese Abweichung oder auch Fehler genannt muss allerdings differenzierbar sein, um sie für das Training verwenden zu können. Es gibt bereits einige Fehlerfunktion, die Vorteile gegenüber anderen bieten und für gewisse Problematiken geeignet sind [7, 18]. Im folgenden werden solche Fehlerfunktion vorgestellt, welche in dieser Arbeit Verwendung finden.

2.3.1 Kategorische Kreuzentropie

Die kategorische Kreuzentropie ist eine Abwandlung der binären Kreuzentropie. Bei der binären Kreuzentropie geht man von zwei Klassen aus. In dieser Arbeit werden allerdings mehr als zwei Klassen benötigt, weshalb die kategorische Kreuzentropie als Fehlerfunktion vorgestellt wird. Dabei wird für jede Klasse $c \in C$ das Fehlermaß berechnet. Die tatsächlichen Werte, welche zum Beispiel One-Hot Encoded sein können, sind mit y_c repräsentiert. Dies ist ein Vektor mit allen tatsächlichen Pixelwerten für die jeweilige Klasse c. $\hat{y}_c$ stammt aus den diskreten Wahrscheinlichkeitsverteilungen in der Ausgabe der Segmentierung. Dies ist also ebenfalls ein Vektor mit allen vorhergesagten Wahrscheinlichkeiten für die Klasse c.

Da jede Klasse unabhängig voneinander betrachtet wird, werden die berechneten Fehler über alle Klassen c summiert.

$$CE = -\sum_{c}^{C} y_c \cdot \log(\hat{y}_c) \tag{2.5}$$

2.3.2 Gewichtete Kategorische Kreuzentropie

In der kategorischen Kreuzentropie werden alle Klassen $c \in C$ gleich stark bewertet. Wenn in den Datensätze allerdings eine Klasse unterrepräsentiert ist, kommt es zu einer Klassenimbalance. Dadurch kann der Fehler in der unterrepräsentierten Klasse keine große Auswirkung im gesamten Fehler mehr bewirken. Damit alle Klassen gleich in den Fehler einfließen, können bestimmte Klasse anders gewichtet werden. So gibt es für jede Klasse $c \in C$ ein konstantes Gewicht w_c.

Diese Gewichte können anhand der Verteilungen im Datensatz bestimmt werden und sind normalisiert.

$$WCE = -\sum_{c}^{C} w_c \cdot y_c \cdot \log(\hat{y}_c) \tag{2.6}$$

2.3.3 Dice Fehlerfunktion

Der Dice Koeffizient, oder auch F1 Score (Abschnitt 2.2.3), ist eine Metrik, um zum Beispiel Bildsegmentierungen zu bewerten. Er kann allerdings auch als Fehlerfunktion verwendet werden. Im Gegensatz zu der Kreuzentropie beachtet der Dice Koeffizient nicht den Hintergrund, da hier nur true positives verwendet werden und keine true negatives. Somit wirkt die Dice Fehlerfunktion indirekt gegen eine Klassenimbalance. Die gewichtete Kreuzentropie berechnet innerhalb jeder Klasse auch den Hintergrund in Form von true negatives mit ein.

Auch hier wird der Dice Koeffizient für jede Klasse seperat ermittelt und am Ende summiert. Ob Am Ende gemittelt oder summiert wird, spielt für die Fehlerfunktion keine Rolle, da der Gradient entscheident ist.

$$D = \sum_{c}^{C} 1 - \frac{2TP_c}{2TP_c + FN_c + FP_c} \tag{2.7}$$

2.3.4 Tversky Fehlerfunktion

Der Tversky Index ist eine Generalisierung des Dice Koeffizienten. Der Untrschied besteht in der Gewichtung des FN und FP. In der Regel werden α und β so gesetzt, dass $\alpha + \beta = 1$. Mit Hilfe dieser Gewichtung kann man entweder die false negatives, also die Pixel, die fälschlicherweise dem Hintergrund zugeordent wurden, höher gewichten oder die false postives, also die Pixel, die fälschlicherweise der Klasse c zugeordent wurden.

$$T = \sum_{c}^{C} 1 - \frac{TP_c}{TP_c + \alpha FN_c + \beta FP_c} \tag{2.8}$$

2.3.5 Focal Tversky Fehlerfunktion

Die Focal Tversky Fehlerfunktion ist eine Generalisierung der Tversky Fehlerfunktion, indem eine Nichtlinearität eingeführt wird, welche durch γ gesteuert werden kann. So kann die Fehlerkurve über kleiner werdendem Fehler beeinflusst werden.

$$FT = \sum_{c}^{C} (1 - \frac{TP_c}{TP_c + \alpha FN_c + \beta FP_c})^{\gamma} \tag{2.9}$$

3 Vorbereitung

In diesem Kapitel geht es um die Vorbereitungen die getroffen werden müssen, um später verschiedene Experimentene mit den neuronalen Netzen durchführen zu können. Dies beinhaltet die benötigten Datensätzen, welche gelabelt und ausgewertet müssen, um überhaupt ein überwachtes Lernen durchführen zu können. Zudem werden auch die Architekturen präsentiert, welche in dieser Arbeit verwendet werden.

3.1 Erhebung der Trainingsdaten

3.1.1 vorhandene Datensätze

Das Ziel ist es, dass das neuronale Netz direkt mit Kamerabildern arbeitet, auf denen keine Vorverarbeitung nötig ist. Diese Kamerabilder sollten idealerweise einen größtmöglichen Teil der Bodenmarkierungen im sichtbaren Bereich haben. Somit sollte die Kamera zentral vorne am Fahrzeug montiert sein und über einen gewissen Weitwinkel verfügen. Für reale Szenarien gibt es bereits einige Datensätze. Die meisten öffentlich zugänglichen Datensätze beschränken sich allerdings oft auf Autobahnen. Dazu zählt zum Beispiel der LLAMAS Datensatz [3], welcher jede Fahrbahnlinie einzeln klassifiziert. Die selbe Klassifizierung findet man im CULane Datensatz, der erstmals im Paper "Spatial As Deep: Spatial CNN for Traffic Scene Understanding"[11] auftaucht. Die dort aufgenommenen Bilder stammen zwar aus urbanen Umgebungen, beinhalten allerdings nur Fahrbahnlinien der aktuellen Fahrtrichtung. Datensätze, wie der Cityscapes, segmentieren hingegen nur die gesamte Fahrbahn als solche, nicht jedoch einzelne Fahrbahnmarkierungen. Angestrebt wird eine Segmentierung der Straßenmarkierungen in verschiedensten Umgebungen. Neben Autobahnen oder Landstraßen vor allem auch urbane Umgebungen mit komplexen Kreuzungen und einer vielzahl verschiedener Bodenmarkierungen. Somit sind die vorgestellten Datensätze für das Anwendungsbiet der Miniaturautonomie nicht gut geeignet.

Um eignene Datensätze zu erzeugen, wurde das CM4 Fahrzeug [9] verwendet. Die Kamera dieses Fahrzeuges enthält auch den benötigten Weitwinkel. Mit einem Winkel von 120 Grad in der horizontalen können auch größere Kreuzungsbereiche eingefangen werden. Durch ein Wi-Fi Modul lässt sich das Auto leicht mit einem Joystick fernsteuern, wodurch

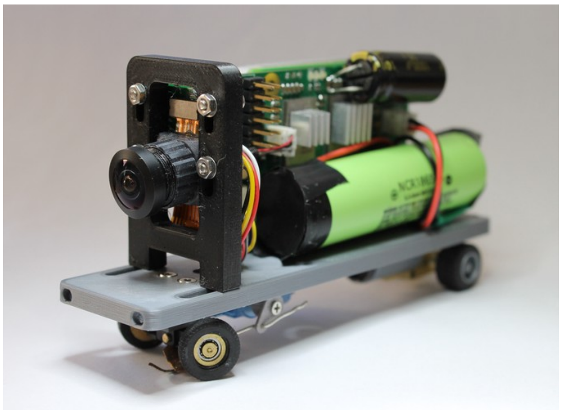


Abbildung 3.1: CM4 Fahrzeug mit Servolenkung. Die Frontkamera verfügt über einen Weitwinkel mit 120 Grad in der horizontalen. Quelle: Markus Kasten

man das Fahrzeug frei in der Umgebung bewegen kan. Eine weitere Möglichkeit wäre gewesen, das Fahrzeug mittels dem Faller Car System in der Fahrspur zu halten, allerdings ist man dann auf Bilder innerhalb der Fahrspur begrenzt. Ziel ist es aber, dass das neuronale Netz die Markierungen aus verschiedenen Blickwinkeln segmentieren kann. So ist es sinnvoll, wenn die Trainingsbilder ebenfalls aus verschiedenen Blickwinkel und Perspektiven gemacht wurden. Das Fahrzeug sich also auch mal ausserhalb der Fahrbahn befindet oder quer auf der Straße steht.

3.1.2 Klassendefinition

Auf Grund der Tatsache, dass die Datensätze aus verschiedenen Perspektiven aufgenommen wurden, können die Straßenmarkierungen nicht nach der Position klassifiziert werden. Also so, wie es z.B. im LLAMAS Datensatz [3] geschieht, wo es die Klassen l_1, l_0, r_0, r_1 gibt, wobei das l für links und r für rechts steht. Zusätzlich würde diese Klassendefinition in urbanen Umgebungen zu Problemen führen, da kreuzende oder zusammenführende Linien nicht mehr eindeutig zugeordnet werden können.

Es brauch also neben neuen Datensätzen auch eine neue Klassendefinition. In Anlage 2 zu § 41 Abs. 1, Abschnitt 9 und Anlage 3 zu § 42 Abs. 2 StVO sind alle auf deutschen

Straßen zulässigen Bodenmarkierungen definiert. An diesen wurde sich hierfür orientiert. Es gibt allerdings auch einige Abweichungen, die unter Berücksichtiging, dass ein neuronales Netz diese korrekt klassifizieren muss und die Datensätze in ihrer Anzahl von Bildern beschränkt sind, getroffen wurden. Die Tabelle 3.1 zeigt, wie gewisse Markierungen zusammengefasst werden. Leitlinien und Wartelinien zum Beispiel sind in ihrer

<table>
<tr><th>Fahrbahnmarkierungen nach StVO</th><th>eigene Klassifizierung</th></tr>
<tr><td>Leitlinie</td><td rowspan="2">gestrichelte Linie</td></tr>
<tr><td>Wartelinie</td></tr>
<tr><td rowspan="2">Fahrstreifenbegrenzung</td><td>Fahrbahnaußenkante</td></tr>
<tr><td rowspan="3">durchgezogene Linie</td></tr>
<tr><td>Parkflächenmarkierung</td></tr>
<tr><td>Grenzmarkierung für Halt- oder Parkverbote</td></tr>
<tr><td>Haltlinie</td><td>Haltlinie</td></tr>
<tr><td>Fußgängerüberweg</td><td>Fußgängerüberweg</td></tr>
<tr><td>Sperrfläche</td><td>Sperrfläche</td></tr>
</table>

Tabelle 3.1: Abbildung von Markierungen nach StVO auf eigene Klassen

Struktur gleich. Es sind gestrichelte Linien. Sie unterscheiden sich zwar im Verhältnis von Strich zu Lücke, so ist die Wartelinie nach Teil 1: Abmessungen und geometrische Anordnung von Markierungszeichen (RMS-1) im Verhältnis 2:1 und die Leitlinie 1:2 oder 1:1, jedoch ist die Grundstruktur einer gestrichelten Linie bei beiden gegeben. Vernachlässigt man das Verhätnis zwischen Strich und Lücke, da das Verhältnis beider Linien in den verwendeten Umgebungen gleich ist, so können diese beiden Arten nur im Kontext unterschieden werden. Das heißt mit dem Hintergrundwissen, wo man sich gerade befindet und wo sich die gestrichelte Linie befindet. Wartelinien erwartet man z.B. immer an Kreuzungen von Fahrbahnen. Abbildung 3.2 zeigt, wie sich die beiden Linien optisch nicht unterscheiden.

Aus diesem Grund findet man die selbe Zusammenfassung auch bei durchgezogenen Linien. Eine Ausnahme bilden Haltlinien, da diese deutlich dicker sind und in ihrer Struktur unterschieden werden können. Abbildung 3.3 verdeutlicht dies. Eine weitere Besonderheit kommt mit den Fahrbahnaußenkanten. Diese bilden keine Art in der StVO, sind aber für das spätere Navigieren auf den segmentierten Bildern essenziell. Sie stecken den Raum ab, auf dem sich ein Fahrzeug bewegen kann. Teilweise sind die Fahrbahnaußenkanten durch durchgezogene Linien gekennzeichnet, was einer Fahrstreifenbegrenzung entspreche, oft gibt es allerdings keine Markierung. Die Fahrbahnaußenkante ist in diesem Fall durch den Übergang des Straßenbelags auf einen anderen Belag, wie zum Beispiel Gras

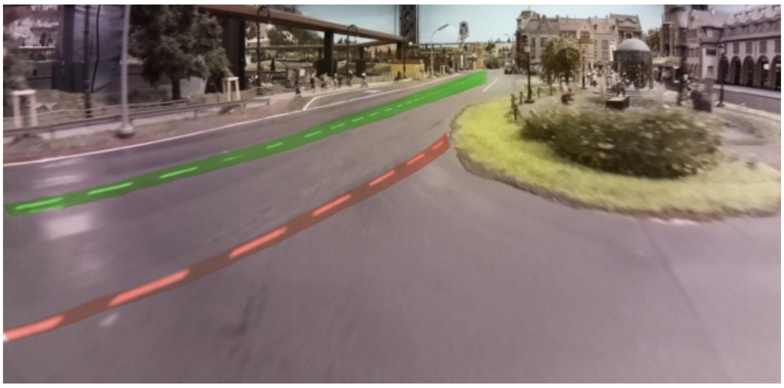


Abbildung 3.2: Leitlinie farblich grün markiert. Wartelinie mit gleichem Verhältnis Strich zu Lücke, wie Leitlinie, farblich orange markiert.

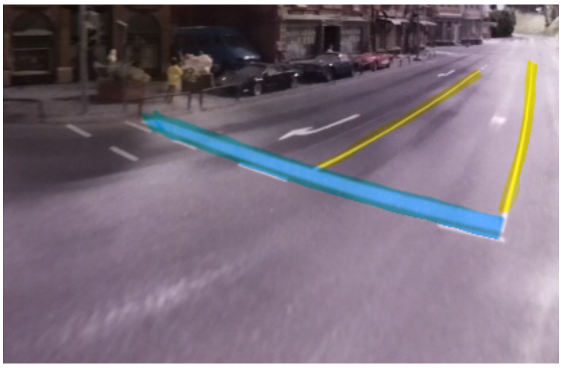


Abbildung 3.3: Fahrstreifenbegrenzung farblich gelb markiert. Haltlinie farblich blau markiert. Die Haltlinie ist deutlich dicker als die Fahrstreifenbegrenzung.

oder eine Bordsteinkante. Wenn sich also eine Fahrstreifenbegrenzung am Straßenbelagsrand befindet, wird sie hier als Fahrbahnaußenkante klassifiziert. Befindet sich auf beiden Seiten Straßenbelag, so wird sie durchgezogene Linie interpretiert. Der Grad, ab wann die Fahrstreifenbegrenzung nah genug am Straßenrand ist, ist dabei allerdings nicht genau definiert. Für das Netz ergibt sich so jedoch keine Klassifizierung anhand des Kontextes, denn Straßenränder sind in der Regel farblich abgesetzt.

3.1.3 Annotation

Mit Hilfe der eigenen Klassifizierungen werden somit die Kamerabilder annotiert. Das Ergebnis, wie bei der Segmenterung üblich, ist pro Klasse ein binäres Bild, welches die zugehörigen Pixel zur Klasse darstellt. Diese erzeugten Masken bilden die Fahrbahn-

(a) Fahrbahnaußenkante durch Asphaltende

(b) Fahrbahnaußenkante mit Fahrstreifenbegrenzung

Abbildung 3.4: Fahrbahnaußenkanten durch verschiedene visuelle Eigenschaften gegenübergestellt.

eigene Klassifizierung	Kurzbezeichnung	Farbe
gestrichelte Linie	guide_lane	grün
Fahrbahnaußenkante	road_edge	rot
durchgezogene Linie	solid_lane	gelb
Haltlinie	hold_line	blau
Fußgängerüberweg	zebra	hellblau
Sperrfläche	middle_curb	orange

Tabelle 3.2: Auflistung aller definierten Klassen. Die Kurzbezeichnungen werden intern verwendet, wenn Grafiken oder ähnliches generiert werden, da die vollen Klassennamen zu lang wären. Die Farbe wird für spätere Beispielsegmentierungen relevant.

markierungen allerdings nicht pixelgenau ab. Ansonsten würde dies bedeuten, dass die Lücken in gestrichelten Linien nicht zugeordnet werden, da nur die Striche segmentiert werden würden. Das Ge- oder Verbot einer Straßenmarkierung gilt allerdings auch für dessen Lücken. So kann ein nachträgliches Interpolieren vermieden werden. Treten in einem Bild mehrere Instanzen derselben Klasse auf, wird zwischen diesen nicht unterschieden. Es ergibt sich ingesamt also eine semantische Segmentierung, welche allerdings nicht pixelgenau ist, sondern über eine extra erzeugte Ungenauigkeit verfügt. Repräsentiert werden die Linien durch einen Polygonzug, also die Vereinigung von Verbindungsstrecken einer Folge von Punkten. Jede Instanz einer Straßenmarkierung erhält einen eigenen Polygonzug. Bei Flächen, wie einer Sperrfläche oder Fußgängerüberweg, werden die Instanzen durch Polygone repräsentiert. Durch das Arbeiten mit Folgen von Punkten ist es möglich die Dicke der Polygonzüge nachträglich zu varieren. Allerdings wird auch dabei eine Ungenauigkeit der Segmenterung in Kauf genommen, da die Polygonzüge nicht immer genau die weißen Bodenmarkierungen treffen. Gerade, wenn die Dicke der Polygonzüge nicht der tatsächlichen Dicke der Markierungen entspricht.

