Freie Universität Berlin

Bachelorarbeit am Institut für Informatik der Freien Universität Berlin

Tiefeneinschätzung mit einer Monokamera im Kontext des autonomen Fahrens

Andor Zenon Amandus Kristen

Betreuer Nicolai Steinke
Erstgutachter Prof. Dr. Daniel Göhring

Berlin, 10. Januar 2024

Zusammenfassung

In vielen Systemen für das autonome Fahren werden LiDAR-Sensoren eingesetzt, um die dreidimensionale Umgebung des Fahrzeuges genau zu vermessen. Eine mögliche Alternative ist der Einsatz von Kamerasystemen, um Tiefeninformationen zu ermitteln. Neben dem klassischen Ansatz mit Stereo-Sichtsystemen besteht die Möglichkeit, aus Bildern einer Monokamera die Tiefe im Bild zu schätzen.

Im Rahmen dieser Arbeit werden zu diesem Zweck zwei Ansätze getestet. Es wird ein künstliches neuronales Netz nach der Referenz von Hu et al. [14] implementiert und auf einem selbsterstellten Datensatz aus Kamerabildern und LiDAR-Daten des „Made in Germany“-Fahrzeuges trainiert. Um fehlende Daten im Himmelbereich auszugleichen, wurde zusätzlich eine Himmelerkennung implementiert. Es werden unterschiedliche Backbones und eine leichte Modifikation der Architektur getestet. Ein zweiter Ansatz nutzt den optischen Fluss einer Bildsequenz und die bekannte Kamerabewegung, um die Distanzen im Bild zu berechnen. Beide Ansätze werden auf dem selbsterstellten Datensatz evaluiert. Das künstliche neuronale Netz liefert dabei deutlich bessere Ergebnisse mit einem mittleren absoluten relativen Fehler von 17 %.

Danksagung

Für die Betreuung der Arbeit möchte ich mich bei Prof. Dr. Daniel Göhring bedanken. Besonderer Dank gilt auch Nicolai Steinke, der mich in die Softwareumgebung eingearbeitet hat und mich während der Arbeit mit Feedback, Ideen und Verbesserungsvorschlägen unterstützt hat.

Für die zum Trainieren des in der Arbeit genutzten neuronalen Netzes benötigten Rechenressourcen und Rechenzeit möchte ich mich herzlich beim HPC-Dienst der ZEDAT, Freie Universität Berlin bedanken.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Die Erfassung der dreidimensionalen Umgebung ist ein entscheidendes Problem bei der Entwicklung von Systemen zum autonomen Fahren. Um eine sichere Fahrt zu gewährleisten, sind Informationen wie der Abstand zu anderen Verkehrsteilnehmern und Hindernissen absolut notwendig. In vielen modernen Systemen für das autonome Fahren werden zu diesem Zweck LiDAR-Sensoren eingesetzt. Diese können durch das Ausstrahlen von Laserimpulsen die Entfernung zu anderen Objekten messen und so eine Repräsentation der dreidimensionalen Umgebung erstellen. Eine mögliche Alternative für die Ermittlung von Tiefeninformationen ist die Nutzung von Kamerasystemen. Ein wichtiger Vorteil gegenüber LiDAR-Sensoren sind die stark reduzierten Kosten [33]. Es stellt sich jedoch die Frage, ob die aus Kamerasystemen gewonnenen Tiefeninformationen die Genauigkeit einer LiDAR-Messung erreichen können.

Der klassische Ansatz zur Extraktion von Tiefeninformationen aus Bildern sind sogenannte Stereo-Sichtsysteme. Dabei wird, ähnlich wie beim stereoskopischen Sehen beim Menschen, eine Szene durch zwei Kameras aufgenommen. Durch die Verschiebung von Objekten in beiden Bildern kann die Entfernung dieser Objekte errechnet werden [27]. Ein alternativer Ansatz ist, die Tiefeninformationen aus Einzelbildern einer Monokamera abzuleiten. Im Gegensatz zur Stereosicht reicht es hierbei nicht aus, im lokalen Umfeld auf Disparität zu prüfen, sondern die Szene muss in einen globalen Kontext gesetzt werden. Dafür müssen Bildelemente wie Perspektive, Fluchtpunkte und Referenzobjekte in der Szene berücksichtigt werden.

Eine frühe Arbeit von Hoiem et al. zu diesem Thema beruht auf der Idee, das Bild in die Regionen “ground”, “sky”, und “vertical” zu segmentieren und mit diesen Informationen ein simples geometrisches Modell der Szene zu erstellen [13]. Einen weiteren Ansatz zur Tiefenschätzung aus einem Bild lieferten Karsch et al., welche mit einer Nächsten Nachbar Klassifikation und einem Datensatz aus bekannten Bild-Tiefenbild-Paaren die Tiefe auf neuen, unbekannten Bildern schätzten [16]. Die rapide Entwicklung von künstlichen neuronalen Netzen in den letzten Jahren eröffnete aber eine weitere Möglichkeit der monokularen Tiefenschätzung: durch eine direkte Regression der Tiefe aus dem Kamerabild. Eine der ersten Veröffentlichungen zu diesem Thema von Eigen et al. [7] erreichte 2014 neue Bestwerte auf den NYU Depth V2 [22] und KITTI-Depth [32] Datensätzen. Fortschritte bei der Architektur von neuronalen Netzen, wie etwa die ResNet-Architektur [12], können meist auf Architekturen für die Tiefenschätzung übertragen werden und führen zu Verbesserungen der Vorhersagen [20]. Weiterhin können optimierte Verlustfunktionen, die die Skaleninvarianz aufweisen oder die Struktur der Szene besser bewerten, zu verbesserten Ergebnissen beitragen [6].

Ergebnisse zeigen allerdings auch, dass Modelle, die auf anderen Datensätzen trainiert wurden, nicht immer gleich gute Ergebnisse auf neuen Datensätzen liefern [23]. Um die Anwendbarkeit der monokularen Tiefenschätzung mit neuronalen Netzen zu prüfen, sollte die „Performance“ der Modelle also auf einem Datensatz getestet werden, welcher dem Anwendungsfall entspricht.

In dieser Arbeit wird die Anwendbarkeit der Tiefenschätzung mit einer Kamera anhand des „Made in Germany“-Fahrzeugs des Dahlem Center for Machine Learning and Robotics getestet. Zu diesem Zweck wird ein Datensatz aus Kamera- und LiDAR-Daten von aufgezeichneten Fahrten erstellt. Es wird ein bestehendes Modell zur Tiefenschätzung aus Einzelbildern implementiert und auf diesem Datensatz trainiert. Weiterhin wird prototypisch ein Ansatz zum Berechnen der Tiefe aus einer Bildsequenz unter Nutzung des optischen Flusses und der bekannten Eigenbewegung der Kamera getestet. Beide Ansätze werden auf Daten des Fahrzeugs evaluiert und die Anwendbarkeit für das autonome Fahren geprüft. Ziel ist es dabei, neben einer möglichst hohen Genauigkeit der Tiefenschätzung, auch eine akzeptable Wiederholungsrate der Schätzung zu

erreichen. Der angestrebte Richtwert sind 10 Hz, da dies der Wiederholungsrate des genutzten LiDAR-Scanners entspricht.

2 Theoretische Grundlagen

2.1 Kameras & Kameraprojektion

Durch eine Kamera wird eine dreidimensionale Szene in eine zweidimensionale Repräsentation, das Bild, überführt. Dabei werden die Lichtstrahlen, die von Objekten reflektiert oder ausgestrahlt werden, auf eine Fläche, die Bildebene, projiziert. Befindet sich auf dieser Fläche ein lichtempfindliches Material, so reagiert dieses auf die Lichteinstrahlung, wodurch die momentane Projektion „eingefangen“ wird. Bei digitalen Kameras werden typischerweise elektronische Lichtsensoren in einer gitterförmigen Anordnung verwendet. Jeder Sensor liefert bei einer Aufnahme ein Signal abhängig von der Intensität des einfallenden Lichtes. Die Werte dieser diskreten Abtastpunkte bilden die Werte der Pixel des digitalen Bildes [29].

Das einfachste Modell zur Beschreibung der Funktionsweise einer Kamera ist das sogenannte Lochkamera-Modell. Bei diesem durchläuft der Lichtstrahl ein unendlich kleines Loch und trifft dahinter auf die Bildebene. Die Lichtstrahlen bewegen sich dabei linear fort und es gibt keine Verzeichnung durch eine Linse. Daraus resultiert eine wichtige Eigenschaft von Bildern einer Lochkamera, die Unverzerrtheit. Geraden im dreidimensionalen Raum werden auch als Geraden auf der Bildebene dargestellt [30].

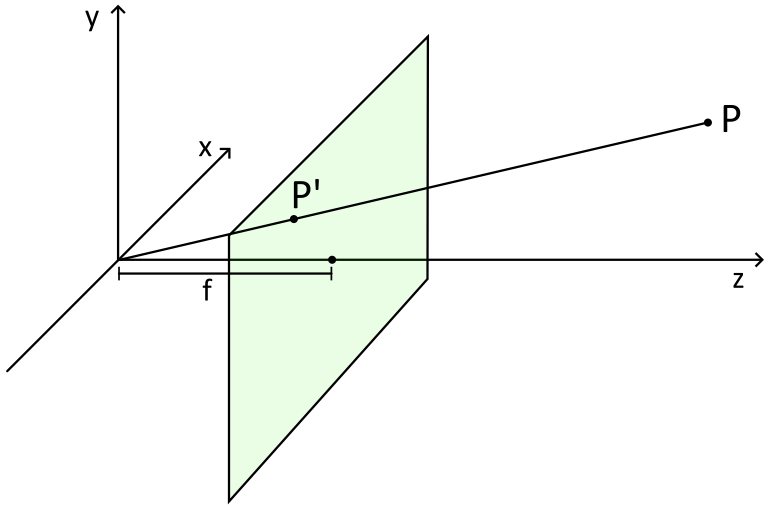


Abbildung 1: Schematische Darstellung des Lochbildkamera-Modells. Der Punkt P im dreidimensionalen Raum wird auf den Punkt P’ in der Bildebene projiziert. Diese befindet sich im Abstand f zum Bündelungspunkt.

Ein mathematisches Schema der Lochkamera ist in Abbildung 1 dargestellt. Für die mathematische Beschreibung wird die Bildebene vor dem Bündelungspunkt und nicht hinter diesem angenommen, da auf der Bildebene hinter dem Bündelungspunkt das Bild gespiegelt ist. Weiterhin wird eine Konvention für das Koordinatensystem der Kamera benutzt, bei der die z-Achse die Blickrichtung der Kamera beschreibt (abhängig von der Konvention ist die positive z-Richtung in Blickrichtung oder genau entgegengesetzt). Sie wird auch optische Achse genannt. Die x-Achse ist die Breite im Bild und die y-Achse die Höhe. Der Nullpunkt des Kamerakoordinatensystems liegt im Bündelungspunkt [30].

Die Breite und Höhe der Projektion eines Punktes auf die Bildebene im Verhältnis zur Bildweite f ist identisch zur x- bzw. y-Koordinate im Verhältnis zur z-Koordinate im Kamerakoordinatensystem: $\frac{x'}{f} = \frac{x}{z}$ und $\frac{y'}{f} = \frac{y}{z}$. Diese Identität lässt sich anhand der ähnlichen Dreiecke in Abbildung 2 herleiten.

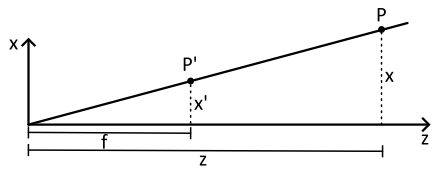


Abbildung 2: Zweidimensionale Darstellung des Zusammenhanges der Position des Punktes im dreidimensionalen Raum, dem Bildabstand f und der Position des projizierten Punktes auf der Bildebene.

Alternativ kann die Position auf der Bildebene durch Multiplikation des 3D-Punktes mit einer 3x3-Projektionsmatrix berechnet werden. In der Literatur werden häufig 3x4-Matrizen angegeben, da der Punkt noch in Kamerakoordinaten umgerechnet wird und dafür typischerweise eine Translation nötig ist. Bei diesen Matrizen handelt es sich um die verrechneten Matrizen zur Umrechnung in Kamerakoordinaten und der Projektionsmatrix. Die erhaltenen Koordinaten sind die homogene Repräsentation der 2D Koordinaten auf der Bildebene. Durch Division durch die z-Komponente kann die Position auf der Bildebene errechnet werden. Es ist auch möglich, mit der Projektionsmatrix direkt die Pixelkoordinaten auszurechnen. Dafür wird eine Translation benötigt, da der Nullpunkt in Pixelkoordinaten die obere linke Bildecke ist.

$$\begin{bmatrix} x' \cdot z + u \cdot z \\ y' \cdot z + v \cdot z \\ z \end{bmatrix} = \begin{bmatrix} f & 0 & u \\ 0 & f & v \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Reale Kameras sind typischerweise keine idealen Lochkameras. Die in den Objektiven verbauten Linsen führen zu einer Verzeichnung des Bildes, der sogenannten radialen Verzeichnung. Zusätzlich kann es zu sogenannten tangentialen Verzeichnungen kommen, wenn die Linse nicht parallel zur Bildebene steht. Die zwei gängigsten Arten der radialen Verzeichnung sind die tonnenförmige und kissenförmige Verzeichnung, welche entstehen, wenn die Bildränder schwächer bzw. stärker vergrößert werden als das Bildzentrum [5]. Sie sind in Abbildung 3 dargestellt. Die tonnenförmige Verzeichnung ist typisch für Weitwinkelobjektive, die kissenförmige Verzeichnung tritt typischerweise bei Teleobjektiven auf.

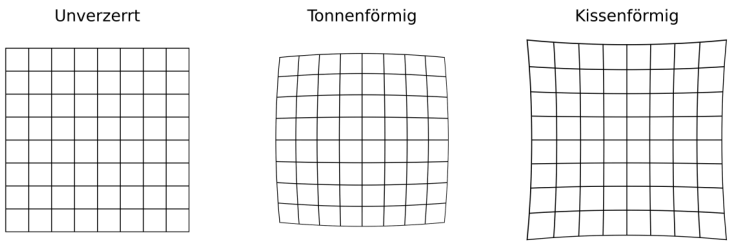


Abbildung 3: Darstellung der typischen radialen Verzeichnung. Links: ohne Verzeichnung, mitte: tonnenförmige Verzeichnung, rechts: kissenförmige Verzeichnung

Die durch die Verzeichnung auftretende Abweichung vom Lochkamerabild kann durch eine Funktion approximiert werden. Diese ordnet den Pixeln im verzeichneten Bild eine Position im begradigten Bild zu. Die Art der Funktion hängt von der Art und Stärke der Verzeichnung ab, typischerweise werden Polynomfunktionen oder gebrochen rationale Funktionen verwendet. Da die Linsen meist symmetrisch geschliffen sind, sind die Korrekturfunktionen rotationssymmetrisch um das Verzeichnungsszentrum [3, 5].

2.2 Optischer Fluss

Der optische Fluss beschreibt die Bewegung von Objekten in einem Bild in Bildkoordinaten. Für zwei zeitlich aufeinanderfolgende Bilder ist er definiert als das Vektorfeld der in die Bildebene projizierten Bewegungsvektoren der sichtbaren Elemente im Kamerakoordinatensystem zwischen den Aufnahmezeitpunkten. Ein Beispiel ist in Abbildung 4 dargestellt. Dabei ist anzumerken, dass der optische Fluss nicht durch die Bewegung der Objekte im Bild entstehen muss, sondern auch die Eigenbewegung der Kamera einen optischen Fluss auslösen kann.