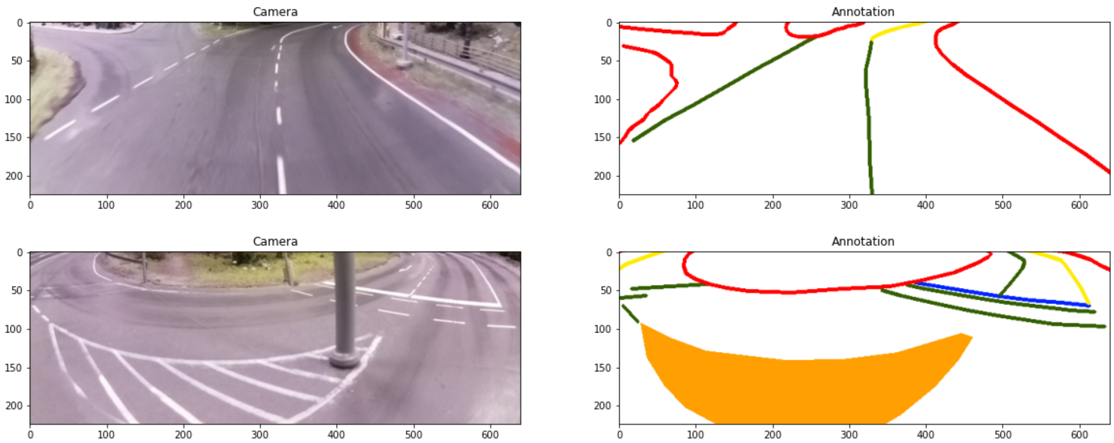


Abbildung 3.5: Annotation von zwei Kamerabildern mit einer Polygonzugdicke von 4px. Die Kamerabilder sind bereits gecropped, sodass nur die untere Hälfte des Bildes verwendet wird. Die Farbe richtet sich nach Tabelle 3.2.

Wie in Abbildung 3.5 zu sehen, sind die Bilder nicht in voller Auflösung im Datensatz abgebildet. In Abbildung 3.4 sind die Kamerabilder in originaler Auflösung. Die obere Hälfte des Bildes ist allerdings für die Straßenmarkierungen irrelevant. Dies liegt an der Ausrichtung der Kamera auf dem Fahrzeug. Das Bild wird also so ausgeschnitten, dass die obere Hälfte weggelassen wird. So sind nur relevante Informationen im Datensatz enthalten und der Speicherbedarf kann minimiert werden. Auch für die neuronalen Netze bedeutet dies weniger Rechenoperationen, da die Anzahl der Pixel minimiert wurde.

3.1.4 Knuffingen Datensatz

Als Umgebung für den ersten Datensatz diente der Bauabschnitt Knuffingen des Miniatur Wunderland Hamburg. Knuffingen zeichnet sich dadurch aus, dass dort bereits Fahrzeuge unter dem Faller Car System im Betrieb sind und somit die Fahrspurbreiten und maximale Kurvenradien den Eigenschaften der Fahrzeuge entsprechen. In statischen Straßenumgebungen, wie die meisten Bauabschnitte ohne Car System, werden solche Eigenschaften oft wegen der Ästhetik der Anlage vernachlässigt. Dadurch, dass aber in

Knuffingen Fahrzeuge im Dauerbetrieb fahren, sind teilweise Straßenmarkierungen abgefahren. Dies ist hilfreich, da die Aufnahmen so realitätnäher sind und die Komplexität des Datensatzes erhöht wird.

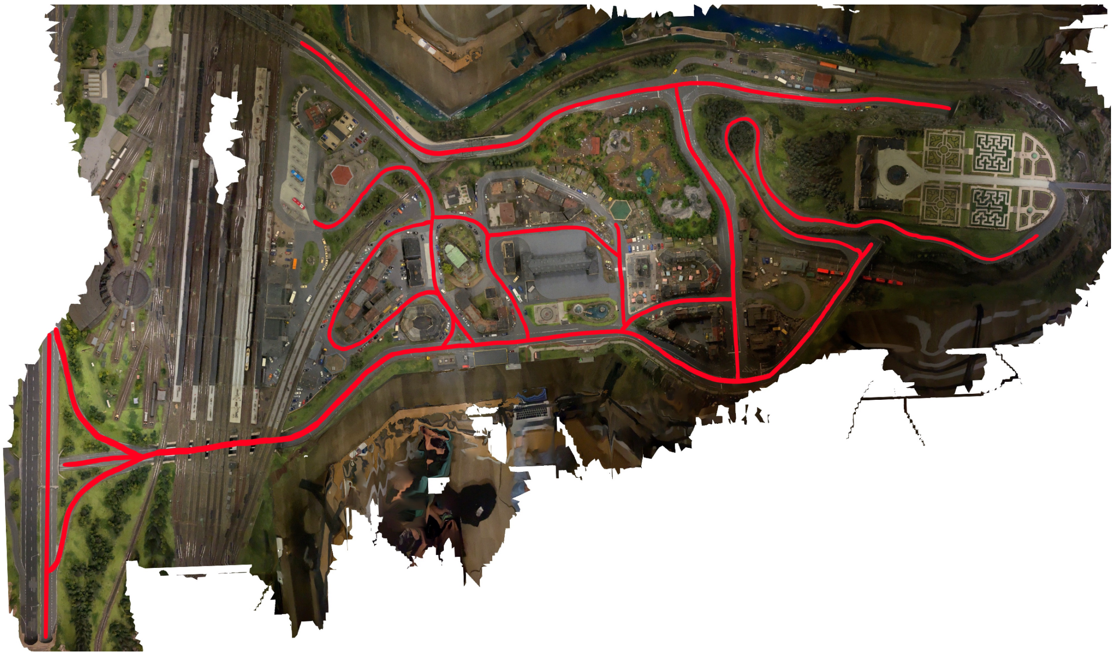


Abbildung 3.6: Knuffingen von oben durch mehrere Bilder zusammengefügt. Aufgenommenes Straßennetz ist in rot eingezeichnet. Quelle: Luk Schwalb

Abbildung 3.6 zeigt eine Karte des Straßennetzes von Knuffingen. In rot eingezeichnet sieht man den mit dem CM4 Fahrzeug abgefahrenen Bereich, jeweils in beide Richtungen, welcher mit einer Framerate von 30 Bildern pro Sekunde aufgenommen wurde. Aus diesen Aufnahmen wurden dann in verschiedenen Zeitabschnitten einzelne Bilder gespeichert, welche annotiert wurden. Für den Knuffingen Datensatz sind das am Ende 708 Bilder bei simulierten Tageslicht. Ebenfalls wurden Aufnahmen bei Nachtbedingungen aufgenommen. Dafür wurde das CM4 Fahrzeug mit Scheinwerfern ausgestattet und die Beleuchtung der Anlage eingeschaltet (siehe Abbildung 3.7). Interessant an dem gesamten Datensatz sind die verschiedenen Straßenkategorien. So gibt es neben dem urbanen Bereich mit komplexen Kreuzungen und vielen parkenden Autos, die die Sicht einschränken, auch eine Autobahn oder Landstraßen. Durch einen Berg auf der Anlage, ergibt sich auch ein Tunnel oder eine Straße, die auf den Berg, durch einen kleinen Waldabschnitt, hinaufführt. Vom Tunnel existieren allerdings nur Aufnahmen bei der Einfahrt, da das Fahrzeug aufgrund der Höhe nicht durch den Tunnel hindurch passt.

Abbildung 3.7: Kamerabild des CM4 Fahrzeuges von Knuffingen bei Nacht.

3.1.5 Mikrowunderland Datensatz

Für das Projekt der Miniaturautonomie wurde an der HAW Hamburg ebenfalls eine Anlage im Maßstab 1:87 errichtet. Im Gegensatz zu Knuffingen im Miniatur Wunderland gibt es keine großen Kreuzungen mit verschiedenen Fahrspuren, sondern lediglich Einmündungen ohne gesonderte Ab- oder Einbiegespuren. Die zwei Arten von Einmündungen, die vorzufinden sind, sind in Abbildung 3.8 zu sehen. Ebenfalls kann man dort erkennen, wie die Farbe des Asphalts auf der Anlage variiert. Das ganze ist bewusst so gemacht, um die Varianz im Datensatz ein wenig zu erhöhen. Verschiedene Straßenkategorien, wie im Miniaturwunderland mit z.B. Autobahnen, gibt es leider noch nicht, da der Platz sehr begrenzt ist. Trotzdem wurde versucht, dass die hier generierbaren Datensätze und Testszenarien eine gewisse Komplexität aufweisen. So gibt es beispielsweise im Zentrum zwei kleine Verkehrsinsel mit einem darauffolgendem Fußgängerüberweg und eine schmale abknickende Einbahnstraße (siehe Abbildung 3.9). Wie auch beim Knuffingen Datensatz, wurde hier auch die Strecke mit dem CM4 Fahrzeug abgefahren und mit einer Rate von 30 Bildern pro Sekunde Aufnahmen gemacht. Die Kamera des Fahrzeuges, mit dem dieser Datensatz aufgenommen wurden, ist allerdings in einem anderen Winkel zur Fahrbahn angebracht. Das könnte später Auswirkungen für die neuronalen Netze bedeuten. Annotiert wurden 52 Bilder bei Tageslicht. Es wurden aber auch hier Aufnahmen bei Nacht gemacht. Einmal mit eingeschalten Straßenlaternen und einmal ohne. Die Scheinwerfern des Fahrzeuges waren dabei immer eingeschaltet. Abbildung 3.9 zeigt die gesamte Anlage von oben, welche abgefahren wurde.

(a) Einmündung mit Rechts vor Links Regelung

(b) Einmündung mit Übergeordneter Straße

Abbildung 3.8: Beispiele der zwei Einmündungsarten auf der Mikrowunderland Anlage der HAW Hamburg.

3.2 Auswertung der Trainingsdaten

Um einen Überblick zu bekommen, wie die gesammelten und annotierten Daten aussehen, werden diese im folgenden ausgewertet. Das beinhaltet vor allem, wie die Verteilungen der segmentierten Pixel in den Klassen aussieht. Notwendig ist das im späteren Verlauf für die Auswahl und Interpretation bestimmer Verfahren und ihrer Ergebnisse. Hinzu kommt, dass die hier annotierten Trainingsdatensätze nicht groß sind, wenn man sie mit den Millionen Bildern aus bestehenden Datensätzen, wie CULane [11], vergleicht.

3.2.1 Überblick der verwendbaren Trainingsdaten

Tabelle 3.3 zeigt, wie viele Bilder aus den einzelnen Datensätze annotiert wurden. Der Grund für den geringen prozentualen Anteil, z.B. aus dem Knuffingen bei Tag Datensatz von 1,5 %, besteht aus zwei Faktoren. Zum einen befindet sich das Fahrzeug während der Aufnahme öfters im Stillstand, weshalb viele gleiche Bilder enthalten sind. Zum anderen

Abbildung 3.9: Das Mikrowunderland der HAW Hamburg von oben. Quelle: Markus Kasten

Datensatz	Aufgenommene Rohbilder	annotierte Bilder
Knuffingen bei Tag	46.780	708
Knuffingen bei Nacht	24.675	10
Mikrowunderland bei Tag	2.553	52
Mikrowunderland bei Nacht	3.188	10
Mikrowunderland bei Nacht ohne Straßenbeleuchtung	1.688	10

Tabelle 3.3: Verteilung der annotierten Bilder auf Datensätze

fuhr das Fahrzeug mit nur rund 50 mm/s, wodurch alle 1,6 mm ein Bild gemacht wurde (bei 30 FPS). Somit gibt es keine großen Unterschiede in den aufeinanderfolgenden Sequenzen von Bildern. Das annotieren der Bilder per Hand ist zudem sehr zeitaufwendig, wodurch dann versucht wurde möglichst verschiedene Bilder zu verwenden. So sind insgesamt nur 708 annotierte Bilder aus den ursprünglich 46.780 Bilder. Die Datensätze bei Nacht wurden aufgrund des zeitaufwendigen Annotieren nur minimal annotiert. Das hat den Hintergrund, dass diese Datensätze hauptsächlich als Testdatensätze verwendet werden. Das heißt, nachdem das neuronale Netz trainiert wurde, werden diese Datensätze verwendet um zu evaluieren, wie gut das Netz generalisieren kann. Deshalb wurden pro Datensatz nur 10 Bilder annotiert, welche später den Ground Truth beim Evaluieren darstellen.

3.2.2 Verteilung der Klassen

In Abschnitt 3.1.2 wurden die für die Traningsdaten verwendeten Klassen definiert. Diese treten allerdings nicht gleichverteilt in den Datensätzen auf. Eine Fahrbahnaußenkante ist zum Beispiel häufiger im Datensatz zu finden als ein Fußgängerüberweg, da dieser auf der Anlage selten vorkommt.

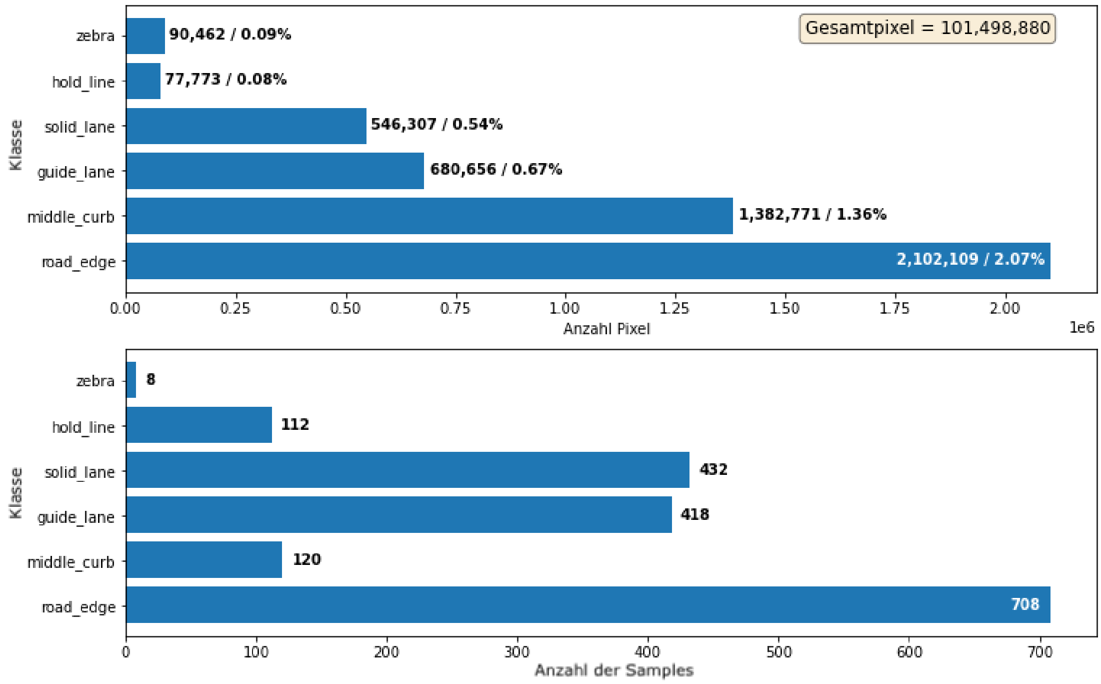


Abbildung 3.10: Verteilung im Knuffigen Datensatz: Oben die zugehörigen Pixel zu einer Klasse, unten die Häufigkeit von Klassen im Datensatz. Die prozentualen Werte setzen sich aus der Anzahl der Pixel insgesamt zusammen, in diesem Fall 101.498.880. Die restlichen Pixel sind nicht klassifiziert und gehören somit dem Hintergrund an. Bei einem perfekt gleichverteilten Datensatz wären die Balken alle gleich groß.

In Abbildung 3.10 kann man sich die entstandene Verteilung im Knuffingen Datensatz anschauen. Für die Anzahl der Pixel wurden pro Klasse alle segmentierten Pixel im gesamten Datensatz gezählt. Bei den 708 Bildern und einer Auflösung von 640 x 224 ergibt sich somit eine Anzahl von 101.498.880 Pixeln. Die Prozentzahl am Ende eines jeden Balken im oberen Diagramm, sagt also aus, wie viel Prozent dies von den Gesamtpixeln ausmacht. Zum anderen zeigt die Abbildung aber auch das Auftreten einer Klasse im Datensatz. Eine Klasse tritt in einem Bild auf, wenn mindestens ein Pixel dieser Klasse zugehörig ist. Wie zu sehen, ist die Fahrbahnaußenkante in jedem Bild zu finden, da sie in allen 708 Bildern auftaucht. Das macht sich auch bei den einzelnen Pixeln bemerkbar, da dieser Klasse die meisten Pixel zugehören. An zweiter Stelle bei der Anzahl der Pixel befindet sich die Sperrfläche mit 1,36 %, obwohl diese nur in 120 Bildern vorkommt. In Abbildung 3.5 ist veranschaulicht, welche Fläche Sperrflächen im Bild einnehmen können und somit eine große Anzahl von einzelnen Pixeln generieren. Sie werden eben nicht als Polygonzüge, sondern als Polygone segmentiert. Also als ganze Flächen. Damit gehören dieser Klasse direkt mehr Pixel an, obwohl das Auftreten einer Sperrfläche im Datensatz vergleichbar gering ist.