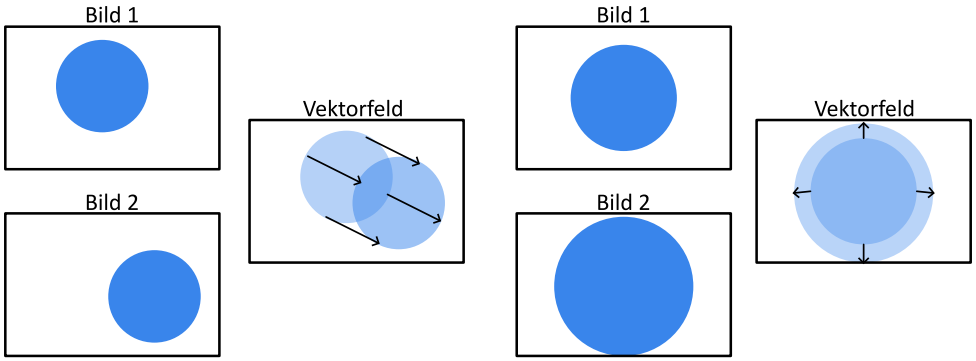


Abbildung 4: Darstellung der Bewegung eines Objektes, hier schematisch als blauer Kreis, in zwei aufeinanderfolgenden Bildern und dem daraus resultierenden zweidimensionalen Vektorfeld, dem optischen Fluss. Links für eine Bewegung parallel zur Bildebene, rechts für eine Bewegung parallel zur Sichtachse. Bei der Darstellung des Vektorfeldes des optischen Flusses sind die blauen Kreise nur zur Veranschaulichung gezeigt.

Bereits aus dem optischen Fluss lassen sich Informationen über die Positionierung von Objekten in der Szene feststellen. Bewegen sich etwa zwei Objekte gleich schnell parallel zur Bildebene, so erzeugt ein näheres Objekt einen größeren optischen Fluss. Bewegt sich ein Objekt auf die Kamera zu, so divergieren die entstehenden Bewegungsvektoren.

Den optischen Fluss aus einem Bildpaar zu bestimmen ist eine komplexe Aufgabe, die ohne vereinfachende Annahmen mathematisch nicht lösbar ist. Eine typische Annahme zur Berechnung ist, dass die Belichtung in beiden Aufnahmen identisch ist bzw., dass der Farbwert eines Pixels in Bild 1 identisch zum Farbwert eines Pixels in Bild 2 ist, wobei die Verschiebung der Pixelposition den optischen Fluss darstellt. Weiterhin seien die Geschwindigkeiten u und v in x- und y-Richtung klein. Definieren wird $I(x, y, t)$ als dem Helligkeits- bzw. Farbwert des Bildes an der Position x, y zum Zeitpunkt t gilt:

$$I(x, y, t) = I(x + u\delta x, y + v\delta y, t + \delta t)$$

Ausgehend von dieser Formel lässt sich durch Taylor-Approximation erster Ordnung und Ableitung nach der Zeit die Optical Flow Constraint Equation aufstellen [18].

$$0 = \frac{\partial I}{\partial x}\frac{dx}{dt} + \frac{\partial I}{\partial y}\frac{dy}{dt} + \frac{\partial I}{\partial t} = \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v + \frac{\partial I}{\partial t}$$

Dabei sind u und v die gesuchten Bewegungskomponenten in x- bzw. y-Richtung. Die partiellen Ableitungen werden durch die Bildung des Gradienten im ersten Bild bzw. durch die Differenz des zweiten zum ersten Bild errechnet. Die Optical Flow Constraint Equation ist in dieser Form nicht lösbar, da für jedes Pixel zwei unbekannte Faktoren u und v mit nur einer Gleichung bestimmt sind. Daher sind weitere Annahmen nötig. In der Veröffentlichung von Lucas und Kanade [21] wird das Bewegungsfeld in einem kleinen Fenster um das Pixel als konstant angenommen, wodurch das Gleichungssystem überbestimmt ist. Das Gleichungssystem wird dann durch die Methode der kleinsten Quadrate optimiert. Der in Abschnitt 3.2.1 genutzte Robust Local Optical

Flow (RLOF) [28] ist eine Weiterentwicklung des Ansatzes von Lucas und Kanade. Unter anderem wird statt der Minimierung des quadratischen Fehlers die Hampel-Norm genutzt, die gegenüber starken Ausreißern weniger sensitiv ist. Dies führt in Bereichen, in denen der optische Fluss im Pixelumfeld nicht gleichmäßig ist, zu besseren Ergebnissen.

Ein Problem der oben aufgestellten Optical Flow Constraint Equation ist, dass sie aufgrund der linearen Näherung nur in einem kleinen Pixelumfeld gültig ist und der optische Fluss so nur für ein kleines Pixelumfeld bestimmt werden kann. Daher werden typischerweise Bildpyramiden genutzt. Dabei wird zuerst der optische Fluss auf einem stark verkleinerten Bild berechnet. Dies liefert eine grobe Einschätzung des optischen Flusses unter Berücksichtigung größerer Bereiche im ursprünglichen Bild. In darauffolgenden Schritten wird immer das nächst höher aufgelöste Bild der Bildpyramide gelöst, wobei die vorher berechneten Bewegungsvektoren mit einbezogen werden.

2.3 Machine Learning

Maschinelles Lernen, auch Machine Learning genannt, ist ein Teilgebiet der Künstlichen Intelligenz, in dem das Lernen aus zugrundeliegenden Daten im Fokus steht. Dabei werden statistische Modelle entworfen und genutzt um Aufgaben zu lösen, ohne explizit für die Aufgabe programmiert worden zu sein. Das Ziel ist dabei nicht das Auswendiglernen der für das Lernen genutzten Daten, sondern die Entwicklung eines Modells, dass möglichst gut die Realität abbildet, also auf der Grundgesamtheit der Daten gute Ergebnisse liefert.

Ein Teilgebiet des maschinellen Lernens ist das sogenannte überwachte Lernen, auch supervised learning genannt. Das Ziel des überwachten Lernens ist es, eine Funktion zu finden, die einem Eingabedatum einen bestimmten Ausgabewert zuordnet. Diese Zuordnung wird dabei anhand eines Datensatzes trainiert, der aus Paaren von Eingabewerten und gewünschten Ausgaben, die Labels genannt werden, besteht. Das überwachte Lernen kann in zwei Unterkategorien unterteilt werden: die Regression und die Klassifikation. Der Unterschied liegt in der Art der Ausgabe. Bei der Regression wird ein kontinuierlicher Wert, meistens ein Zahlenwert, vorhergesagt. Bei der Klassifikation gibt es hingegen eine endliche Menge an möglichen Ausgaben, die Klassen, denen die Eingabe zugeordnet werden soll.

Um maschinelles Lernen überwacht zu trainieren, muss zuerst eine geeignete Repräsentation der Eingaben und Labels gefunden werden. Anschließend wird ein Modell ausgewählt, dass die Zuordnungsfunktion repräsentiert. Das Modell wird dann trainiert. Dabei werden üblicherweise Eingabedaten an das Modell gegeben und die erhaltene Ausgabe (Vorhersage) wird mit den zugehörigen Labels verglichen. Die Abweichung zwischen der Vorhersage und dem Label wird durch eine Verlustfunktion (auch Verlustmetrik oder loss function) bestimmt und die Parameter des Modells werden angepasst, um die Abweichung der Vorhersage zum Label zu minimieren. Das Ziel ist aber nicht, die Trainingsbeispiele perfekt zu lernen, sondern auf vorher ungesehenen Daten bestmögliche Vorhersagen zu treffen. Nach dem Training eines Modells kann daher ein vorher ungesehener Datensatz mit vorhandenen Labels, das Test Set, genutzt werden, um die Vorhersagequalität des Modells einzuschätzen [11].

Die Wahl des Modells hat einen entscheidenden Einfluss auf die Qualität der Ergebnisse und auf den Trainingsprozess. Da im Rahmen dieser Arbeit mit neuronalen Netzen gearbeitet wird, werden nur diese hier genauer erläutert.

2.3.1 Künstliche neuronale Netze

Künstliche neuronale Netze, im weiteren Text auch neuronale Netze genannt, sind eine weit verbreitete und vielseitige Art von Machine Learning Modellen. Sie sind lose an der Funktionsweise von Neuronen in Tieren angelehnt, es gibt aber einige Unterschiede zur biologischen Referenz. Das Grundelement von neuronalen Netzen bildet das Neuron. Es erhält einen oder mehrere numerische Eingabewerte, die jeweils mit einem Gewichtungsfaktor (weight) multipliziert werden. Zusätzlich wird ein optionaler Summand (bias) hinzuaddiert. Die Summe über alle Eingabewerte wird gebildet und fließt in eine optionale Aktivierungsfunktion ein, deren Ergebnis die Ausgabe des Neurons ist [10].

Die Neuronen werden in Schichten angeordnet. Ein Neuron in einer Schicht bekommt als Eingaben die Werte eines oder mehrerer Neuronen der vorherigen Schicht und die Ausgabe wird an die Neuronen der nächsten Schicht weitergegeben. Die erste Schicht bildet die Eingabewerte und wird daher Eingabeschicht (input layer) genannt, die letzte Schicht bildet die Ausgabe des Netzes und wird daher Ausgabeschicht (output layer) genannt. Die dazwischenliegenden Schichten werden als verborgene Schichten (hidden layers) bezeichnet. Das einfachste Schema eines neuronalen Netzes ist das dichte neuronale Netz. Hier erhält jedes Neuron als Eingaben alle Werte der Neuronen der vorhergehenden Schicht. Ein solches Netz ist exemplarisch in Abbildung 5 abgebildet.

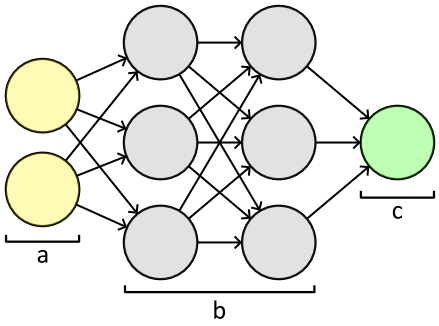


Abbildung 5: Ein einfaches dichtes neuronales Netz, bestehend aus einer Eingabeschicht mit zwei Eingabeneuronen (a), zwei verborgenen Schichten mit jeweils drei Neuronen (b) und der Ausgabeschicht mit einem Neuron (c).

Die Neuronen und ihre Verbindungen bilden einen gerichteten Graphen. Wenn dieser azyklisch ist, dann spricht man von einem Feedforward-Netz, da die Ausgaben der Neuronen nur in eine Richtung fließen. Gibt es Zyklen im Graph, so handelt es sich um ein rekurrentes neuronales Netz. Dies ermöglicht es, die Ausgabe abhängig von vorhergegangenen Eingaben zu verändern.

Bei der Initialisierung eines neuronalen Netzes werden die Gewichte und biases zufällig initialisiert, daher entspricht die Ausgabe nicht dem gewünschten Resultat. Das Training des neuronalen Netzes läuft in Teilschritten ab. In jedem Schritt wird ein Teil des Datensatzes als Eingabe an das neuronale Netz gegeben und die Ausgabe des Netzes wird ermittelt. Dieser Schritt nennt sich forwardpass. Die Ausgabe wird mit dem Label in der Verlustfunktion verrechnet. In der Backpropagation wird nun der Gradient des Fehlers in Bezug auf alle Parameter des Netzes gebildet und die Parameter werden anhand der Optimierungsstrategie verändert [10].

Für die Verarbeitung von Bildern bieten sich dichte neuronale Netze nicht an, da die Anzahl der Parameter mit wachsender Zahl von Neuronen stark ansteigt. Sind zwei Schichten eines neuronalen Netzes dicht verbunden und haben n und m Neuronen, so werden $2 \cdot n \cdot m$ Parameter für die Verbindungen benötigt. Stattdessen werden sogenannte Convolutional Layers genutzt, in denen die Neuronen nur mit einer begrenzten Anzahl an Neuronen der vorhergehenden

Schicht verbunden sind. Eine Schicht wird dabei als zwei- oder mehrdimensional interpretiert. Die Ausgabe einer Schicht ist die Kreuzkorrelation eines Filters, auch Kernel genannt, mit der vorhergehenden Schicht. Wenn die Eingabeschicht mehrere Kanäle besitzt, werden ebenso viele Filter benötigt und die Ausgabe ist die Summe über die Kreuzkorrelationen der verschiedenen Kanäle. Wenn eine Schicht mehrere Kanäle ausgeben soll, dann werden so viele verschiedene Filter wie die Anzahl von Ausgabekanälen gebildet. Jeder Wert eines Kanals einer Convolutional Layer wird mit demselben Filter gebildet, aber der Eingabebereich aus der vorherigen Schicht unterscheidet sich. Diese Struktur ist in Abbildung 6 dargestellt.

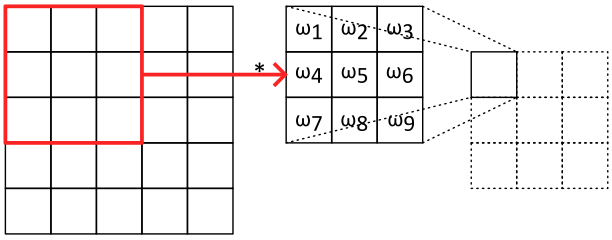


Abbildung 6: Darstellung der Berechnung der Ausgabe eines Wertes einer 2D Convolutional Layer mit einem 3x3-Kernel. Die Gewichte ω_1-ω_9 sind für die Berechnung jedes Ausgabewertes des Kanals identisch, der rot markierte Quellbereich wird aber verschoben.

Einige wichtige Parameter für die Beschreibung von Convolutional Layers sind die Kernelgröße K, Stride S, Padding P und die Anzahl der Ausgabekanäle C. Die Kernelgröße bestimmt die Dimension des Filters, er legt also den Bereich fest, den jedes Neuron aus der vorhergehenden Schicht als Eingabe erhält. Dieser Bereich wird das rezeptive Feld genannt. Der Stride legt die Schrittweite fest, um die der Filter verschoben wird, um den nächsten Wert zu berechnen. Standardmäßig ist dieser Wert eins, er kann aber erhöht werden, um eine Verkleinerung der Dimension der Ausgabe zu erhalten. Das Padding legt fest, ob die Eingabe um eine feste Breite erweitert wird, sodass die Ausgabe dieselbe Größe wie die Eingabe hat. Der Grund dafür ist, dass die Ausgabe um die Kernelgröße kleiner ist als die Eingabe, da der Kernel nicht über den Rand der Eingabe hinweggehen kann [4]. Die Dimension O der Ausgabe lässt sich bei bekannter Eingabedimension I wie folgt berechnen:

$$O_x = (I_x - K_x + 2P_x)/S_x + 1$$

Die Berechnung ist analog für die y-Dimension der Ausgabe.

Neben Convolutional Layern werden in convolutional neuronalen Netzen auch Pooling Layer genutzt. Bei diesen wird, ähnlich wie bei Convolutional Layern, ein Fenster bestimmter Größe über die Eingabe geschoben, dabei werden die Werte aber nicht mit trainierbaren Gewichten multipliziert, sondern mit einer Operation gefiltert. Typisch sind die MaxPool und AveragePool Operationen, die das Maximum oder den Durchschnittswert des Eingabebereichs ermitteln. Analog zu Convolutional Layern kann die Filtergröße, Schrittweite und das Padding bestimmt werden. Die Pooling Layer dient normalerweise zur Reduzierung der Auflösung einer Eingabe [4].

2.3.2 Encoder-Decoder Architektur

Die Encoder-Decoder-Architektur ist eine allgemeine Architektur für verschiedene Arten von neuronalen Netzen und findet in vielen Bereichen wie der Verarbeitung natürlicher Sprache und Bildverarbeitung, aber auch beim unüberwachten Lernen in der Form von Autoencodern Anwendung. Die Grundidee der Architektur ist dabei, dass der Enocoder eine Eingabe kodiert,

indem er für die Aufgabe wichtige Features aus der Eingabe ermittelt. Der Decoder arbeitet auf der kodierten Form der Eingabe und erstellt daraus die gewünschte Ausgabe.