Mit Hilfe dieser Diagramme kann man zum Beispiel eine Klassenimbalance feststellen. Das bedeutet die Unausgewogenheit des Datensatzes. Im Idealfall sind alle Vorkommnisse gleich verteilt. In dem hier aufgeführten Fall ist dem allerdings nicht so. Die Annahme ist, dass ein neuronales Netz beim Training immer auf die überwiegende Klasse optimiert und somit gering auftretende Klassen ignoriert. Bei einer Segmentierung ist vor allem die Anzahl der einzelnen Pixel entscheidend und nicht das Auftreten einer Klasse im Bild. Allerdings ist in beiden Fällen eine Klassenimbalance festzustellen.

Schaut man sich in Abbildung 3.11 die Verteilung für den Mikrowunderland Datensatz an, kann man ebenfalls eine Klassenimbalance feststellen, vor allem in Richtung des Fußgängerüberwegs. Hinzu kommt, dass Markierungen wie Haltlinien oder Sperrflächen gar nicht aufkommen. Das liegt vor allem daran, dass es diese auf der Anlage der HAW Hamburg nicht gibt (siehe Abbildung 3.9). Aufgrund dieser Tatsache ist der Mikrowunderland Datensatz als alleiniger Trainingsdatensatz nicht geeignet, da manche Klassen gar nicht abgebildet werden.

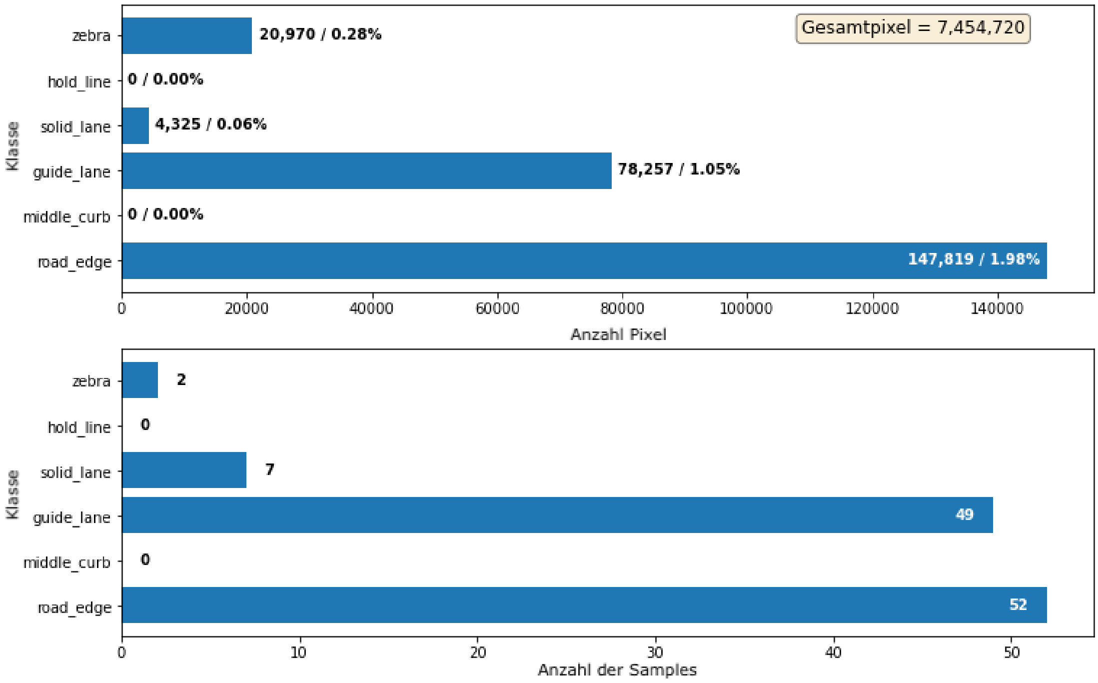


Abbildung 3.11: Verteilung im Mikrowunderland Datensatz: Oben die zugehörigen Pixel zu einer Klasse, unten die Häufigkeit von Klassen im Datensatz.

3.2.3 Zerteilung der Trainingsdaten

In Machine Learning Verfahren ist es üblich, dass der Datensatz in Traningssatz und Validierungssatz gesplittet wird. Der Traniningssatz wird dabei ausschließlich fürs Lernen, also für die backpropagation genutzt. Um während des Trainings bereits die Generalisierung zu prüfen, wird ein weiterer Validierungssatz erzeugt. Dieser wird am Ende einer jeden Epoche ins Netz gegeben, um zu evaluieren, wie gut die Generalisierung vorangeschritten ist. Somit kann auch ein Auswendiglernen frühzeitig erkannt werden. In der Regel geschieht dieses Sampeln in Trainings- und Validierungssatz zufällig und es wird eine Verteilung angegeben, wie viel Prozent in den Validierungssatz eingehen. Der Rest geht in den Trainingssatz ein. Üblich sind Werte zwischen 15-30 %.

In Abbildung 3.12 ist zu sehen, wie die Verteilung nach einem zufälligen Sampeln aussehen kann. Beim Fußgängerüberweg sind gar keine Bilder mehr im Validierungssatz zu finden. Das bedeutet, dass wenn ein Auswendiglernen für den Fußgängerüberweg stattfindet, dieses nicht erkannt werden kann, da es keine Daten mehr mit einem Fußgängerüberweg gibt. Allgemein bedeutet das, dass man die Parameter in anbetracht auf den

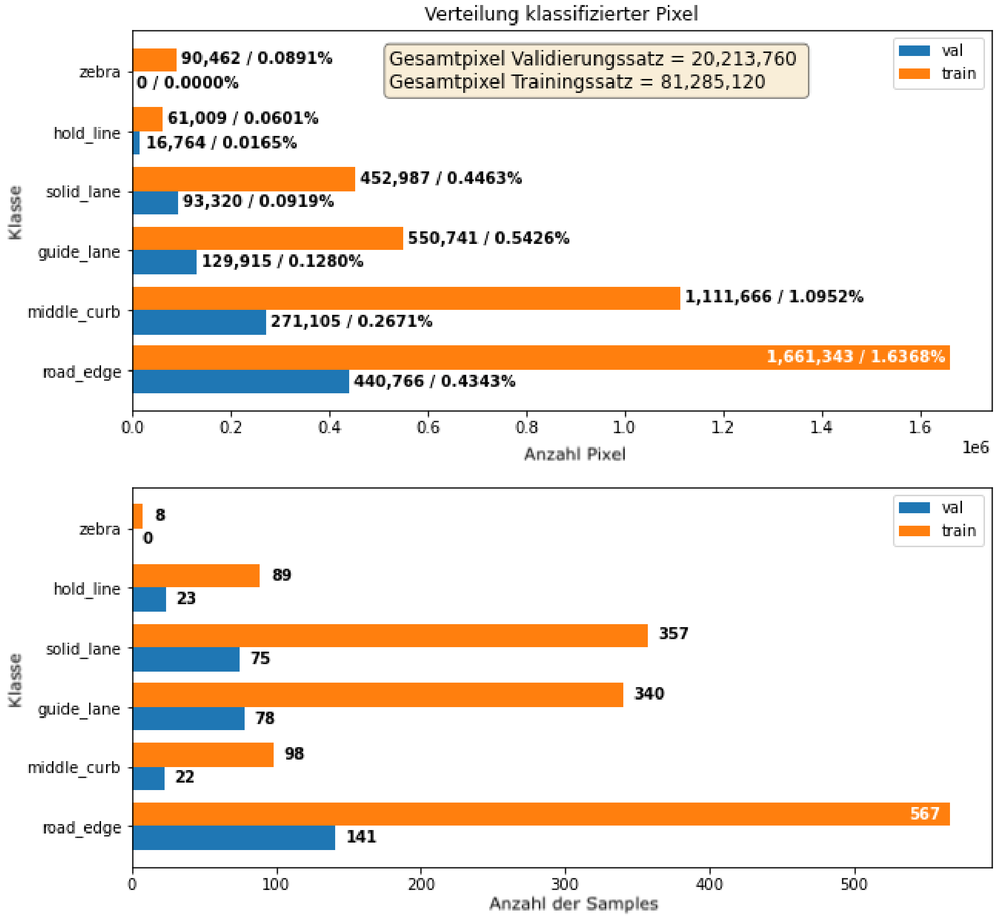


Abbildung 3.12: Verteilung im Knuffingen Datensatz nach Random Sampling mit 20 % Validierungsgröße: Oben die zugehörigen Pixel zu einer Klasse, unten die Häufigkeit von Klassen im Datensatz. Hierbei ist der Fall eingetreten, dass der Fußgängerüberweg keine Daten im Validierungsdatensatz hat.

Fußgängerüberweg nicht gut einstellen kann, da man nicht weiß, wie sich das Netz bei unbekannten Fußgängerüberwegen verhält. Es sei allerdings zu dieser Verteilung gesagt, dass hier ein möglicher Worst Case dargestellt wird. In der Regel sind selbst die seltenen Klassen nach der Validierungsgröße verteilt.

Eine andere Möglichkeit ist das Sampeln mit Berücksichtiging der Klassenverteilung. Dabei kann das Sampeln so stattfinden, dass die Klasse mit dem geringsten Aufkommen zuerst berücksichtigt werden und dort ein Sampling nach Validierungsgröße stattfindet. Bei 20 % Validierungsgröße und 8 Bildern mit Fußgängerüberweg wäre das z.B. ein Bild im Validierungssatz und sieben im Traningssatz (wenn immer abgerundet wird).

Ist das Sampeln für die erste Klasse erledigt, so kann man zur nächst größeren springen und dort das gleiche Prozedere anwenden, bis man insgesamt 20 % des Datensatzes im Validierungssatz hat. Das funktioniert allerdings nur bei stark ungleich verteilten Klassen wie hier. Ansonsten würde die letzte zu behandelnde Klasse leiden, wenn zuvor schon 20 % erreicht wurden. In dem angewandten Datensatz wären das aber trotzdem 20 %, da die häufigste Klasse, die Fahrbahnaußenkante, in jedem einzelnen Bild vorkommt. Somit wird die automatisch in den ersten Durchläufen mitgesampelt.

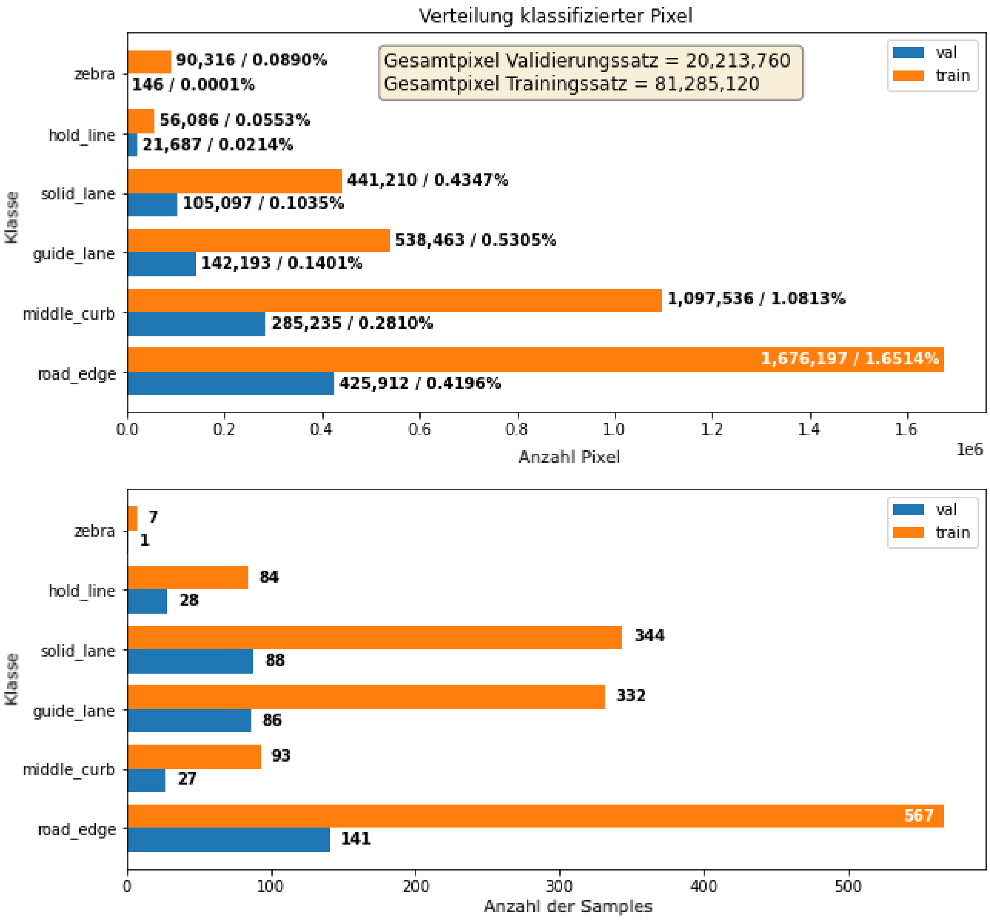


Abbildung 3.13: Verteilung im Knuffingen Datensatz nach Stratified Sampling mit 20 % Validierungsgröße: Oben die zugehörigen Pixel zu einer Klasse, unten die Häufigkeit von Klassen im Datensatz.

Wie das ganze beim gleichen Datensatz, wie in Abbildung 3.12, aussieht, kann man in Abbildung 3.13 erkennen. Das vorgestellte Verfahren ist nicht optimal, da dies andere Probleme erzeugen kann, wie eben dargestellt. Allerdings gewährleistet es, dass mindestens die angestrebte Validierungsgröße in den seltensten Klassen vorhanden sein wird. Inwiefern die verschiedenen Sampling Methoden einen Einfluss auf die Genauigkeit der Netze haben können, wird in den Experimenten im späteren Kapitel getestet.

3.3 Architekturen

In diesem Abschnitt wird ein Überblick über die in dieser Arbeit verwendeten Architekturen von neuronalen Netzen gegeben. Wie bereits zu Beginn erwähnt, wird sich in dieser Arbeit auf sogenannten Unet Architekturen fokussiert. Das sind Architekturen, welche ähnlich einem Autoencoder aufgebaut sind. Das bedeutet es gibt einen Encoder, welcher verschiedene Merkmale des Eingangsbildes lernt und einen Decoder, welcher anhand der extrahierten Merkmale des Encoders ein neues Bild mit der gleichen Auflösung generiert. Bildinformationen werden also zwischen Encoder und Decoder in einer sehr komprimierten Form repräsentiert. Die Besonderheit bei Unet Architekturen ist, dass es skip connections gibt. Dies sind Verbindungen zwischen Encoder und Decoder, an denen Informationen vom Layern im Encoder direkt an einen Layer, mit den gleichen Dimensionen, im Decoder gegeben werden. Im Decoder werden diese mit den Informationen aus den unteren Layern zusammengefügt. So soll ermöglicht werden, dass feingranulare Informationen aus dem Encoder nicht verloren gehen können und jeweils mit in den Output einfließen können. Das ist allerdings nur eine Annahme, da es keinen theoretischen Beweis dafür gibt. Solche skip connections waren allerdings schon vor Unet Architekturen im Einsatz. Sie sind zum Beispiel auch die Idee hinter residual networks (ResNet) [5].

Im folgenden werden nun verschiedene Unet Architekturen vorgestellt, welche in den darauffolgenden Experimenten verwendet werden. Die Grundidee hinter allen vorgestellten Architekturen ist, dass bereits existierende convolutional neural nets (CNN) als Grundlage verwendet werden. Diese bilden das backbone der neuen Unet Modelle. Die CNNs sind bereits auf dem imagenet Datensatz vortrainiert. Somit wird hier ein Transfer Learning angewendet. Da es sich bei Imagenet um eine Klassifikation handelt, bestehten diese vortrainierten Netze aus einem Faltungsteil und fully connected (FC) Layern. Bei der Segmentierung geht es allerdings um Klassifikation auf Pixelebene, weswegen der fully connected Teil durch den neu geschaffenen Decoder ersetzt wird. Das hat auch zum Vorteil, dass so die Bildauflösung des Eingangsbildes varieren kann. Die Bildgröße ist für die Faltungskerne des CNN irrelevant, lediglich bei Verwendung von FC Layern beeinflusst die Bildgröße die Anzahl der Gewichte. Das hat den Hintergrund, dass bei Faltungslayern die Gewichte in den Faltungskernen liegen und nicht, wie bei den fully connected Layern zwischen jedem möglichen Pixel.

Zudem verwenden alle vorgestellten Architekturen die Softmax Aktivierung im letzten Layer. In diesem Anwendungsfall der Straßenmarkierungen kann jeder Pixel nur einer Klasse angehörig sein. Es handelt sich also um eine 1 aus n Klassifikation auf Pixelebene. Daher wird eine Softmax Aktivierung genutzt, und keine Sigmoid. Wenn ein Pixel allerdings keiner Klasse angehörig ist, muss sie einer neuen Background Klasse zugehören, ansonsten funktioniert Softmax nicht. Es wird also zu allen vorgestellten Klassen eine künstliche Background Klasse erzeugt, welche alle nicht klassifizierten Pixel in den Annotationen beinhaltet.

Modell	Anzahl Parameter ohne FC Layer	Anzahl Parameter insgesamt [19]
VGG16 [16]	**14.714.688**	128.357.544
VGG19 [16]	20.024.384	143.667.240
ResNet50V2 [6]	24.564.800	25.613.800
ResNet101V2 [6]	42.626.560	60.380.648
MobileNetV2 [14]	**2.257.984**	3.538.984
InceptionV3 [17]	21.802.784	23.851.784
Xception [4]	20.861.480	22.910.480

Tabelle 3.4: Auszug der verwendbaren CNNs. Die Anzahl der Parameter ohne FC Layer wurde bestimmt, indem nur die Parameter bis zum letzten Faltungslayer gezählt wurden.

In Tabelle 3.4 sind einige verwendbare CNNs aufgelistet. Die Spalte mit Anzahl Parameter ohne FC Layer ist dabei die hier relevante. Obwohl VGG16 mit seinen insgesamt 128.357.544 Parametern mit Abstand zu den größten Netzen gehört, ist der reine CNN Anteil mit 14.714.688 der zweitkleinste. Durch die Anforderung der Miniaturautonomie und die damit einhergehende Begrenztheit an Speicher und Rechenleistung werden die zwei kleinsten CNNs im weiteren Verlauf verwendet, somit also VGG16 und MobileNetV2.