Im Kontext der Bildverarbeitung bedeutet das Kodieren, dass durch wiederholte Anwendung von Convolution- und Poolingoperationen mit Schrittgröße größer als eins, die Dimension des Bildes kontinuierlich verkleinert wird. Bei der Verringerung der Dimension werden im Gegenzug üblicherweise mehr Kanäle gespeichert, um verschiedene Features darzustellen. Ein typisches Schema ist es, mit einer Aneinanderreihung von identischen Blöcken zu arbeiten. Hierbei wird nach einer festgelegten Anzahl an Blöcken die Dimension der Eingabe um den Faktor ½ verringert. Bei der Dekodierung wird die Auflösung schrittweise wiederhergestellt, z.B. durch eine UpConvolution. Im Rahmen dieser Arbeit wird mit zwei Backbones für den Encoderteil gearbeitet, ResNet50 und SENet154.

Bei der ResNet-Architektur kommen sogenannte ResidualBlocks zum Einsatz. Diese zeichnen sich dadurch aus, dass die Eingabe des Blocks zur Ausgabe hinzuaddiert wird. Es entstehen also zwei Pfade zur Ausgabe des Blocks, der eine überspringt aber die im Block vorgenommenen Operationen. Die direkte Verbindung der Eingabe zur Ausgabe wird skip-connection genannt. Die Idee hinter der Addition der Eingabe ist es, dass der Block für die Eingabe x anstatt der Funktion $H(x)$ die Residualfunktion $F(x) = H(x) - x$ approximiert. Dies ist für die Fälle einfacher, in denen $H(x)$ ähnlich der Eingabe x ist, also eine annähernde Identitätsfunktion darstellt [12].

Ein einfacher ResidualBlock besteht aus zwei Convolutional Layern mit 3x3 Kernel, Zero-Padding und Stride von 1. Falls erforderlich, wird die Dimensionsreduzierung durch eine Schrittweite von 2 bei der Convolution erreicht, gleichzeitig wird die Anzahl der Featuremaps verdoppelt. Auf die Ausgabe der ersten Convolutional Layer und die Ausgabe des Blocks nach der Addition der Eingabe wird zusätzlich eine ReLU-Aktivierungsfunktion angewandt. Die Ausgaben aller Convolutional Layer werden mit BatchNormalization vor Anwendung der Aktivierungsfunktion normalisiert. Neben dem normalen ResidualBlock, der in ResNet-34 genutzt wird, gibt es noch den BottleneckResidualBlock. Dieser wird in größeren ResNet-Strukturen, unter anderem auch ResNet50, genutzt. Er unterscheidet sich vom einfachen ResidualBlock dadurch, dass anstelle von zwei stattdessen drei Convolutional Layer im Block vorhanden sind, wobei die erste und dritte Schicht einen 1x1 Kernel besitzen. Weiterhin geben die ersten zwei Schichten nur ein Viertel der Anzahl der Featuremaps der Eingabe aus. Dadurch wird die Anzahl der zu lernenden Parameter stark reduziert. Die letzte Schicht erzeugt wieder die ursprüngliche Anzahl an Featuremaps, um mit der Eingabe kompatibel zu sein. Wenn eine Dimensionsreduzierung nötig ist, wird die Schrittweite in der zweiten Schicht erhöht. Beide Blockstrukturen sind in Abbildung 7 abgebildet.

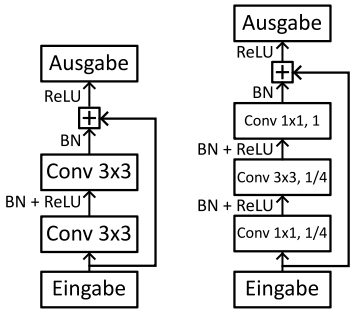


Abbildung 7: Vergleich der Struktur eines einfachen ResidualBlocks bestehend aus zwei Schichten (links) mit einem BottleneckResidualBlock (rechts). Die Eingabe wird zur Ausgabe der letzten Schicht über eine skip-connection addiert.

Aufbauend auf einem Encoder Block wie beispielsweise dem ResidualBlock kann ein Squeeze-and-Excitation-Block (SEBlock) definiert werden. Die Struktur des eigentlichen Blockes bleibt

dabei erhalten und wird nur durch einen Teil erweitert, der eine Gewichtung der Kanäle der Featuremaps durchführt. Dafür wird ein GlobalPooling über jede Featuremap der Eingabe durchgeführt. Anschließend wird eine fully-connected Layer mit ReLU-Aktivierung und eine zweite fully-connected Layer mit Sigmoid-Aktivierung genutzt, um einen Vektor an Skalierungsfaktoren für die Featuremaps zu erhalten. Die Struktur ist in Abbildung 8 dargestellt. Die Erweiterung verschiedener bestehender Encoder-Strukturen mit dem SE-Modul führte zu deutlichen Verbesserungen bei den erzielten Ergebnissen der Bildklassifizierung auf dem ImageNet Datensatz und erhöht den Rechenaufwand und die Parameterzahl nur geringfügig [15].

Die SENet154-Struktur basiert auf der ResNext-Struktur, welche wiederum eine Abwandlung der ResNet-Struktur ist. Bei einem ResNext-Block gibt es nicht einen Pfad zur Ausgabe, sondern die Anzahl der Pfade wird durch einen Parameter, Kardinalität genannt, bestimmt. Alle Pfade, außer der skip-connection, führen identische Operationen durch, allerdings mit stark verringerter Kanalgröße im Vergleich zum eigentlichen ResidualBlock [34]. Die ResNext-Struktur ist in Abbildung 8 gezeigt.

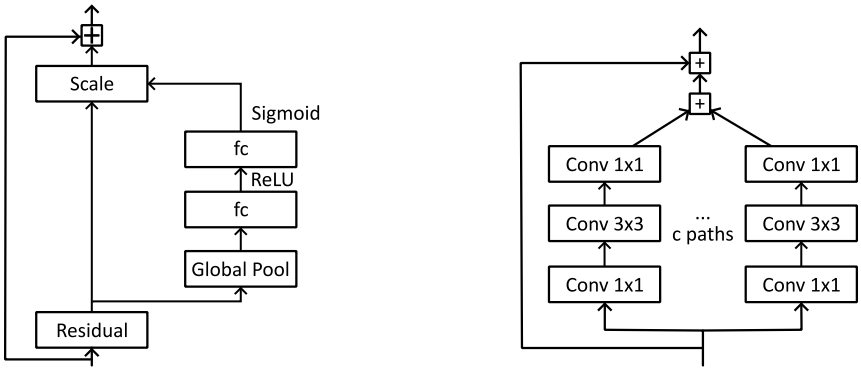


Abbildung 8: Generische Erweiterung des ResidualBlock mit dem SE-Modul (links) und Struktur eines ResNext-Blocks (rechts).

Nach der Kodierung muss der Decoder die Dimension der ursprünglichen Eingabe wiederherstellen. Dazu muss ein upsampling auf dem kodierten Featurevektor durchgeführt werden. Eine mögliche Struktur ist die von Laina et al. eingeführte UpProjection [20]. Sie kann als Umkehrung eines ResidualBlocks verstanden werden. Zuerst wird auf der Eingabe eine Unpooling Operation durchgeführt, die die Dimension der Featuremap verdoppelt. Dabei werden die fehlenden Werte auf 0 gesetzt. Darauffolgend wird eine 5x5 Convolution sowohl auf dem Hauptpfad als auch auf der „skip-connection“ durchgeführt. Der Grund für die Kernelgröße 5 ist, dass bei einer 3x3 Convolution nur die benachbarten Nullen in den Filter einbezogen werden. Nach einer weiteren 3x3 Convolution werden die beiden Pfade durch Addition zusammengeführt und ergeben sich durch ReLU-Aktivierung zur Ausgabe des Blocks. Die Struktur des UpProjectionBlock ist in Abbildung 9 dargestellt.

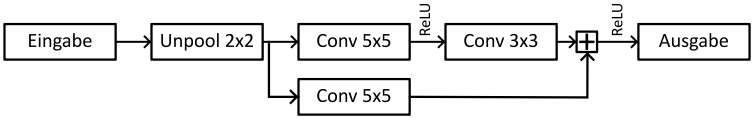


Abbildung 9: Schematische Darstellung eines UpProjectionBlocks für den Decoder.

Ein Problem der einfachen Encoder-Decoder Architektur ist, dass die Ausgabe des Decoders unscharf ist und die Feinstruktur der Eingabe nicht richtig dargestellt wird. Dieses Problem lösten Ronnenberger et al. im Jahr 2015 durch die Einführung des sogenannten UNets [25]. Dieses zeichnet sich durch besondere skip-connections zwischen Teilen des Encoders und Decoders aus,

die zu einer Verbesserung der Auflösung der Ausgabe führen. Das Prinzip, Zwischenausgaben des Encoders im Decoder oder an anderer Stelle weiterzunutzen, erwies sich als sehr effizient und wird auch in aktuellen neuronalen Netzen angewandt. Die Struktur des originalen UNets ist in Abbildung 10 abgebildet.

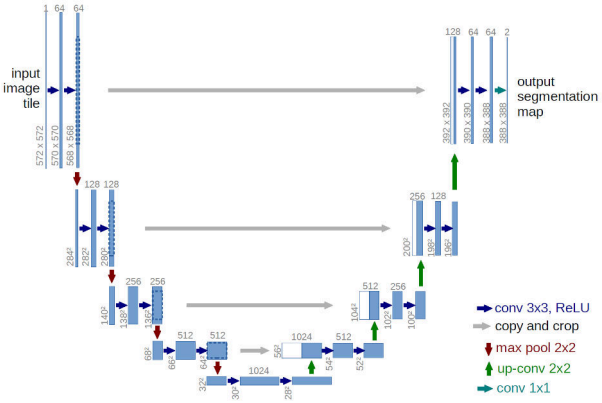


Abbildung 10: Darstellung der Struktur von UNet. Abbildung aus [25]. Die grauen Pfeile stellen die skip-connections zwischen dem Encoder- und Decoderteil des Netzes dar.

Da besonders der Encoderteil der Encoder-Decoder-Architektur eine große Anzahl von Parametern besitzt, werden häufig die Parameter mit transfer learning initialisiert. Dabei wird ein bereits trainiertes Modell, das in Teilen die gleiche Struktur aufweist, genutzt und die Parameter für die strukturell gleichen Teile werden übernommen. Die übernommenen Parameter sind meist nicht optimal für den eigenen Anwendungsfall, sind aber besser als eine zufällige Initialisierung und führen daher zu kürzeren benötigten Trainingszeiten und potenziell auch zu besseren Ergebnissen [10]. Ein gängiges Trainingsset ist der ImageNet-Datensatz für Objekterkennung und Klassifikation, da dieser über etwa 1,3 Millionen Bilder mit 1000 verschiedenen Klassen verfügt [26]. Die auf ImageNet trainierten Modellparameter sollten also gut für eine allgemeine Feature-Extraktion auf Bildern sein.

2.4 Metriken

Beim Training werden die Parameter des Modells optimiert, indem der Wert einer Verlustfunktion minimiert wird. Daher muss die Verlustfunktion den Unterschied zwischen der Vorhersage und dem Label quantifizieren, sodass ein kleinerer Wert mit einem „besseren“ Ergebnis einhergeht. Im Folgenden werden einige Verlustfunktionen und Metriken erläutert, die sich für die Bewertung von Tiefenbildern eignen. Für die mathematische Beschreibung seien im folgenden g das Ground Truth Tiefenbild, d das vorhergesagte Tiefenbild, n die Anzahl der Pixel im Tiefenbild und der Index i das i-te Pixel in der Ground Truth oder Vorhersage. Weiterhin seinen alle Werte $g_i, d_i > 0$.

Zwei einfache Berechnungen, um den Fehler zwischen Vorhersage und Ground Truth zu quantifizieren, sind der **Mean Absolute Error (MAE)** und der **Root Mean Squared Error (RMSE)**. In beiden Fällen wird die Differenz zwischen Vorhersage und Ground Truth pro Pixel berechnet. Beim MAE wird der Absolutwert dieser Differenz über alle Pixel gemittelt. Beim RMSE wird statt des Absolutwertes das Quadrat der Differenz gebildet.

$$MAE(g,d) = \frac{1}{n} \cdot \sum_{i=1}^{n} |g_i - d_i| \qquad RMSE(g,d) = \sqrt{\frac{1}{n} \cdot \sum_{i=1}^{n} (g_i - d_i)^2}$$

Der RMSE gewichtet starke Ausreißer deutlich schwerer als der MAE. Beide Funktionen haben den Nachteil, dass der Fehler nicht in Relation zur Entfernung gesehen wird. Eine Fehleinschätzung um 5 m ist bei einer realen Distanz von 1 m schlechter zu bewerten, als bei einer Realdistanz von 100 m. Daher bietet sich anstatt des absoluten Fehlers der **absolute relative Fehler (REL)** an. Hier werden die absoluten Differenzen durch die Entfernung in der Ground Truth geteilt, um die Fehleinschätzung relativ zu bewerten. Eine weitere Variante ist der **logarithmische Fehler (LOG)**, bei dem der Betrag der Differenz der Logarithmen zwischen Vorhersage und Ground Truth gebildet wird. Aufgrund der Logarithmusgesetze kann diese Form zum Logarithmus des Verhältnisses zwischen Ground Truth und Vorhersage umgeformt werden.

$$REL(g,d) = \frac{1}{n} \cdot \sum_{i=1}^{n} \frac{|g_i - d_i|}{g_i} \qquad LOG(g,d) = \frac{1}{n} \cdot \sum_{i=1}^{n} |log(d_i) - log(g_i)|$$

Um ein besseres Verständnis über die Streuung der Abweichung zwischen Vorhersage und Ground Truth zu erhalten, können die Abweichungen in Kategorien unterteilt werden. Dafür wird ein Schwellwert δ definiert und der Anteil der Pixel errechnet, bei denen das Verhältnis zwischen der Vorhersage und der Ground Truth kleiner als der Schwellwert ist. Üblicherweise werden mehrere Schwellwerte definiert, um eine Übersicht über die Fehlerbereiche zu erhalten.

$$ACC_\delta(g,d) = \frac{1}{n} \sum_{i=1}^{n} \begin{cases} 1 & \text{, wenn } \max(\frac{d_i}{g_i}, \frac{g_i}{d_i}) \leq \delta \\ 0 & \text{sonst} \end{cases}$$

Ein Problem der einfachen Mittelwertbildung über alle einzelnen Pixelfehler ist, dass die Struktur der Tiefenbilder nicht berücksichtigt wird. So ist es möglich, dass zwei Vorhersagen einen gleichen mittleren Fehler aufweisen, aber eine die Struktur der Ground Truth deutlich besser widerspiegelt. Dies ist Beispielhaft in Abbildung 11 gezeigt. Dabei ist die schwarze Linie die Ground Truth, rot und grün die Vorhersagen. Beide haben denselben mittleren Fehler zur Ground Truth, die grüne Linie stellt die Struktur der Ground Truth aber deutlich besser dar als die rote.

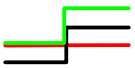


Abbildung 11: Darstellung zweier Vorhersagen (rot, grün) mit identischem absoluten relativen Fehler bezüglich einer Ground Truth (schwarz). Dabei spiegelt die grüne Funktion die Struktur der Ground Truth deutlich besser wider als rot.