3.3.1 Architektur 1: VGGU

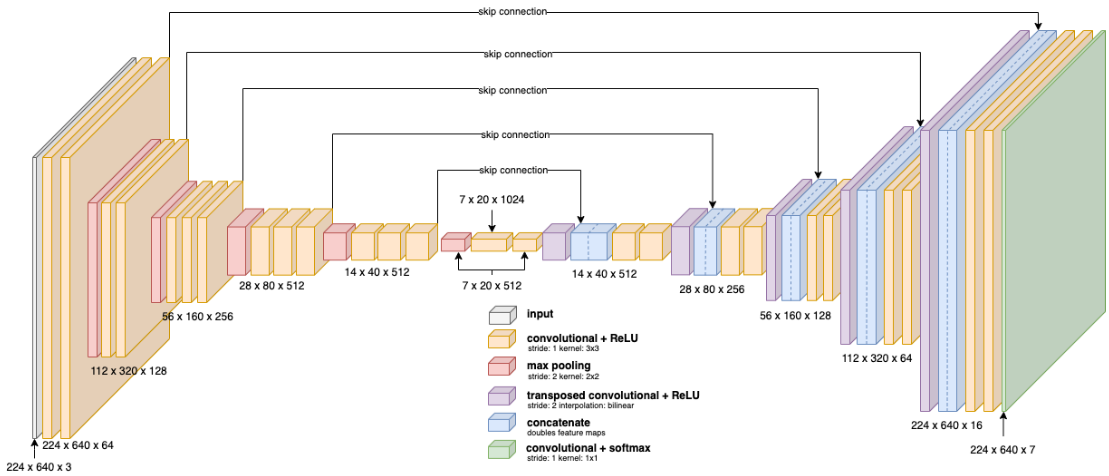


Abbildung 3.14: Darstellung der VGGU Architektur. Breite der Blöcke ist aus Darstellungsgründen nicht proportional zur Anzahl der feature maps.

VGG16 bildet in Tabelle 3.4 nicht nur das zweit kleinste CNN, sondern ist auch im Aufbau simpel gehalten, indem nur reine Faltungslayer verwendet werden [16]. Abbildung 3.14 zeigt diesen Aufbau der gesamten Architektur. Der Encoder besteht im wesentlichen aus Pooling- und Faltungslayern. Beim Decoder werden immer wiederkehrende Layer verwendet. Dabei werden die feature maps zuerst um den Faktor 2 hochskaliert. Das geschieht durch eine lineare Interpolation. Darauf folgt eine Konkatenation, welche die feature maps aus dem upscaling und dem jeweiligen Layer aus dem Encoder (skip connection) aneinanderhängt. So verdoppelt sich die Anzahl der feature maps. In den zwei darauf folgenden Faltungslayer, wird die Anzahl der feature maps, auf die für diesen Block definierte Anzahl, reduziert. Die Anzahl der feature maps in den fünf Decoder Blöcken halbiert sich dabei immer durch den transponierten Faltungslayer. Lediglich der letzte Block bildet eine Außnahme mit seinen 16 Bildern. Die Anzahl musste so gewählt werden, da es bei einer Anzahl von 32 zu einem Fehler in der späteren Konvertierung des Modells kommt. Diese wird im Kapitel zu den Experimenten noch genauer beschrieben. Insgesamt besteht diese Architektur aus 35.901.735 trainierbaren Parametern.

3.3.2 Architektur 2: MobileNetU

Das kleinste CNN, nach Tabelle 3.4, ist MobileNetV2 [14]. Die Besonderheit dabei ist die Verwendung von tiefenweiser seperierbarer Faltung, wie sie in der Xception Architek-

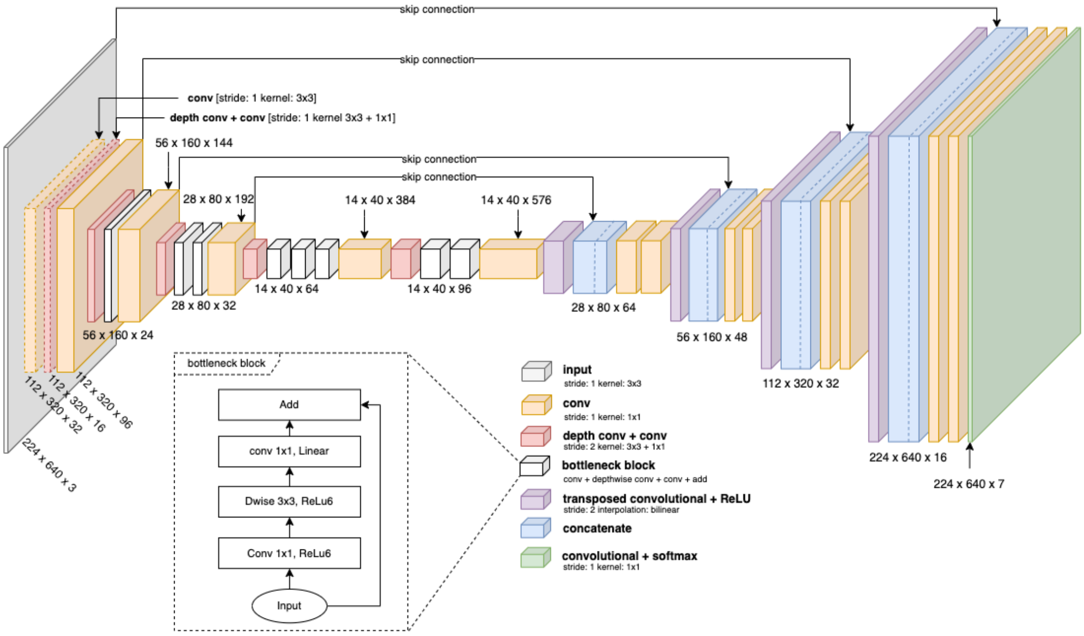


Abbildung 3.15: Darstellung der MobilenetU Architektur. Breite der Blöcke ist aus Darstellungsgründen nicht proportional zur Anzahl der feature maps.

tur vorgestellt wurden [4]. Dadurch können die Parameter und nötigen Multiplikationen reduziert werden. Alle in MobileNetV2 verwendeten normalen Faltungen sind mit 1×1 Kernen. Eine weitere Eigenschaft sind die bottleneck Blöcke, welche in Abbildung 3.15 durch die weißen Blöcke repräsentiert werden. Diese nutzen den Ansatz aus residual networks [5], dass es skip bzw. jump connections über gewisse Layer gibt. Gradienten können so auch durchs Netzwerk gehen, ohne durch nicht lineare Layer laufen zu müssen. Um die Architektur so klein wie möglich zu halten, wurde auch nicht die komplette Tiefe des MobileNetV2 Netzes verwendet. Anstatt alle 10 Bottleneck Blöcke zu verwenden, wurde der Schnitt nach dem 8ten Block, wie in Abbildung 3.15 zu sehen, gemacht. Somit konnte die Anzahl der trainierbaren Parameter von 2.257.984 auf 616.256 reduziert werden. Zusammen mit den Parametern des Decoders ergeben sich 1.335.111 Parameter insgesamt.

3.3.3 Architektur 3: TinyU

Um die MobileNetU Architektur noch weiter in der Größe zu reduzieren, wurde im Decoder die selbe Technik angewendet, welche bei MobileNetV2 dazu führt, dass die Pa-

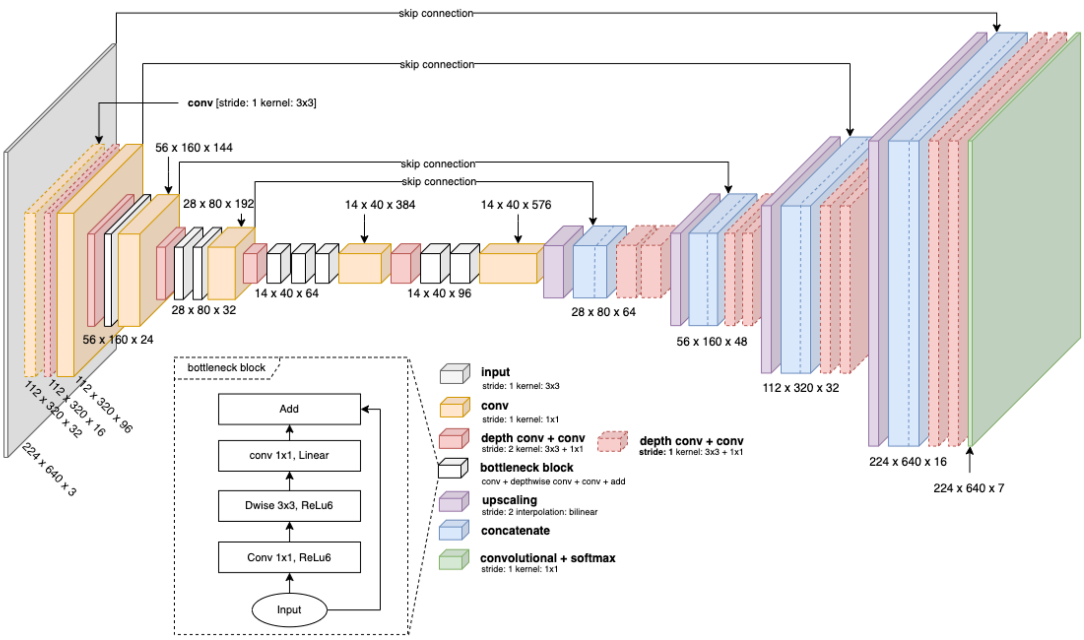


Abbildung 3.16: Darstellung der TinyU Architektur. Breite der Blöcke ist aus Darstellungsgründen nicht proportional zur Anzahl der feature maps.

rameteranzahl so gering ist. Es wurde also ebenfalls tiefenweise seperierbare Faltung verwendet. Zudem wurden die transposed convolutional Layer durch reine Upscaling Layer getauscht, um die Parameter von 718.855 auf 86.853 im Decoder zu reduzieren. Eine Vermutung dadurch ist, dass dieses Netz keine gute Generalisierung mehr zu stande bekommt. Daher wird die vorangegangene MobileNetU Architektur trotzdem in den Experimenten verwendet, da eine Verringerung der Parameter zu einem schlechter generalisierten Netz führen kann. Diese TinyU Architektur besitzt durch die im Decoder verwendete Reduzierung 703.109 Parameter insgesamt. Der Encoder Teil ist der gleiche wie in der MobileNetU Architektur.

4 Experimente

In diesem Kapitel werden die vorgestellten Architekturen trainiert und evaluiert. Anhand der in Kapitel 2.2 eingeführten Metriken, können die neuronalen Netze miteinander verglichen und bewertet werden. Ebenfalls werden Experimente durchgeführt, welche auf die Erklärbarkeit und Zuverlässigkeit deuten könnten. Im letzen Abschnitt werden die Netze anhand ihrer Inferenzzeit evaluiert.

4.1 Training

4.1.1 Fehlerfunktionen

Wie im Grundlagenkapitel vorgestellt, gibt es verschiedene Fehlerfunktionen, mit denen man diese Segmentierungsnetze trainieren kann. Insbesondere wegen der Klassenimbalance in den Datensätzen wurden Fehlerfunktionen vorgestellt, welche diesem entgegenwirken.

Auf Grund der Tatsache, dass alle verwendeten Architekturen in der letzten Schicht die Softmax Funktion als Aktivierungsfunktion nutzen, wird eine weitere Klasse dem Trainingsdatensatz hinzugefügt. Diese spiegelt den Hintergrund wieder, also alle nicht durch den Ground Truth klassifizierten Pixel. So ist sichergestellt, dass ein Pixel nur maximal einer Klasse zugehört.

Im folgenden werden die neuronalen Netze mit dem Knuffingen bei Tag (Tabelle 3.3) Datensatz trainiert. Die Validierungsdaten werden durch random sampling aus dem Trainingsdatensatz erzeugt. Dabei werden 20 % der Bilder gesampled. Die Samples sind allerdings bei allen trainierten Modell gleich, um eine Vergleichbarkeit zu gewährleisten. Für den Testdatensatz wird der Mikrowunderland bei Tag (Tabelle 3.3) Datensatz verwendet. Die Lernrate wurde für jede Fehlerfunktion eingehändig durch testen ermittelt. Als

Optimierer für die Backpropagation wurde dabei Adam verwendet. Alle Netze wurden in 100 Epochen trainiert.

	Validation				Test			
Modell	IoU	F1	Precision	Recall	IoU	F1	Precision	Recall
VGGU + CE	0.42	0.60	0.55	0.57	0.26	0.39	0.34	0.34
VGGU + WCE	0.43	0.50	**0.75**	0.58	0.27	0.31	**0.48**	0.35
VGGU + D	0.64	**0.71**	0.70	**0.71**	0.32	0.46	0.37	0.40
VGGU + T	**0.65**	0.69	0.71	0.70	**0.35**	**0.48**	0.43	**0.42**
VGGU + FT	**0.65**	0.70	0.73	**0.71**	0.34	0.44	0.41	0.41
MobileNetU + CE	0.42	0.56	0.55	0.56	0.28	0.37	0.37	0.36
MobileNetU + WCE	0.49	0.53	**0.73**	0.61	0.31	0.36	0.47	0.39
MobileNetU + D	0.59	0.67	0.67	0.66	0.31	0.38	0.37	0.37
MobileNetU + T	0.58	0.64	0.68	0.65	0.39	0.44	**0.50**	0.46
MobileNetU + FT	**0.62**	**0.69**	0.71	**0.69**	**0.40**	**0.46**	**0.50**	**0.47**
TinyU + CE	0.39	0.52	0.52	0.51	0.29	0.37	0.42	0.38
TinyU + WCE	0.39	0.46	**0.65**	0.51	0.28	0.33	**0.52**	0.37
TinyU + D	0.51	**0.65**	0.57	**0.60**	**0.33**	**0.45**	0.42	**0.40**
TinyU + T	0.49	0.58	0.60	0.58	0.32	0.38	0.42	0.39
TinyU + FT	**0.52**	0.61	0.61	**0.60**	0.32	0.38	0.41	0.38

Tabelle 4.1: Trainingsergebnisse mit $\alpha = 0.6$ bei T und FT und $\gamma = 0.75$ bei FT. Bei WCE wurden folgende Gewichte W verwendet: 0.017, 0.026, 0.052, 0.065, 0,443, 0.394, 0,0003. Aus jeder Architektur wurde der jeweils beste Score markiert.

In Tabelle 4.1 sind die Ergebnisse des Trainings dargestellt. Für jede Architektur wurden dabei die besten Ergebnisse fett markiert. Die Gewichte für die WCE Fehlerfunktion wurden anhand der Verteilung der Pixel, nach Abbildung 3.10, berechnet. Diese Gewichte sind normalisiert und ergeben in Summe eins. Schaut man sich die Werte der einzelnen Metriken an, so kann man einen deutlichen Unterschied zwischen Validierungs- und Testdatensatz erkennen. Erwähnt sei dabei aber, dass der Testdatensatz, also der Knuffingen bei Tag Datensatz, mit einem anderen CM4 Fahrzeug aufgenommen wurde. Bei diesem Fahrzeug war die Kameraperspektive ein wenig mehr nach unten geneigt als beim Fahrzeug, mit dem der Trainingsdatensatz aufgenommen wurde. So gibt es zwei Vermutungen für die schlechteren Werte. Zum einen kann dies auf die andere Kameraperspektive zurückzuführen sein oder zum anderen auf die Generalisierbarkeit der Netze auf andere Umgebungen. Schaut man sich als Beispiel den Traningsverlauf des MobileNetU + FT Netzes an (Anhang A), so wird man auch dort einen Unterschied zwischen Trainings-und Validierungsdatensatz in den Plots zu den Metriken erkennen. Dieser Unterschied ist allerdings im Vergleich zu dem Unterschied zum Testdatensatz geringer.

Mit Hilfer einer Konfusionsmatrix kann man feststellen, welche Klassen oft falsch zugeordnet werden. Diese sind in Abbildung 4.1 zu sehen. Es wurden dafür die drei Modelle

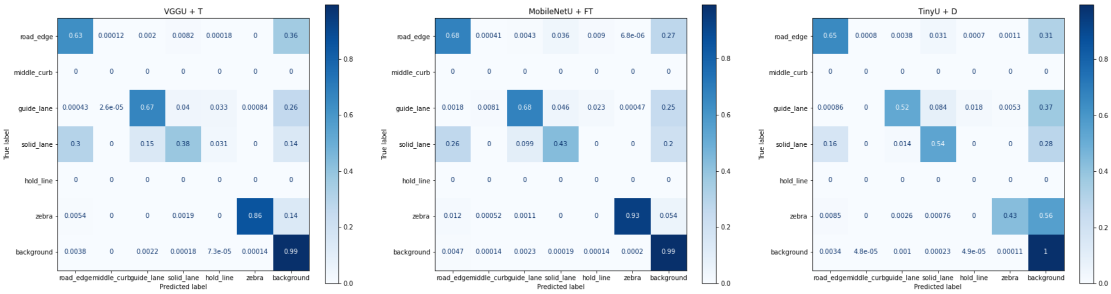


Abbildung 4.1: Konfusionsmatrix der drei Netze mit den höhsten Scores im Testdatensatz. Die Werte sind über die True Label normalisiert.

ausgewählt, welche jeweils die höhsten Scores im Testdatensatz erreicht haben. Man kann auch direkt einen Unterschied zwischen dem TinU + D Netz und dem Rest feststellen, wenn man sich die Fehler in der Zebra Klasse anschaut. Während MobileNetU + FT mit 93 % die höhste Erfolgsquote hat, erreicht TinU + D lediglich 43 %. Die meisten Pixel, welche eigentlich zur Zebra/Fußgängerüberweg Klasse gehören, werden dem Background zugeordnet. Für die restlichen Klassen sehen die Matrizen ähnlich aus. Eine Vermutung für den niedrigeren IoU und F1 Score in Tabelle 4.1, kann die häufige Fehlklassifikation in der Zebra/Fußgängerüberweg Klasse sein. Diese Vermutung wird im weiteren Verlauf näher untersucht.

4.1.2 Parametertuning

Im vorherigen Schritt wurden verschiedene Fehlerfunktionen an den Architekturen getestet und miteinander verglichen. In diesem Abschnitt soll darauf aufgebaut werden, indem exemplarisch an der MobileNetU Architektur die Parameter an der Focal Tversky Fehlerfunktion eingestellt werden. Die einzelnen Kombinationen werden miteinander verglichen und Probeweise auf die anderen beiden Architekturen überführt. Auch hier wird anhand der in Abschnitt 2.2 vorgestellten Metriken evaluiert. Wie in Abschnitt 2.3 gezeigt, können α und γ an der Tversky bzw. Focal Tversky Fehlerfunktionen eingestellt werden.

Modell	Parameter	Test IoU	F1	Precision	Recall
MobileNetU + FT	$\alpha = 0.6, \gamma = 0.75$	**0.40**	**0.46**	0.50	**0.47**
MobileNetU + FT	$\alpha = 0.5, \gamma = 0.75$	0.31	0.38	0.36	0.36
MobileNetU + FT	$\alpha = 0.7, \gamma = 0.75$	0.39	0.44	0.51	0.46
MobileNetU + FT	$\alpha = 0.8, \gamma = 0.75$	0.31	0.36	0.41	0.37
MobileNetU + FT	$\alpha = 0.9, \gamma = 0.75$	0.39	0.43	**0.52**	0.46
MobileNetU + FT	$\alpha = 0.6, \gamma = 1.25$	0.38	0.44	0.48	0.45
MobileNetU + FT	$\alpha = 0.6, \gamma = 0.5$	0.39	**0.46**	0.50	0.46
MobileNetU + FT	$\alpha = 0.7, \gamma = 1.25$	0.39	**0.46**	0.47	0.46
TinyU + FT	$\alpha = 0.6, \gamma = 0.75$	0.32	0.38	0.41	0.38
TinyU + FT	$\alpha = 0.7, \gamma = 1.25$	**0.38**	**0.45**	**0.46**	**0.45**
VGGU + FT	$\alpha = 0.6, \gamma = 0.75$	**0.34**	**0.44**	0.41	**0.41**
VGGU + FT	$\alpha = 0.7, \gamma = 1.25$	0.31	0.39	**0.42**	0.39

Tabelle 4.2: Verschiedene Parameter der Focal Tversky Fehlerfunktion am Testdatensatz evaluiert. An der MobileNetU Architektur wurden verschiedene Parameter getestet. An den anderen Architekturen wurde nur eine weitere Kombination getestet, um einen Vergleich zu haben, ob sich die Parameter bei jedem Netz anders auswirken.

Tabelle 4.2 zeigt die Ergebnisse für die MobileNetU + FT Architektur mit verschiedenen Parameterwerten. Die Standardwerte, mit denen zuvor trainiert wurde (Tabelle 4.1), ergeben auch hier im Schnitt die besten Scores. Im F1 Score wurde der gleiche Werte allerdings auch mit $\alpha = 0.7, \gamma = 1.25$ und $\alpha = 0.6, \gamma = 0.5$ erzielt. Die erste Kombination wurde auch an der TinyU und VGGU Architektur getestet. An der TinyU kann man dadurch eine deutliche Erhöhung der Messmetriken erkennen. Bei der VGGU Architektur allerdings eine Verschlechterung. Diese Diskrepanz lässt datauf schließen, dass für jede Architektur ein eigenständiges Parametertuning durchgeführt werden muss.

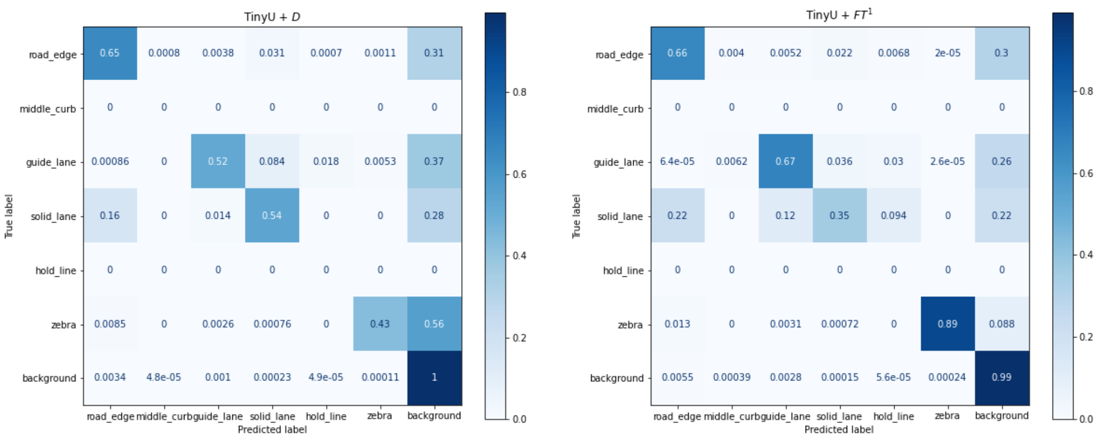
In Tabelle 4.3 wurden die gleichen Netze am Validierungsdatensatz angewendet. Auch hier sind ähnliche Ergebnisse zu erkennen. Die $\alpha = 0.7, \gamma = 1.25$ Kombination erzielt hier sogar die höhsten Messwerte in allen Metriken.

Im vorherigen Abschnitt konnte man in Abbildung 4.1 feststellen, dass das TinU + FT Netz einige Fehlklassifikationen an der Zebraklasse aufweist. Mit Hilfe der FT Parameter, die mit dem MobileNetU Netz ermittelt wurden, konnte allerdings auch die Scores des TinyU Netzes erhöht werden. Die Annahme war, dass der ehemalige niedrige Score, mit der D oder auch FT Fehlerfunktion, an genau dieser Zebraklasse liegt. Anhand von Abbildung 4.2 kann man nun allerdings sehen, dass durch die neuen Parameter in der

Modell	Parameter	Validation IoU	F1	Precision	Recall
MobileNetU + FT	$\alpha = 0.6, \gamma = 0.75$	0.62	**0.69**	0.71	0.69
MobileNetU + FT	$\alpha = 0.5, \gamma = 0.75$	0.57	0.65	0.64	0.64
MobileNetU + FT	$\alpha = 0.7, \gamma = 0.75$	0.60	0.66	0.70	0.67
MobileNetU + FT	$\alpha = 0.8, \gamma = 0.75$	0.60	0.63	0.72	0.67
MobileNetU + FT	$\alpha = 0.9, \gamma = 0.75$	0.58	0.61	0.74	0.66
MobileNetU + FT	$\alpha = 0.6, \gamma = 1.25$	0.59	0.64	0.68	0.65
MobileNetU + FT	$\alpha = 0.6, \gamma = 0.5$	0.60	0.66	0.67	0.66
MobileNetU + FT	$\alpha = 0.7, \gamma = 1.25$	**0.63**	**0.69**	**0.72**	**0.70**
TinyU + FT	$\alpha = 0.6, \gamma = 0.75$	0.52	0.61	0.61	0.60
TinyU + FT	$\alpha = 0.7, \gamma = 1.25$	**0.59**	**0.66**	**0.68**	**0.66**
VGGU + FT	$\alpha = 0.6, \gamma = 0.75$	**0.65**	**0.70**	**0.73**	**0.71**
VGGU + FT	$\alpha = 0.7, \gamma = 1.25$	0.63	0.67	0.72	0.69

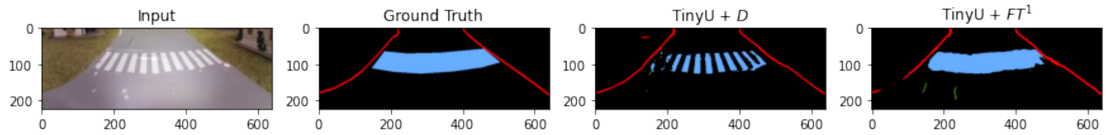
Tabelle 4.3: Verschiedene Parameter der Focal Tversky Fehlerfunktion am Validierungsdatensatz evaluiert. An der MobileNetU Architektur wurden verschiedene Parameter getestet. An den anderen Architekturen wurde nur eine weitere Kombination getestet, um einen Vergleich zu haben, ob sich die Parameter bei jedem Netz anders auswirken.

FT Fehlerfunktion die Zebraklasse weitaus weniger Fehlklassifikationen aufweist. Das bestätigt die Annahme, dass die niedrigen Scores an der Zebraklasse lagen. Durch die höhere Gewichtung der *FN* konnte hier eine Verbesserung des Netzes erreicht werden. Diesen Unterschied kann man auch visuell in Abbildung 4.3 wiedererkennen.



[1] mit $\alpha = 0.7, \gamma = 1.25$

Abbildung 4.2: Konfusionsmatrix des TinyU + *FT* und TinyU + *D* Netzes im Testdatensatz. Die Werte sind über die True Label normalisiert.



[1] mit $\alpha = 0.7, \gamma = 1.25$

Abbildung 4.3: Beispielsegmentierungen eines Zebrastreifens aus dem Testdatensatz mit der TinyU Architektur. Ganz links jeweils der Inputframe, daneben der Ground Truth, gefolgt von den einzelnen Segmentierungen

In Abbildung 4.4 und im Anhang B sind Segmentierungen zwischen den Modellen gezeigt, die die höhsten Metriken aufweisen. Dabei wurde aus jeder Architektur das beste Modell ausgewählt.

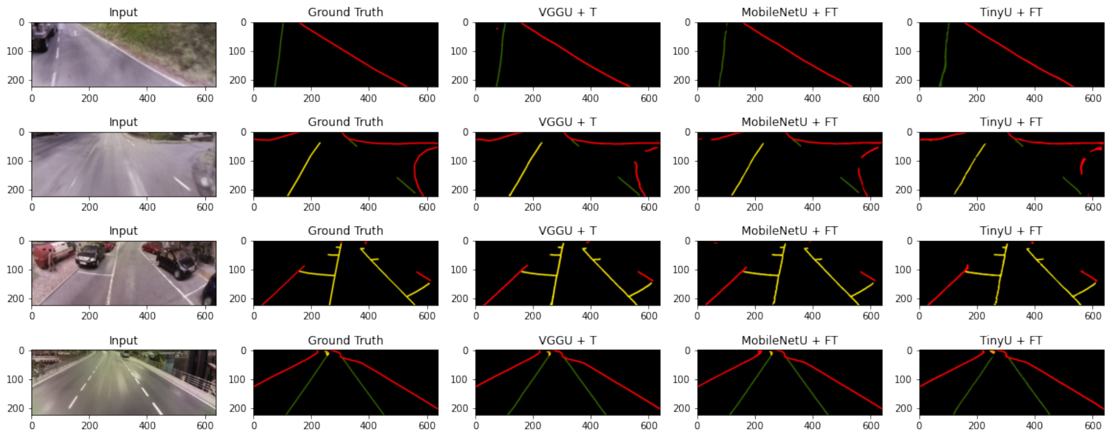


Abbildung 4.4: Beispielsegmentierungen aus dem Validierungsdatensatz mit VGGU, MobileNetU und TinyU. Ganz links jeweils der Inputframe, daneben der Ground Truth, gefolgt von den einzelnen Segmentierungen

4.1.3 Samplingmethoden

In Abschnitt 3.2.3 zur Zerteilung der Trainingsdaten gesehen, entscheidet die Samplingmethode darüber, welche Daten im Trainingssatz und welche im Validierungssatz landen. Nun soll getestet werden, ob ein Stratified Sampling einen Unterschied in der Performance der Netze macht. Dafür wird als Beispiel das MobileNetU + FT mit $\alpha = 0.6, \gamma = 0.75$ verwendet. Im vorherigen Training wurde, wie bereits erwähnt, ein Random Sampling verwendet. Die Verteilung dessen entspricht der, wie in Abbildung 3.12. Dort wird ein möglicher worst case vorgestellt. Interessant hierbei ist die Anzahl der Zebra Klasse im

Trainingsdatensatz, da diese die Minderheit darstellt und bei einem Random Sampling, wie in Abschnitt 3.2.3 erklärt, nur sehr geringfüging auftauchen kann. Um einen Vergleich zu erstellen wurde dieses Architektur mit den gleichen Parametern trainiert, allerdings mit der vorgestellten Stratified Sampling Methode.

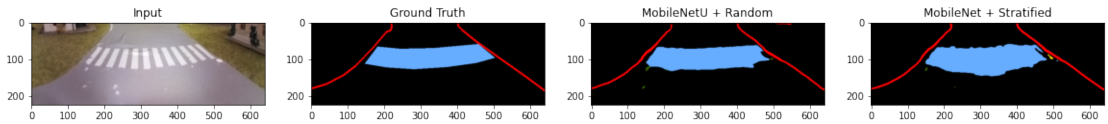


Abbildung 4.5: Beispielsegmentierungen eines Zebrastreifens aus dem Testdatensatz mit der MobileNetU Architektur. Einmal wurde das Netz mit Random Sampling trainiert und einmal mit Stratified Sampling. Bei beiden Samplingmethoden lag die Validierungsgröße bei 20 %.

	Test Zebra			
Modell	IoU	F1	Precision	Recall
MobileNetU + Random	0.34	0.37	0.38	0.38
MobileNetU + Stratified	0.34	0.36	0.39	0.37

Tabelle 4.4: Vergleich zwischen Random und Stratified Sampling anhand der Metriken eines Bildes mit Zebtrastreifen.

In Abbildung 4.5 wurde als Beispiel ein Zebtrastreifen aus dem Testdatensatz segmentiert. Die Metriken, die über dieses Eingangsbild berechnet wurden, zeigen in Tabelle 4.4 keinen großen Unterschied. Man kann also davon ausgehen, dass die vorgestellte Stratified Samplingmethode in diesem Anwendungsbeispiel keine Verbesserung der gesamten Genauigkeit bewirkt.

4.2 Evaluation

Um die Zuverlässigkeit von neuronalen Netzen zu bestimmen, werden nun einige Vorschläge vorgestellt. Darunter auch der Versuch sogenannte Unknown Unkowns aufzudecken. Das sind Dinge von denen wir noch nicht wissen, dass wir sie nicht wissen. Zudem wird auch untersucht, inwiefern die Initialisierung der Gewichte einen Einfluss auf das Ergebnis haben könnte. Zum Schluss werden die Netze noch anhand von Aufnahmen bei Nacht evaluiert, da solche Situationen nicht im Training vorkamen.

Als zu evaluierenden Netzen kommen VGGU + T, MobileNetU + FT mit $\alpha = 0.6, \gamma = 0.75$ und TinyU + FT mit $\alpha = 0.7, \gamma = 1.25$ zum Einsatz. Diese haben in den vorherigen

Abschnitten auf dem Testdatensatz jeweils die höhsten Scores erreicht. Im nachfolgenden sind somit diese Netze + Parameter gemeint, sofern nicht anders angegeben.

4.2.1 Unkown Unkowns

Neben den ganzen vorgestellten Datensätze werden für die Unkown Unkowns extra Sequenzen von Bildern erzeugt. Darin werden einzelne Bereiche der Bilder verändert, bei denen man erwartet, dass es zu keinem Unterschied in der Segmentierung kommt. Es soll also versucht werden, ob man Unterschiede in den Segmentierungen messen kann, obwohl keine Unterschiede zu erwarten wären. Im folgenden werden verschiedene Beispielsequenzen präsentiert, mit denen Segmentierungen mit drei verschiedenen Netzen erzeugt wurden. Diese Segmentierungen werden dann anhand ihrer Pixelwerte miteinander verglichen, um ein Maß für Unterschiede zu bekommen. Eine absolute Aussage darüber, ob ein Netz richtig segmentiert ist nur über einen Ground Truth möglich. Hier geht es allerdings darum, eine Aussage über die Zuverlässigkeit zu bekommen und dabei sollten bei fast gleichem Input keine großen Unterschiede in den Segmentierungen zu sehen sein.

Die in Abbildung 4.6 verwendete Sequenz beinhaltet ein Haus, welches sich am Straßenrand um seine eigene Achse dreht. Dargestellt sind aus der Sequenz Frame f_1 bis f_3, jeweils mit jedem Netz segmentiert. In der linken Spalte ist die Basis b für jedes Netz zu sehen. Diese Basis wird genutzt, um eine Differenz $d(x)$ zu bestimmen. Die segmentierten Bilder der einzelnen Frames werden dabei jeweils von der Basis abgezogen:

$$d(x) = |b - f_x| \tag{4.1}$$

Diese Berechnung passiert Pixelweise. Ist also ein Pixel in $d(x) = 0$, so gab es an dieser Stelle keinen Unterschied. Ist dieser > 0, so ist ein Unterschied zu verzeichnen. Die Anzahl der Pixel > 0 kann also nun als Indikator dienen, inwiefern ein Netz in der Lage ist zuverlässige Ergebnisse zu liefern. Entscheident bei dieser Methode ist allerdings, dass sich die Kameraposition und -perspektive nicht ändern darf.

In Tabelle 4.5 ist jeweils diese Anzahl der Pixel > 0 im Differenzbild zur den jeweiligen Frames und Netzen dargestellt. Dabei erreicht VGGU mit 1013.33 Pixeln im Durchschnitt die kleinsten Veränderungen zwischen den segmentierten Bildern. Anhand dieser Metrik kann man nun festhalten, dass VGGU am zuverlässigsten Segmentiert, wenn es kleine Änderungen in der Umgebung gibt, die die Segmentierung nicht beeinflussen sollten.