Hu et al. definieren zur Einschätzung des Fehlers der Kanten daher die **Edge Accuracy** [14]. Dazu wird durch Anwenden des Sobel-Operators (gekennzeichnet mit $f_{x/y}$) in x- und y-Richtung die Stärke der Kanten für jedes Pixel berechnet und mit einem Schwellwert Δ verglichen. Liegt der berechnete Wert über dem Schwellwert, ist eine Kante vorhanden.

$$\text{is\_edge}_\Delta(i) = (\sqrt{f_x(i)^2 + f_y(i)^2} \geq \Delta)$$

Diese Kantenklassifikation wird mit Precision, Recall und F1-Score bewertet. Dabei ist die Precision das Verhältnis der Anzahl aller Kanten, die auch als solche klassifiziert wurden, zur Anzahl aller als Kanten identifizierten Elemente.

$$\text{Precision} = \frac{TP}{TP + FP}$$

Der Recall ist das Verhältnis der Anzahl aller Kanten, die auch als solche klassifiziert wurden, zur Anzahl aller existierenden Kanten.

$$\text{Recall} = \frac{TP}{TP + FN}$$

Der F1-Score ist das harmonische Mittel von Precision und Recall.

$$\text{F1-Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

3 Eigener Ansatz & Implementierung

Alle Implementierungen sind auf der Gitlab-Instanz des Fachbereichs Informatik der Freien Universität Berlin unter folgendem Link zugänglich: `https://git.imp.fu-berlin.de/autoauto/Bachelorarbeit_Andor_Kristen`

3.1 Vorbereitung

Bevor eine Tiefenberechnung auf Kamerabildern durchgeführt werden kann, muss eine Datengrundlage geschaffen werden. Dabei muss zuerst eine Kamera für die Berechnung ausgewählt werden. Da eine Tiefenschätzung vor dem Fahrzeug am sinnvollsten erscheint, ist eine nach vorne gerichtete Kamera nötig. Daher gab es die Auswahl zwischen der „broadrreach"-Kamera und der „Hella"-Kamera. Es wurde aus zwei Gründen die „Hella"-Kamera genutzt. Zum einen besitzt sie aufgrund des kleineren Sichtfelds eine höhere Winkelauflösung. Zum Anderen ist sie im Vergleich zur „broadrreach"-Kamera deutlich höher positioniert, was einen besseren Überblick über das Straßengeschehen erlaubt. So können auch Objekte hinter flachen Hindernissen gesehen werden.

Um mit den Bildern gut rechnen zu können, muss das Kamerabild zuerst begradigt werden, sodass die Abbildung der einer idealen Lochbildkamera entspricht. Dies ermöglicht das einfache Umrechnen zwischen Pixelposition und Richtungsvektor im Kamerakoordinatensystem. Weiterhin vereinfacht es den Umgang mit den Punktwolken des LiDAR-Scanners, da diese mit einer einfachen Matrixmultiplikation in das Kamerabild projiziert werden können. Bei der Hella-Kamera handelt es sich um eine Weitwinkelkamera mit tonnenförmiger Verzeichnung. Die Funktion zur Beschreibung der Verzeichnung ist vom Hersteller gegeben. Sie ist gebrochen rational und besitzt 4 Parameter. Die Pixelposition im verzeichneten Bild errechnet sich nach:

$$df = 1 + \frac{\kappa_1 r^2 + \kappa_2 r^4}{1 + \lambda_1 r^2 + \lambda_2 r^4} \qquad \begin{pmatrix} x_{dist} \\ y_{dist} \end{pmatrix} = \begin{pmatrix} df \cdot x_{undist} \\ df \cdot y_{undist} \end{pmatrix}$$

Dabei ist r der Abstand vom Bildmittelpunkt im verzeichnungsfreien Bild in Bildkoordinaten. Die x und y Koordinaten sind ebenfalls in Bildkoordinaten angegeben, jeweils für das verzeichnete und verzeichnungsfreie Bild. Die gegebenen Parameter für λ_1,λ_2, κ_1, κ_2 führten zu keiner optimalen Verzeichnungskorrektur. Dies lässt sich daran erkennen, dass Geraden in der dreidimensionalen Welt nicht als Geraden ins Bild projiziert werden. In Abbildung 12 sind einige dieser Geraden in Grün markiert. Um die Parameter anzupassen, wurden auf 20 Kamerabildern Linien identifiziert, die als Geraden projiziert werden sollten, wie z.B. Häuserecken, Bordsteinkanten und Straßenlaternen. Die Parameter der Verzeichnungsfunktion wurden dann manuell so angepasst, dass die verzeichnungskorrigierten Bilder der Abbildung einer Lochbildkamera entsprechen, also die Linien als Geraden im Bild dargestellt sind. Um die gefundenen Parameter zu prüfen, wurden die Punkte der 3D-Punktwolke des LiDAR-Scanners mit der gegebenen Kameramatrix in das Bild projiziert und auf Deckung mit dem Bild geprüft. Da die projizierten Punktwolken mit den verzeichnungsfreien Bildern eine gute Übereinstimmung besaßen, wurden die so gefundenen Parameter als ausreichend genau bewertet und im weiteren Verlauf genutzt. Der Vergleich zwischen originalen und angepassten Parametern ist in Abbildung 12b und 12c abgebildet.

Um die Korrektur des Kamerabildes schnell und flexibel durchzuführen, wurde eine Zuweisungstabelle berechnet, die jedem Pixel im korrigierten Bild eine Position im originalen, verzeichneten Kamerabild zuordnet. Die Koordinaten im Originalbild sind dabei als Gleitkommazahlen gegeben, sodass unterschiedliche Interpolationsverfahren genutzt werden können. Nach dem Laden der Zuordnungstabelle kann so das verzerrte Kamerabild mit der „remap"-Funktion der Bibliothek open-cv in nur einer Zeile effizient entzerrt werden.

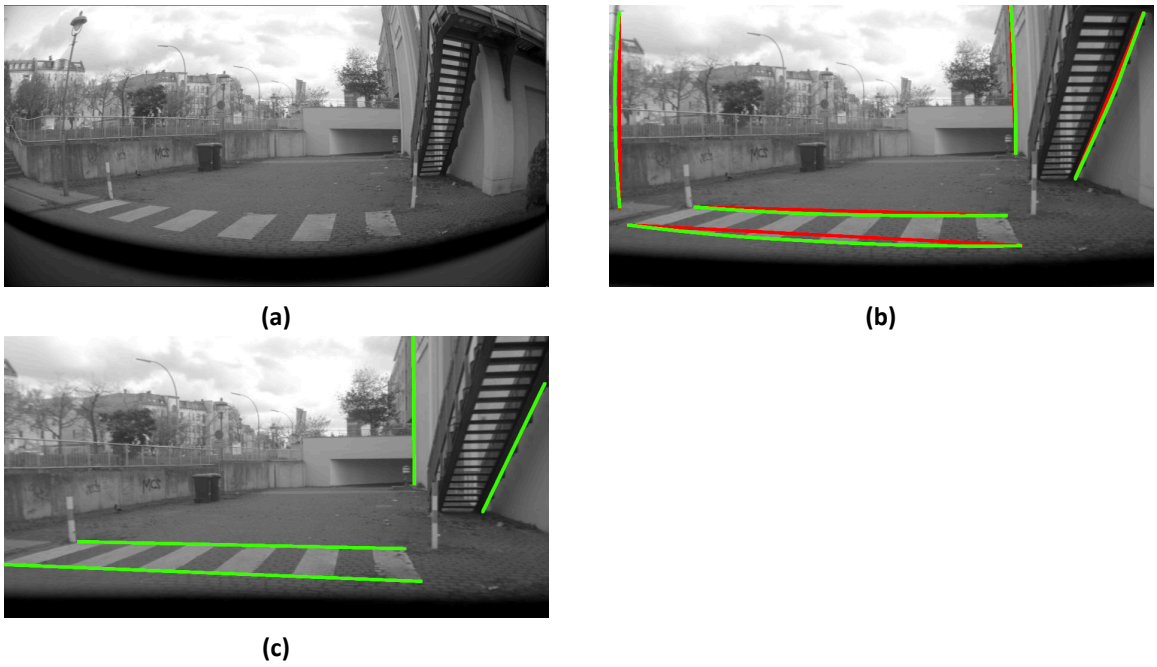


Abbildung 12: Darstellung eines Bildes der Hella-Kamera in verschiedenen Varianten. In Grün sind Linien von Objektkanten gekennzeichnet, die Geraden in der Realität darstellen. In Rot sind Geraden im Bild markiert, die von den Objektkanten abweichen. **a**: Unverändertes Kamerabild **b**: Verzeichnungskorrektur mit gegebenen Parametern. **c**: Verzeichnungskorrektur mit angepassten Parametern.

Mit den optimierten Kameraparametern können nun die Datensätze für die beiden Ansätze zur Tiefenbestimmung erstellt werden. Da sich die Ansätze grundsätzlich unterscheiden, wurden zwei verschiedene Datensätze erstellt.