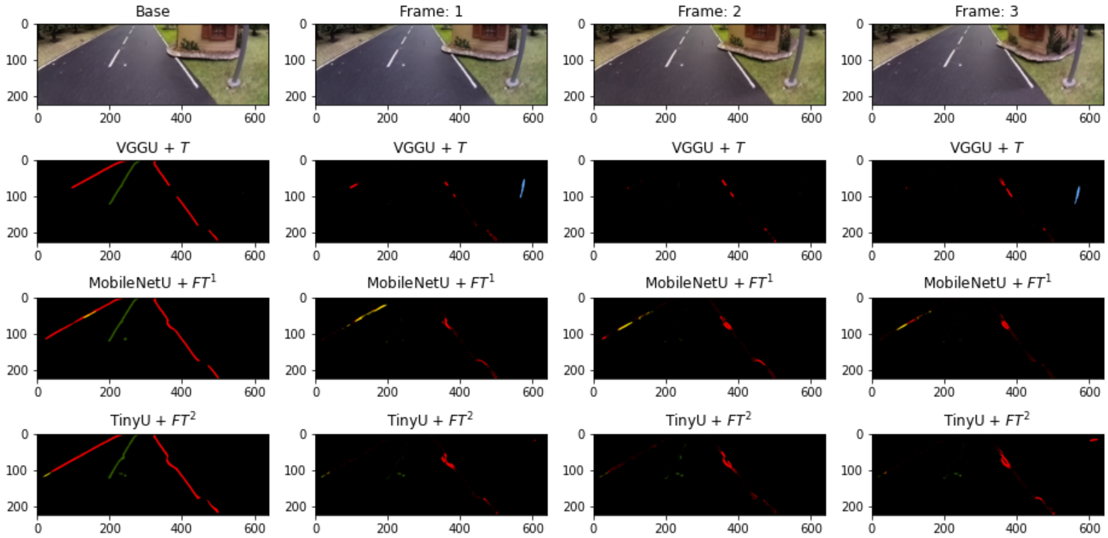


Abbildung 4.6: Differenzbilder für drehendes Haus am Straßenrand. In der oberen Reihe sind die einzelnen Frames der Sequenz zu sehen. Der linke Frame entspricht dem ersten und ist die Basis. In den Reihen dadrunter sind die Differenzbilder der einzelnen Netze zu sehen. Ganz links ist allerdings die einfache Segmentierung der Netz vom Basisframe. Die Differenzbilder beziehen sich immer auf die jeweilige Basis.

Modell	Frame 1	Frame 2	Frame 3	avg
VGGU	1242	636	1162	**1013.33**
MobileNetU	1897	2130	1668	1898.33
TinyU	1575	1832	1950	1785.66

Tabelle 4.5: Anzahl Pixel > 0 im Differenzbild einzelner Frames zur Basis an der Sequenz zum drehenden Haus.

Schaut man sich in Tabelle 4.6 an, in welchen Klassen jeweils diese Differenzen auftreten, stellt man fest, dass nur VGGU Erkennungen von Zebrastreifen aufweist. Sie sind in Abbildung 4.6 blau gekennzeichnet. Diese Klassifizierung tritt auch nicht in der Basis auf und ist nur in Frame 1 und 3 vorhanden. Somit kann man hier davon ausgehen, dass insgesamt VGGU weniger Differenzen aufweist, dafür aber in unerwarteten Klassen. Das gleiche trifft auch auf MobileNetU und TinyU mit der durchgezogenen Linie zu.

Das gleiche Verfahren wird nun auch nochmals an einem anderen Beispiel angewendet, um eine bessere Einschätzung zu erlangen. In Tabelle 4.7 sind die Häufigkeiten zu den Differenzbildern in Abbildung 4.7. Auch hier erreicht VGGU die wenigsten Differenzen. Für dieses Beispiel werden ebenfalls die Anzahlen in Summe pro Klasse in Tabelle 4.8

Modell	Fahrbahn-außenkante	gestrichelte Linie	durch-gezogene Linie	Zebra-streifen
VGGU	987	53	0	480
MobileNetU	2647	123	697	0
TinyU	2213	418	60	0

Tabelle 4.6: Anzahl Pixel > 0 der Differenzbilder im Durchschnitt pro Klasse an der Sequenz zum drehenden Haus. In allen nicht aufgeführten Klassen gibt es keine Pixel > 0. Background Klasse ist ebenfalls nicht aufgeführt.

dargestellt. Im Vergleich zu Tabelle 4.6, in der VGGU Differenzen in der Zebrastreifen Klasse hatte, tritt dieser Fall hier nicht auf. Die Differenzen in den durchgezogenen Linien bei MobileNetU und TinyU bleiben jedoch.

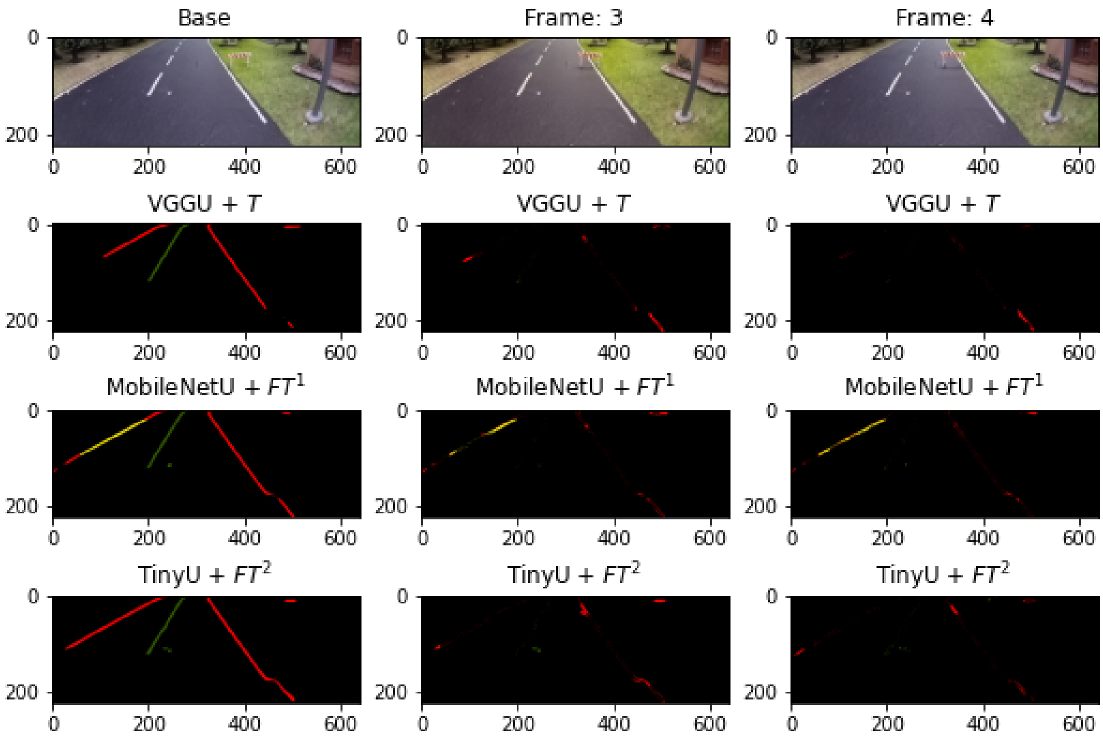


Abbildung 4.7: Differenzbilder für Baustellenabgrenzung, welche auf Fahrbahn wandert.In der oberen Reihe sind die einzelnen Frames der Sequenz zu sehen. Der linke Frame entspricht dem ersten und ist die Basis. In den Reihen dadrunter sind die Differenzbilder der einzelnen Netze zu sehen. Ganz links ist allerdings die einfache Segmentierung der Netz vom Basisframe. Die Differenzbilder beziehen sich immer auf die jeweilige Basis.

Modell	Frame 3	Frame 4	avg
VGGU	1140	646	**893**
MobileNetU	1937	2114	2025.5
TinyU	1613	1194	1403.5

Tabelle 4.7: Anzahl Pixel > 0 im Differenzbild einzelner Frames zur Basis an der Sequenz zur Baustellenabgrenzung.

Modell	Fahrbahn-außenkante	gestrichelte Linie	durch-gezogene Linie	Zebra-streifen
VGGU	842	51	0	0
MobileNetU	1809	98	859	0
TinyU	1193	211	4	0

Tabelle 4.8: Anzahl Pixel > 0 der Differenzbilder im Durchschnitt pro Klasse an der Sequenz zur Baustellenabgrenzung. In allen nicht aufgeführten Klassen gibt es keine Pixel > 0. Background Klasse ist ebenfalls nicht aufgeführt.

Für die Ursachen dieser Differenzbilder wird nun eine Vermutung aufgestellt. Die teilweise großen Unterschiede zwischen den segmentierten Bildern können an einem Unterschied in der Belichtung der Inputbilder liegen. Um dies genauer zu untersuchen wird die Differenz, wie bei den segmentierten Bildern, auch für die Input Bilder berechnet. Dies geschieht hier exemplarisch am Beispiel mit dem drehenden Haus zwischen Basis und Frame 1. Die Ergebnisse sind in Abbildung 4.8 zu sehen. Im linken Bild sieht man, dass es einen

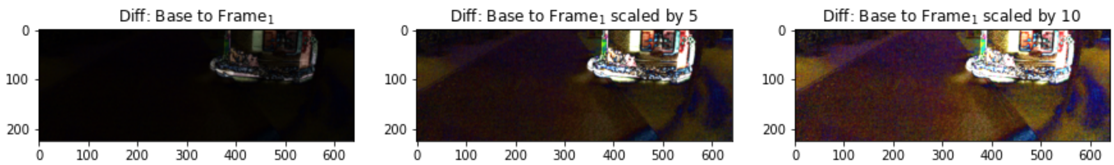


Abbildung 4.8: Differenz zwischen Basis und Frame 1 am drehenden Haus. Dieses Differenzbild wurde noch mit dem Faktor 5 und 10 multipliziert, um Unterschiede besser sichtbar zu machen. Im Bereich des Hauses kam es dabei zu einem Clipping, weswegen die Unterschiede dort anders aussehen.

Unterschied im Bild im Bereich des Hauses gibt. Um auszuschließen, dass die Unterschiede eventuell einfach zu klein sind, um sie darzustellen, werden diese in den danebenliegenden um einen bestimmten Faktor verstärkt. Dort kann man erkennen, dass es Unterschiede im gesamten Bildbereich gibt. Auch wenn diese gering sind, wäre es zu vermuten, dass es ausreicht, damit die segmentierten Bilder umschwenken.

Es bleibt also unklar, ob die Ursachen für die Differenzen tatsächlich an den sich verändernden Objekten in bestimmten Bildbereichen liegt oder aber an der, wie eben untersucht, Belichtung der Umgebung. Zusammengefasst kann man dennoch sagen, dass VGGU innerhalb der drei Netze am zuverlässigsten funktioniert, da dieses Netz die wenigsten Differenzen aufweisen kann. MobileNetU weisst die meisten Differenzen auf und das vor allem auch in anderen Klassen, die man in diesen Beispielen nicht erwarten würde.

4.2.2 Einfluss der initialen Gewichte

Beim Training von neuronalen Netze werden die Gewichte zu Anfang mit bestimmten Werten initialisiert. Bei den in dieser Arbeit vorgestellten Netze werden die Gewichte der einzelnen Layer nach GlorotUniform initialisiert. Dabei werden Stichproben mit einem Limit aus einer gleichmässigen Verteilung gezogen. Diese Stichproben sind bei jeder Initialisierung unterschiedlich.

Im folgenden wird nun untersucht, ob es einen Einfluss der Gewichtsinitialisierung auf die Genauigkeit der Netze nach dem Training gibt. Das Training passt die einzelnen Gewichte bekannterweise bei jedem Schritt an, um den Fehler der Fehlerfunktion zu minimieren. Es werden also nun verschiedene Modelle trainiert, allerdings unterschieden sich nur die initialen Gewichte. Diese sind jedesmal rein zufällig. Die Architektur, Parameter, Fehlerfunktion und Samples der Trainingsdaten bleiben alle gleich. Anhand von Metriken am Testdatensatz wird evaluiert, ob es überhaupt Unterschiede gibt. Die Ergebnisse dieser Auswertung sind in Tabelle 4.9 abgebildet. In jeder Metrik sind Unterschiede von bis zu 0.02 zu verzeichnen.

	Test			
Modell	IoU	F1	Precision	Recall
VGGU + T 1	0.35	0.48	0.43	0.42
VGGU + T 2	0.35	0.47	0.41	0.43
VGGU + T 3	0.36	0.47	0.43	0.42

Tabelle 4.9: Unterschiede in den Metriken bei unterschiedlicher Gewichtsinitialisierung. Als Grundlage diente der Testdatensatz.

In Abbildung 4.6, im vorherigen Abschnitt, konnte man erkennen, dass VGGU + T Pixel als Zebrastreifen klassifiziert hat, obwohl diese vorher nicht vorkamen. Dieses VGGU + T Netz ist das gleiche, was für diesen Abschnitt nochmals mit verschiedener Initialisierung trainiert wurde. In Abbildung 4.9 wurde der gleiche Input verwendet, welcher die Zebrastreifen Klassifizierung getriggert hat, um zu vergleichen, ob dies bei identisch trainierten anderen Netzen auch auftritt. Die dazugehörige Konfusionsmatrix ist in Abbildung 4.10 abgebildet.

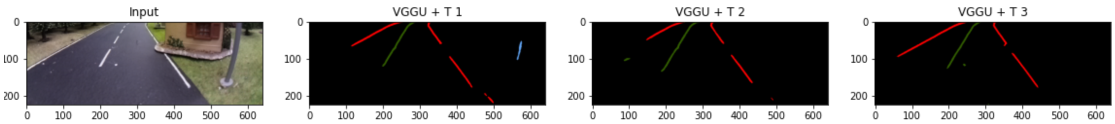


Abbildung 4.9: Segmentierung eines Bild mit identisch trainierten VGGU Netzen. Gewichtsinitialisierung unterscheidet sich.

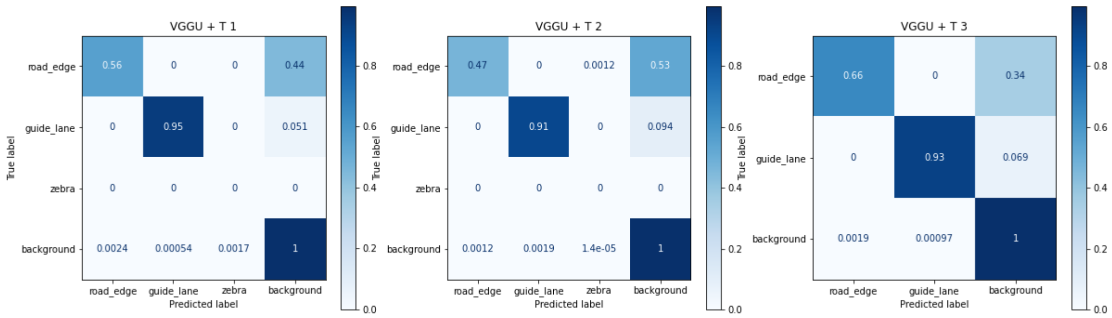


Abbildung 4.10: Konfusionsmatrix zu den Segmentierungen in Abbildung 4.9. Wenn es zu einer Klasse keine Daten gibt, sind diese nicht aufgeführt.

Zu erkennen ist, dass es beim ersten Netz Fehlklassifikationen beim Zebrastreifen gibt. Beim zweiten Netz gibt es zwar auch welche, diese sind mit $1.4e-5$ allerdings sehr gering. Beim dritten Model kommen diese Fehlklassifikationen gar nicht vor, weshalb diese Spalte nicht mit aufgeführt wurde. Aber auch in den anderen Klassen erkennt man, dass das dritte Model zum Beispiel bei den Fahrbahnaußenkanten die besten Ergebnisse liefert. Zusammenfassend konnte also gezeigt werden, dass die Gewichtsinitialisierung durchaus einen Unterschied ausmacht.

4.2.3 Nachtbedingungen

Im Trainingsabschnitt wurde zum Evaluieren hauptsächlich der Knuffingen und Mikrowunderland bei Tag Datensatz verwendet. Allerdings gibt es auch extra Datensätze mit Nachtbedingungen. Anhand dieser werden im folgenden nochmals die Netze verglichen. Die Ergebnisse sind in Tabelle 4.10 dargestellt. Auch hier werden wieder die eingeführten Metriken genutzt, um die Genauigkeit mit Hilfe der Ground Truths zu bewerten.

Modell	Dark Uni				Dark Knuffingen			
	IoU	F1	Precision	Recall	IoU	F1	Precision	Recall
VGGU + T	0.20	0.38	0.22	0.25	0.40	0.62	0.47	0.52
MobileNetU + FT	**0.27**	**0.40**	**0.31**	**0.34**	**0.46**	**0.63**	**0.53**	**0.57**
TinyU + FT	0.26	0.38	0.29	0.32	0.43	0.59	0.52	0.55

Tabelle 4.10: Evaluation bei Datensätzen mit Nachtbedingungen

Auch hier ist ein deutlicher Unterschied zwischen der Knuffingen und Mikrowunderland Umgebung zu erkennen, da alle Netze im schnitt in der Mikrowunderland Umgebung schlechter abschneiden. Einige Beispielsegmentierungen sind in Anhang C zu finden. Ein Ausschnitt ist in Abbildung 4.11 dargestellt. Vergleicht man die Scores der Metriken mit denen aus dem Training, in Tabelle 4.1, mit den hier vorgestellten Werten, so stellt man fest, dass diese im Knuffingen bei Nacht Datensatz besser sind, als im Mikrowunderland bei Tag.

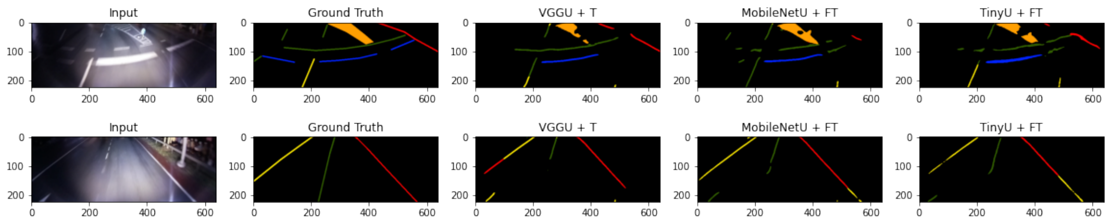


Abbildung 4.11: Beispielsegmentierungen aus dem Knuffingen bei Nacht Datensatz

4.3 Analyse der Inferenzzeit

Dieser Abschnitt beschätigt sich damit die drei Architekturen (VGGU, MobileNetU und TinyU) auf ihre Inferenzzeit zu untersuchen. In den vorherigen Abschnitten wurde die Genauigkeit und Zuverlässigkeit der Segmentierungen untersucht. Aber auch die Inferenzzeit soll in den Bewertungsmaßstab mit einfließen, da die neuronalen Netz Anwendung in der Miniaturautonomie finden. Wie bereits in der Einleitung erwähnt, soll das CM4 Fahrzeug [9] in Kombination mit einem Coral TPU Stick als Maßstab gelten. Es sei hierbei angemerkt, dass die Tests mit einem externen über USB 2.0 angeschlossenen Coral Stick durchgeführt wurden.

Durch die Verwendung von Faltungslayern in den Architekturen ist es möglich, die Bildgrößen für Ein- und Ausgang variabel zu halten. Dies kann bei diesem Test genutzt werden, um verschiedene Bildauflösungen zu testen. Es ist nämlich zu erwarten, dass kleinere Bilder eine schnellere Inferenzzeit erzeugen, da weniger Multiplikationen durchgeführt werden müssen. Die Speichergröße wird allerdings nicht beeinflusst, da die Anzahl der Gewichte konstant bleibt. Lediglich in den verschiedenen Architekturen gibt es Unterschiede.

4.3.1 Messungen

Um eine Vergleichbarkeit zu schaffen wurden alle Tests mit der gleichen Sequenz von Bildern gemacht und die Inferenzzeiten gemittelt. Angemerkt sei auch hierbei, dass die Tests mit der C++ Bibliothek von Tensorflow Lite implementiert wurden, da die Performance bekannterweise besser ist, als mit der Python Bibliothek. Tabelle 4.11 zeigt die Ergebnisse der Testdurchläufe. Mit dem VGGU Netz gab es allerdings fast immer Probleme beim Ausführen, weswegen zwei Meßwerte fehlen.

Modell	Auflösung	Inferenzzeit CM4* in [s]
TinyU	640×224	1.312 ± 0.051
TinyU	320×112	0.066 ± 0.006
TinyU	224×96	$\mathbf{0.027 \pm 0.004}$
MobileNetU	640×224	0.886 ± 0.022
MobileNetU	320×112	0.086 ± 0.002
MobileNetU	224×96	0.039 ± 0.006
VGGU	640×224	6.538 ± 0.813
VGGU	320×96	n/a
VGGU	224×96	n/a

* mit Coral Edge TPU

Tabelle 4.11: Inferenzzeitmessungen am CM4 Fahrzeug mit externem Coral Stick über USB 2.0. Die Auflösung der Input- und Outputbilder wurde variiert, da das zu einer Reduktion der nötigen Operationen führt. Bei VGGU kam es teilweise zu gar keiner Messung, da das Netz ein Absturz des Programms verursachte.

Die kürzeste Inferenzzeit erreichte das TinyU Modell mit einer Auflösung von 224 x 96 Pixeln. Mit einer durchschnittlichen Inferenzzeit von 27 ms, was ungefähr 37 FPS entspricht, ist dieses Netz auch schnell genug, um die Bilder der Kamera mit 30 FPS zu verarbeiten.

4.3.2 Genauigkeitsverlust

Die Coral TPUs unterstützen keine floating pointer numbers, weshalb neuronale Netze, die mit float32 trainiert wurden, quantisiert werden müssen auf int8 oder uint8. Dafür bietet Tensorflow auch einen eigenen Converter an. Man kann nun die Annahme aufstellen, dass durch die Quantisierung von 4 Byte auf 1 Byte Informationen verloren gehen. Schaut man sich Abbildung 4.12 an, so sieht man, dass sich diese Annahme bestätigt.

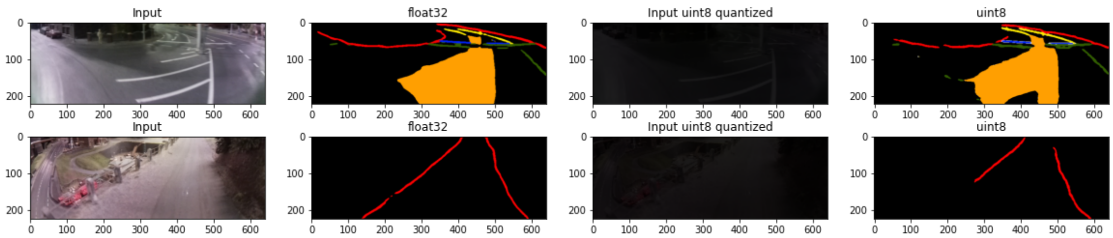


Abbildung 4.12: Segmentierung mit float32 und uint8 Netz im Vergleich. Skalierungsfaktor für uint8 liegt bei 0.027.

Das uint8 Netz scheint eine schlechtere Segmentierung durchzuführen. Dies liegt aber nicht unbedingt an der Quantisierung innerhalb des Netzes, sondern viel mehr an der Quantisierung des Inputs. Dieser ist ebenfalls in Abbildung 4.12 in Spalte 3 dargestellt. Das Bild sieht schwarz aus, allerdings ist das Bild nur sehr dunkel.

Die Quantisierung funktioniert so, dass jeder Layer im Netz einzeln quantisiert wird. Dabei wird ein Skalierungsfaktor und ein Null-Punkt-Offset anhand von Beispieldaten berechnet. Mit diesen beiden Werten kann dann zum Beispiel jeder einzelne Pixelwert des Inputs von float32 auf uint8 gemappt werden.

$$int8\_value = \frac{real\_value}{scale} + zero\_point \tag{4.2}$$

Jedoch scheint es einen Fehler im Tensorflow Converter zu geben, da bei float32 zu uint8 ein Skalierungsfaktor von 0.027 herauskommt. Der Null-Punkt-Offset ist 0, was bei uint8 zu erwarten ist. Mit diesem Skalierungsfaktor werden allerdings float32 Nummern im Bereich $[0.0, 1.0]$ auf uint8 Nummern im Bereich $[0, 37]$ gemappt. Zu erwarten wäre ein Interval von $[0, 255]$, um den gesamten uint8 Raum zu nutzen. Durch das kleine Interval bis 37 kommen dann die dunklen Eingangsbilder zu stande, die in Abbildung 4.12 zu sehen sind. Wenn die Quantisierung auf einem anderen Computer ausgeführt wird, kommt sogar ein Skalierungsfaktor von 0.074 heraus, mit dem nur noch eine Farbtiefe von $[0, 13]$

genutzt wird. Das sind, wie in Abbildung 4.13 zu sehen, definitv zu viele Bildinformationen, die verloren gehen. Eine Lösung für dieses Problem konnte nicht gefunden werden,

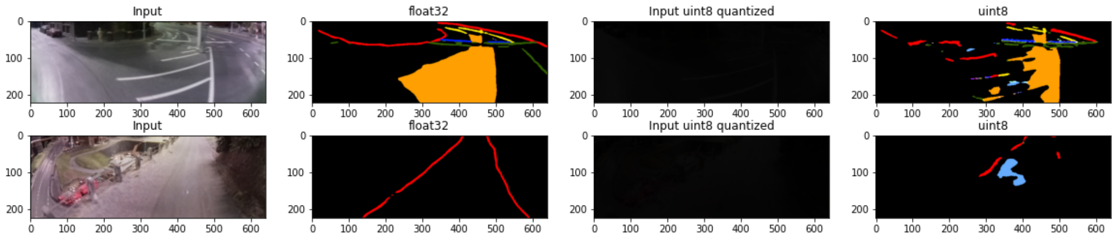


Abbildung 4.13: Segmentierung mit float32 und uint8 Netz im Vergleich. Skalierungsfaktor für uint8 liegt bei 0.074.

weshalb die Genauigkeit der Segmentierungen auf dem CM4 Fahrzeug mit TPU schlechter ist. Zu erwarten ist aber, dass wenn der gesamte uint8 Farbraum genutzt werden könnte, dass es kaum Unterschiede geben wird.

5 Fazit

Im Rahmen dieser Arbeit wurden neuronale Netze entwickelt, welche Kamerabilder eines Miniaturfahrzeuges auf Straßenmarkierungen segmentieren. Dabei wurde sich auf UNet Segmentierungsarchitekturen beschränkt und verschiede Umsetzungen getestet.

Zunächst wurden dafür Bilder gesammelt und ausgewertet. Diese Bilder wurden mit einem Miniaturfahrzeug im Miniatur Wunderland Hamburg und dem Mikrowunderland der HAW Hamburg aufgenommen und anschließend per Hand annotiert. Es wurde eine Definition aufgestellt, welche Klassen von Markierungen verwendet werden und wie diese mit den realen Straßenmarkierungen zusammenhängen. Dabei wurde festgestellt, dass die Datensätze stark unbalanciert sind. Markierungen, wie die eines Zebrastreifens, kommen deutlich weniger vor als Fahrbahnaußenkanten.

Es wurden drei Architekturen vorgestellt, mit denen verschiedene Experimente durchgeführt wurden. Diese Architekturen sind alles Umsetzungen der UNet Architektur, auf die sich beschränkt wurde. Mit Hilfe bekannter Faltungsnetze konnte ein Transfer Learning angewendet werden, um den Trainingsprozess zu beschleunigen. Innerhalb der Architekturen wurden verschiedene Techniken umgesetzt, um den Speicherbedarf der Netze zu verkleinern.

In den Experimenten wurden Netze mit verschiedenen Fehlerfunktionen und Parametern trainiert. Anhand verschiedener Metriken konnten die Netze miteinander verglichen werden. Dort konnte gezeigt werden, dass auch die kleinste Architektur mit 700.00 Gewichten brauchbare Ergenisse liefert. Auch die Klassenimbalance konnte durch die Focal Tversky Fehlerfunktion kontrolliert werden. Dies wurde examplarisch an einem Fußgängerüberweg gezeigt.

Da sich diese Arbeit mit dem Anwendungsfall in der Miniaturautonomie beschäftigt, wurden zusätzlich noch Inferenzzeiten am CM4 Fahrzeug mit einem Coral Stick gemessen. Es konnte dabei gezeigt werden, dass das kleinste vorgestellte Netz mit 37 FPS das schnellste aller getesten ist. Allerdings kommt diese Zeit auch mit Einschränkungen in der Bildauflösung und Genauigkeit der Segmentierungen, wie es anhand von Beispielen gezeigt wurde.

Die Frage, welches das optimalste Netz ist, hängt stark vom Anwendungsfall ab. Je nachdem wie stark die Rechenkapazität ist, kann man auf eine höhere Bildauflösung zurückgreifen oder aber auch auf eine andere Architektur, welche genauere Ergebnisse liefert. Aber in dieser Arbeit konnte schlußendlich eine Palette von verschiedenen Netzen vorgestellt werden, welche alle erfolgreich Straßenmarkierungen in der Miniaturautonomie segmentieren konnten.

5.1 Ausblick

Um die Segmentierungen zu verbessern, können noch andere Segmentierungsansätze getestet werden, wie zum Beispiel PSPNet. In dieser Arbeit wurde sich auf UNet fokussiert, da es zu viele andere Ansätze gibt, um sie alle im Rahmen dieser Arbeit abzudecken. Mit anderen Ansätzen können die Scores eventuell weiter erhöht werden, um noch zuverlässigere Segmentierung zu liefern. Insbesondere kann somit versucht werden, die Genauigkeit in anderen Umgebungen zu erhöhen. Diese Problematik wurde in dieser Arbeit anhand des Mikrowunderland Datensatzes ersichtlich.

Desweiteren kann die Inferenzzeit noch weiter optimiert werden. Dabei kann versucht werden die nötigen Multiplikationen für ein Bild weiter zu senken oder andere Beschleuniger zu verwenden, welche aber dennoch Platz in einem 1:87 Fahrzeug finden sollten.

Seitens der Erklärbarkeit müssen noch weitere Methoden entwickelt werden, um sicher eine Aussage treffen zu können, inwiefern man dem Netz vertrauen kann, alles richtig zu machen. Gerade im Bereich des autonmen Fahrens sind diese Netze sicherheitskritisch und sollten dahingehend auch ausgiebig untersucht werden.

Allerdings kann man bereits mit den vorgestellten Netzen in die nächsten Schritte gehen. Dabei kann man anfangen in die Planungsebene überzugehen, um dem entgültigem Ziel, dass autonome Autos durch das Miniatur Wunderland Hamburg fahren, näher zu kommen. Diese Arbeit sollte gerade für diesen Weg eine Basis bieten.

Literaturverzeichnis

[1] Aly, Mohamed: Real Time Detection of Lane Markers in Urban Streets. In: *2008 IEEE Intelligent Vehicles Symposium* (2008), Juni, S. 7–12. – URL http://arxiv.org/abs/1411.7113. – Zugriffsdatum: 2021-12-14

[2] Badrinarayanan, V. ; Kendall, A. ; Cipolla, R.: SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39 (2017), Dezember, Nr. 12, S. 2481–2495. – ISSN 1939-3539

[3] Behrendt, Karsten ; Soussan, Ryan: Unsupervised Labeled Lane Markers Using Maps. In: *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*. Seoul, Korea (South) : IEEE, Oktober 2019, S. 832–839. – URL https://ieeexplore.ieee.org/document/9022318/. – Zugriffsdatum: 2021-10-28. – ISBN 978-1-72815-023-9

[4] Chollet, François: Xception: Deep Learning with Depthwise Separable Convolutions. In: *arXiv:1610.02357 [cs]* (2017), April. – URL http://arxiv.org/abs/1610.02357. – Zugriffsdatum: 2021-12-09

[5] He, Kaiming ; Zhang, Xiangyu ; Ren, Shaoqing ; Sun, Jian: Deep Residual Learning for Image Recognition. In: *arXiv:1512.03385 [cs]* (2015), Dezember. – URL http://arxiv.org/abs/1512.03385. – Zugriffsdatum: 2021-11-20

[6] He, Kaiming ; Zhang, Xiangyu ; Ren, Shaoqing ; Sun, Jian: Identity Mappings in Deep Residual Networks. In: *arXiv:1603.05027 [cs]* (2016), Juli. – URL http://arxiv.org/abs/1603.05027. – Zugriffsdatum: 2021-12-09

[7] Jadon, Shruti: A Survey of Loss Functions for Semantic Segmentation. In: *arXiv:2006.14822 [cs, eess]* (2020), September. – URL http://arxiv.org/abs/2006.14822. – Zugriffsdatum: 2021-04-20

[8] JUNG, Soonhong ; YOUN, Junsic ; SULL, Sanghoon: Efficient Lane Detection Based on Spatiotemporal Images. In: *IEEE Transactions on Intelligent Transportation Systems* 17 (2016), Januar, Nr. 1, S. 289–295. – ISSN 1558-0016

[9] KASTEN, Markus: *Hardwareplattform für autonome Straßenfahrzeuge im Maßstab 1:87*, HAW Hamburg, Bachelorarbeit, September 2021. – URL https://autosys.informatik.haw-hamburg.de/publication/2021markuskasten/. – Zugriffsdatum: 2021-09-27

[10] NEVEN, D. ; BRABANDERE, B. D. ; GEORGOULIS, S. ; PROESMANS, M. ; GOOL, L. V.: Towards End-to-End Lane Detection: An Instance Segmentation Approach. In: *2018 IEEE Intelligent Vehicles Symposium (IV)*, Juni 2018, S. 286–291. – ISSN 1931-0587

[11] PAN, Xingang ; SHI, Jianping ; LUO, Ping ; WANG, Xiaogang ; TANG, Xiaoou: Spatial As Deep: Spatial CNN for Traffic Scene Understanding. (2017), Dezember

[12] QIN, Zequn ; WANG, Huanyu ; LI, Xi: Ultra Fast Structure-aware Deep Lane Detection. In: *arXiv:2004.11757 [cs]* (2020), August. – URL http://arxiv.org/abs/2004.11757. – Zugriffsdatum: 2021-04-17

[13] RONNEBERGER, Olaf ; FISCHER, Philipp ; BROX, Thomas: U-Net: Convolutional Networks for Biomedical Image Segmentation. In: *arXiv:1505.04597 [cs]* (2015), Mai. – URL http://arxiv.org/abs/1505.04597. – Zugriffsdatum: 2021-12-14

[14] SANDLER, Mark ; HOWARD, Andrew ; ZHU, Menglong ; ZHMOGINOV, Andrey ; CHEN, Liang-Chieh: MobileNetV2: Inverted Residuals and Linear Bottlenecks. In: *arXiv:1801.04381 [cs]* (2019), März. – URL http://arxiv.org/abs/1801.04381. – Zugriffsdatum: 2021-12-09

[15] SCHÖNHERR, Nils: *Kamera-Basierte Minimalautonomie*, HAW Hamburg, Bachelorarbeit, Januar 2019. – URL https://autosys.informatik.haw-hamburg.de/publication/2019sch%C3%B6nherr/. – Zugriffsdatum: 2021-09-27

[16] SIMONYAN, Karen ; ZISSERMAN, Andrew: Very Deep Convolutional Networks for Large-Scale Image Recognition. In: *arXiv:1409.1556 [cs]* (2015), April. – URL http://arxiv.org/abs/1409.1556. – Zugriffsdatum: 2021-12-09

[17] SZEGEDY, Christian ; VANHOUCKE, Vincent ; IOFFE, Sergey ; SHLENS, Jonathon ; WOJNA, Zbigniew: Rethinking the Inception Architecture for Computer Vision.

In: *arXiv:1512.00567 [cs]* (2015), Dezember. – URL http://arxiv.org/abs/1512.00567. – Zugriffsdatum: 2021-12-09

[18] Taubert, Oskar ; Götz, Markus ; Schug, Alexander ; Streit, Achim: Loss Scheduling for Class-Imbalanced Image Segmentation Problems. In: *2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA)*, Dezember 2020, S. 426–431

[19] Team, Keras: *Keras Documentation: Keras Applications*. – URL https://keras.io/api/applications/. – Zugriffsdatum: 2021-11-20

[20] Zou, Qin ; Jiang, Hanwen ; Dai, Qiyu ; Yue, Yuanhao ; Chen, Long ; Wang, Qian: Robust Lane Detection from Continuous Driving Scenes Using Deep Neural Networks. In: *IEEE Transactions on Vehicular Technology* 69 (2020), Januar, Nr. 1, S. 41–54. – URL http://arxiv.org/abs/1903.02193. – Zugriffsdatum: 2021-04-21. – ISSN 0018-9545, 1939-9359

A Anhang: Trainingsverlauf

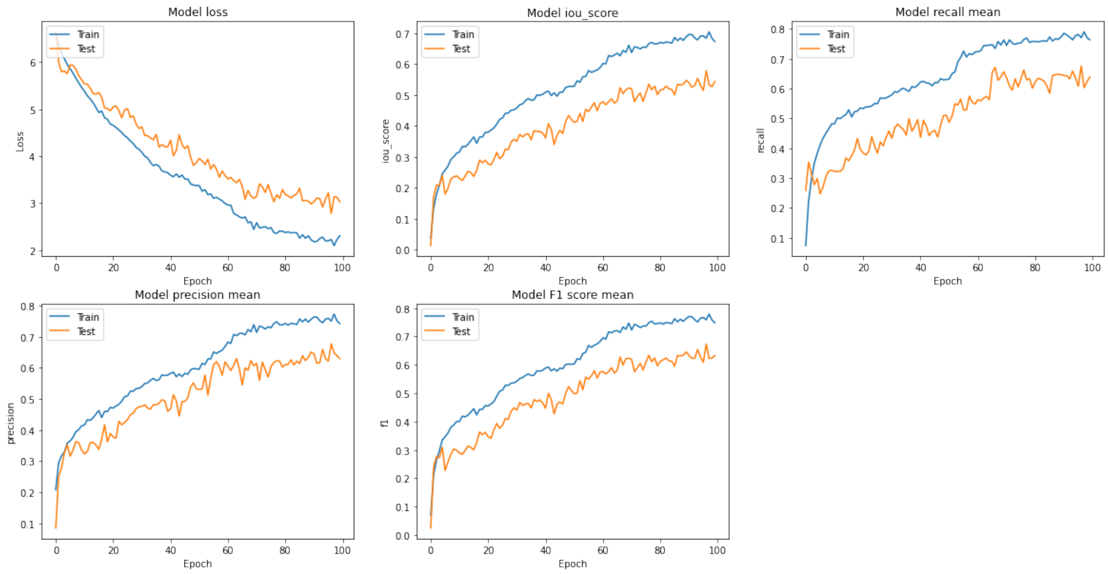


Abbildung A.1: Trainingsverlaug der MobilenetU + FT Architektur mit $\alpha = 0.6$ und $\gamma = 0.75$.

B Anhang: Segmentierungen bei Tag

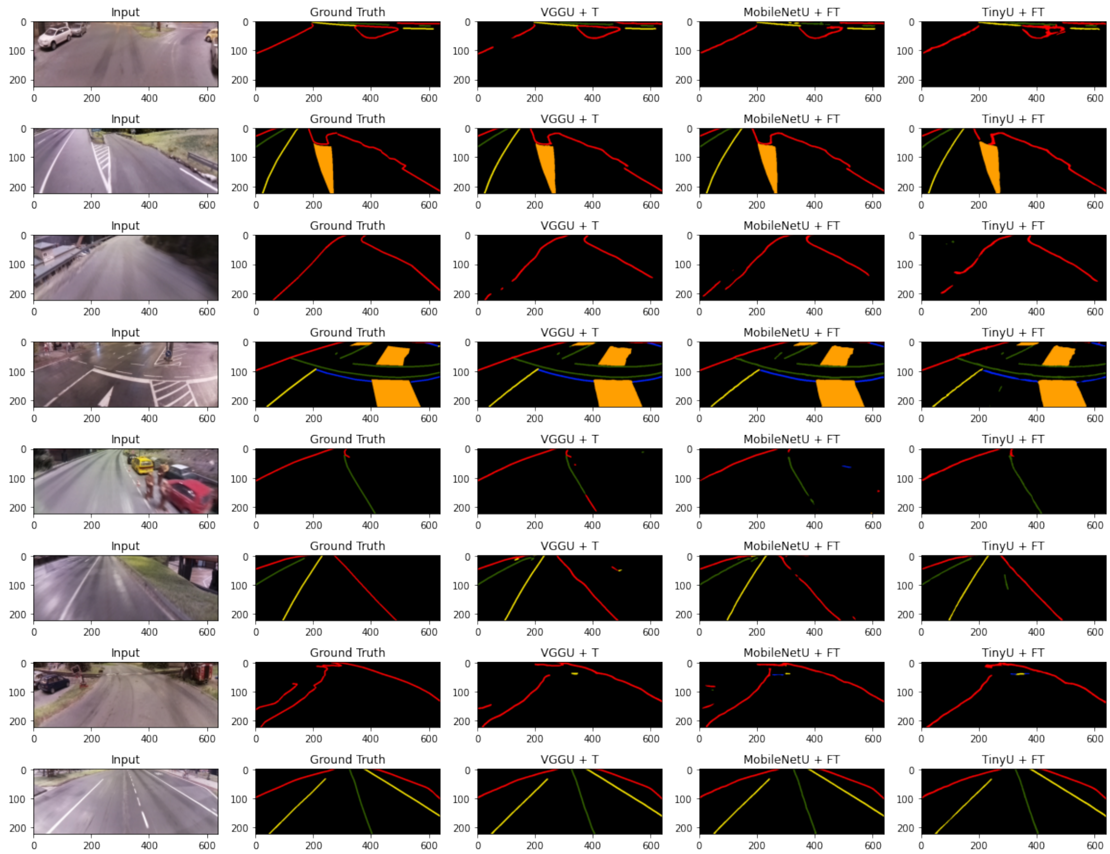


Abbildung B.1: Segmentierungen im Knuffingen bei Tag Datensatz

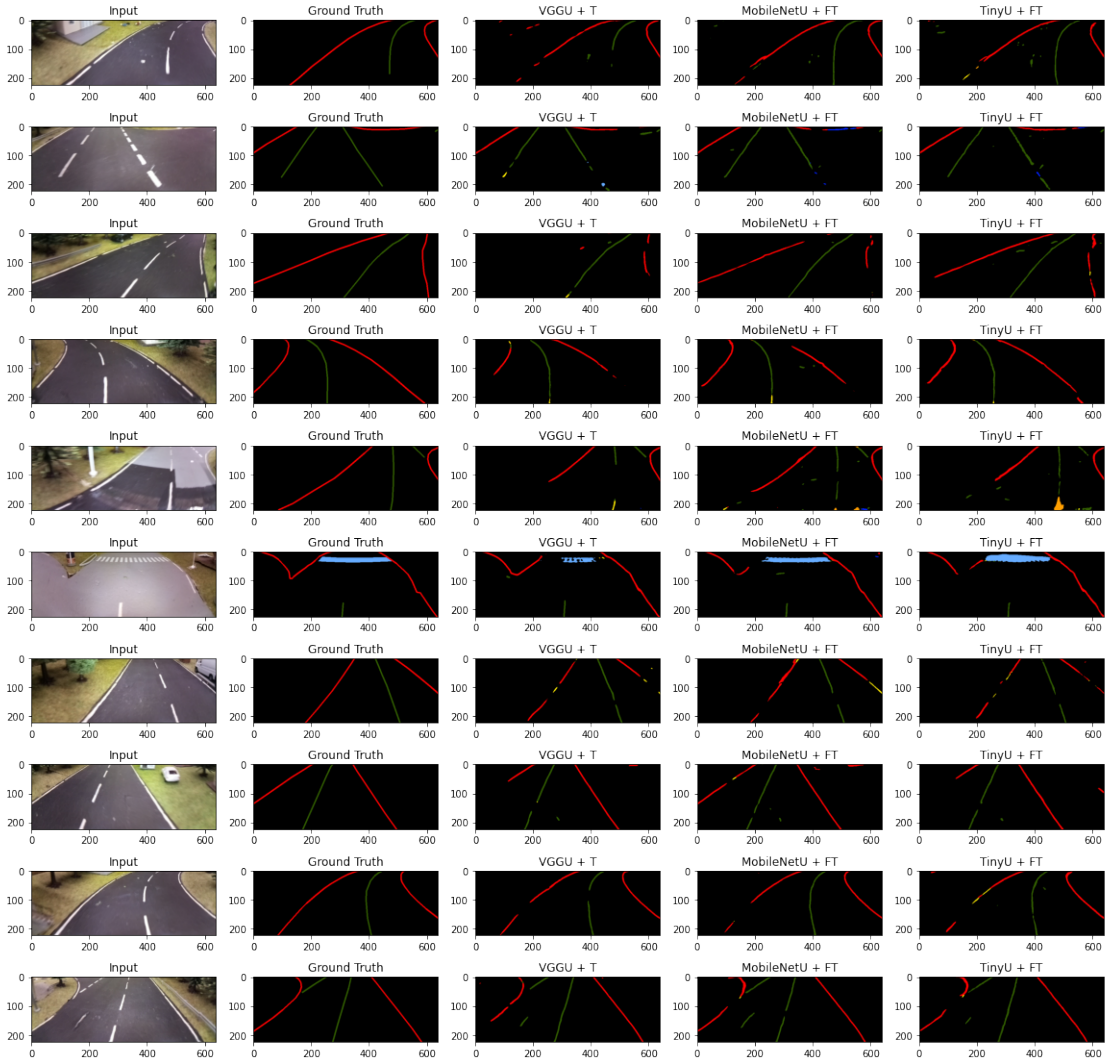


Abbildung B.2: Segmentierungen im Mikrowunderland bei Tag Datensatz

C Anhang: Segmentierungen bei Nacht

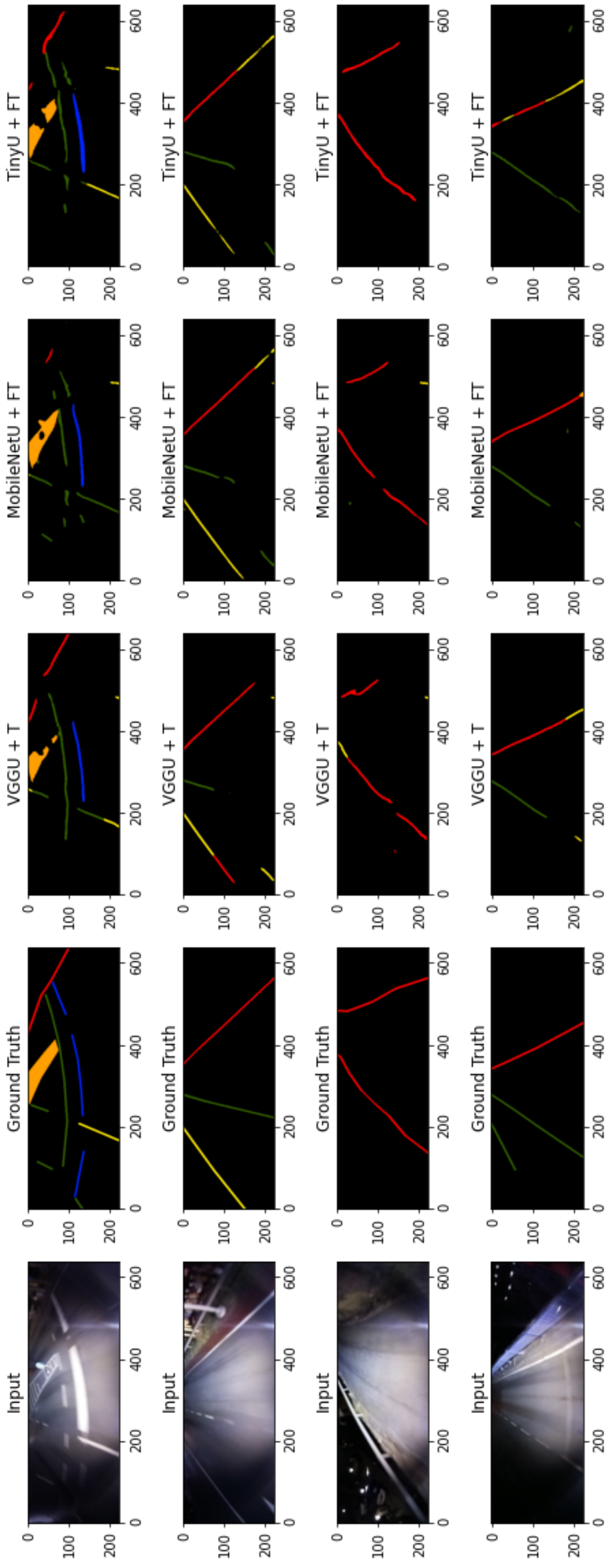


Abbildung C.1: Segmentierungen im Knuffingen bei Nacht Datensatz

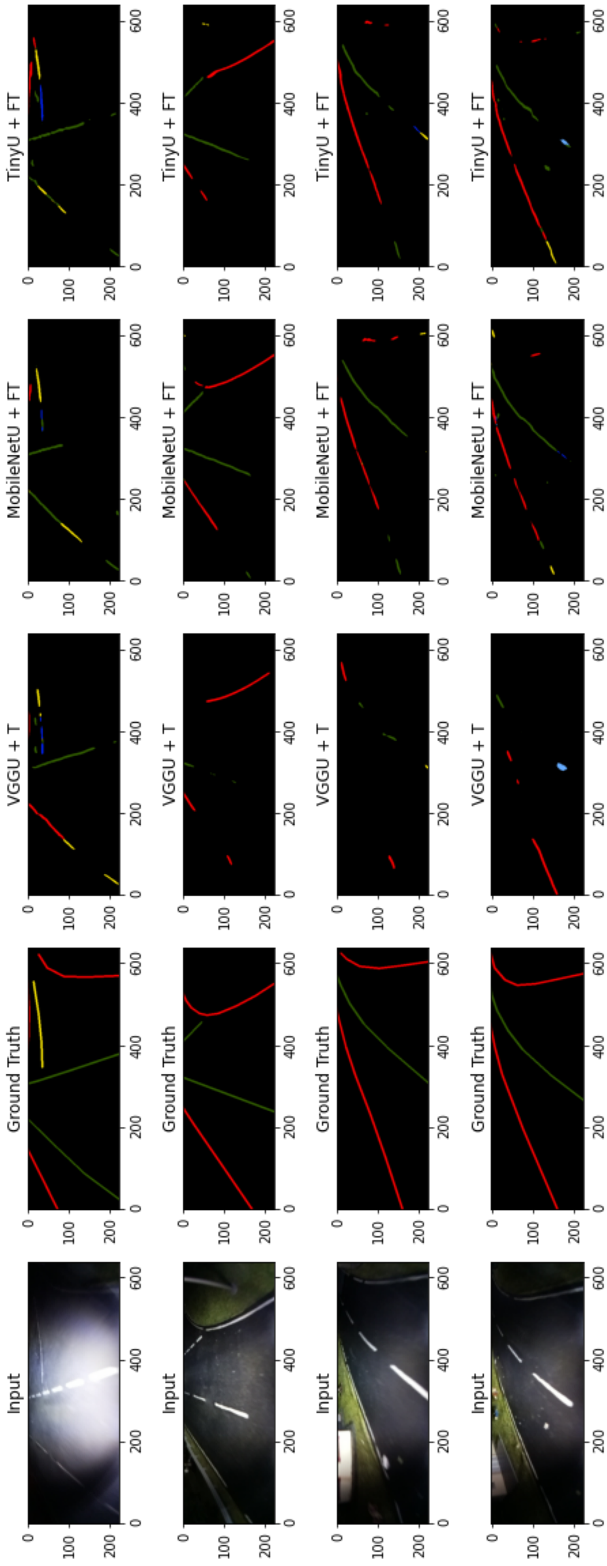


Abbildung C.2: Segmentierungen im Mikrowunderland bei Nacht Datensatz

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Gemäß der Allgemeinen Prüfungs- und Studienordnung ist zusammen mit der Abschlussarbeit eine schriftliche Erklärung abzugeben, in der der Studierende bestätigt, dass die Abschlussarbeit „— bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit [(§ 18 Abs. 1 APSO-TI-BM bzw. § 21 Abs. 1 APSO-INGI)] — ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt wurden. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen.“

Quelle: § 16 Abs. 5 APSO-TI-BM bzw. § 15 Abs. 6 APSO-INGI

Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: ____

Vorname: ____

dass ich die vorliegende Bachelorarbeit – bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit – mit dem Thema:

Segmentierung von Straßenmarkierungen durch maschinelles Lernen für die Miniaturautonomie

ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original