Beide Ansätze zur Tiefenbestimmung liefern Tiefenbilder, also Bilder, bei denen die Pixelwerte Entfernungen repräsentieren. Um diese zu prüfen, werden mit den LiDAR-Punktwolken auch Tiefenbilder berechnet. Dafür wird für jeden Punkt einer Punktwolke der Abstand zur Kamera (Koordinatenursprung) berechnet. Anschließend werden die Punkte mit der Kameramatrix in Pixelkoordinaten projiziert. Für jedes Pixel des Bildes wird dann der Abstand des Punktes eingetragen, der den geringsten Abstand zur Kamera hat.

3.2 Tiefenbestimmung mit optischem Fluss

3.2.1 Ansatz

Wie in Abschnitt 2.2 beschrieben, lassen sich aus dem optischen Fluss unter gewissen Umständen bereits grobe Informationen über die Positionierung von Objekten ableiten. Um aus dem optischen Fluss eine genaue Abstandsinformation zu errechnen, sind weitere Annahmen und Informationen notwendig. Dafür seien zwei aufeinanderfolgende Bilder einer Kamera gegeben sowie die Kameramatrix und die Bewegung der Kamera zwischen den Zeitpunkten der Bildaufnahmen bekannt. Für die Berechnung der Entfernungen wird weiterhin angenommen, dass der optische Fluss nur durch die Eigenbewegung der Kamera ausgelöst wird.

Die Idee ist, aus dem optischen Fluss von zwei aufeinanderfolgenden Bildern zwei Geradengleichungen herzuleiten, die auf denselben Punkt im dreidimensionalen Raum zeigen und diesen Punkt durch den Schnitt der beiden Geraden zu bestimmen. Das Prinzip ist in Abbildung 13 dargestellt. Um zwei Geradengleichungen aufstellen zu können, muss die Pixelposition eines Objektes, welches im ersten Bild in Pixel b_1 ist, in Bild 2 bestimmt werden. Die Pixelposition im

zweiten Bild sei b_2. Dafür wird der optische Fluss der Bildsequenz berechnet. Dieser ordnet jedem Pixel im ersten Bild einen Vektor zu, der die Verschiebung dieses Pixels im zweiten Bild angibt. Sei K die Kameramatrix und P_1, P_2 die Koordinaten des Punktes P im Kamerakoordinatensystem des ersten und zweiten Bildes, dann lässt sich wie folgt für beide Punkte ein Richtungsvektor im jeweiligen Kamerakoordinatensystem bestimmen und eine Geradengleichung im jeweiligen Kamerakoordinatensystem aufstellen:

$$\text{Koordinatensystem Bild 1: } r_1 = K^{-1}b_1 \qquad P_1 = \vec{0} + d_1 \cdot \vec{r_1}$$
$$\text{Koordinatensystem Bild 2: } r_2 = K^{-1}b_2 \qquad P_2 = \vec{0} + d_2 \cdot \vec{r_2}$$

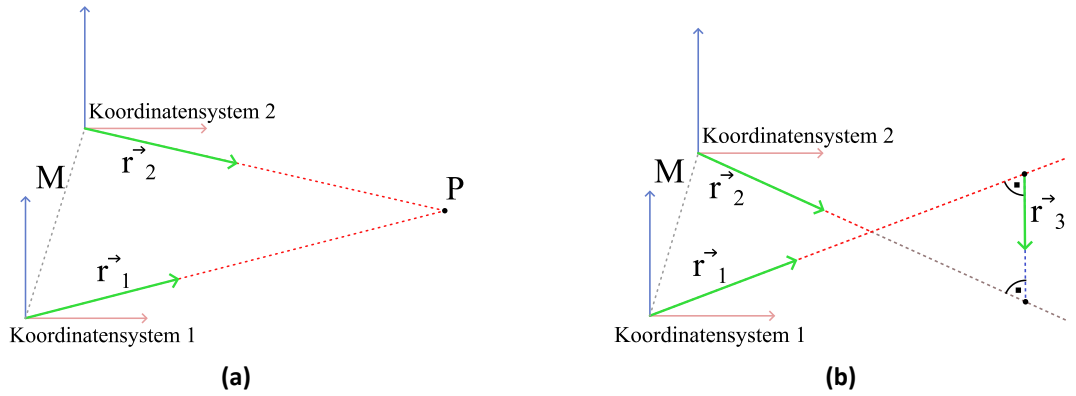


Abbildung 13: Schematische Darstellung der durch den optischen Fluss berechneten Geraden aus den jeweiligen Kamerakoordinatensystemen. **a**: Einfacher Fall, P ergibt sich als Schnittpunkt der beiden Geraden. **b**: Allgemeiner Fall, die Geraden schneiden sich nicht, es lässt sich nur der nächste Punkt der Geraden bestimmen.

Weiterhin ist die Bewegung der Kamera zwischen den beiden Aufnahmezeitpunkten bekannt. Sie lässt sich in Form einer Transformationsmatrix M darstellen, welche die Transformation eines Punktes vom Kamerakoordinatensystem zum Zeitpunkt des ersten Bildes in das Kamerakoordinatensystem zum Zeitpunkt des zweiten Bildes beschreibt. Da P_1 und P_2 die Repräsentationen des Punktes P im jeweiligen Kamerakoordinatensystem sind, gilt daher:

$$M^{-1}P_2 = P_1$$
$$\Rightarrow M^{-1}\vec{0} + d_2 \cdot (M^{-1}\vec{r_2}) = d_1 \cdot \vec{r_1}$$

Die einzigen Unbekannten in dieser Gleichung sind die Skalare d_1 und d_2, welche die Distanz des Punktes P zur Kameraposition (Koordinatenursprung) für das erste und zweite Bild darstellen. Sind die Richtungsvektoren $\vec{r_1}$ oder $\vec{r_2}$ nicht normiert, ergibt sich die Distanz durch Multiplikation der Skalare mit dem Betrag der Richtungsvektoren. Da es sich um eine Gleichung im dreidimensionalen Raum handelt, ist das Gleichungssystem überbestimmt.

Im Allgemeinen müssen sich die Geraden nicht schneiden. Dieser Fall ist in Abbildung 13 rechts dargestellt. Er kann beispielsweise eintreten, wenn der errechnete optische Fluss nicht exakt die Bewegung eines Objektes im Bild widerspiegelt. Daher wird anstelle des Schnittpunktes der Geraden der Punkt auf den Geraden berechnet, an denen sich beide Geraden am nächsten sind. Dafür wird mit dem Kreuzprodukt ein Hilfsvektor $\vec{r_3}$ berechnet, der senkrecht zu den beiden

Richtungsvektoren $\vec{r_1}, \vec{r_2}$ steht. Wenn man von einem Punkt auf der Gerade 1 in Richtung des Vektors $\vec{r_3}$ geht und so die Gerade 2 schneidet, dann sind dies die Punkte der beiden Geraden, die sich am nächsten sind. Im Kamerakoordinatensystem des ersten Bildes bedeutet dies:

$$\vec{r_3} = \vec{r_1} \times (M^{-1}\vec{r_2})$$
$$d_1\vec{r_1} + t\vec{r_3} = M^{-1}\vec{0} + d_2 \cdot M^{-1}\vec{r_2}$$

Dabei ist d_1 die Entfernung des Punktes auf der Geraden in Richtung $\vec{r_1}$, der der Geraden $\vec{r_2}$ am nächsten ist, analog für d_2. Der Faktor t ist die minimale Distanz zwischen den beiden Geraden. Die Gleichung besitzt drei Unbekannte, ist im dreidimensionalen Raum gestellt und lässt sich daher exakt lösen.

Das zum ersten Bild korrespondierende Tiefenbild erhält dann am Pixel b_1 den Wert d_1. Das Tiefenbild der zweiten Aufnahme an Position b_2 den Wert d_2. Es wird also der Punkt auf der jeweiligen Geraden genommen, welcher der anderen Geraden am nächsten ist. Eine alternative wäre es, den Mittelpunkt zu nutzen und dessen Entfernung in beiden Bildern einzutragen. Der Faktor t gibt die Entfernung zwischen den beiden nächsten Punkten der Geraden an und kann somit als eine Art Qualitätsmaß interpretiert werden. Er wird aber nicht weiter berücksichtigt.

3.2.2 Datensatz & Implementierung

Für den beschriebenen Ansatz werden zeitlich eng aufeinanderfolgende Bildsequenzen und gleichzeitig die Position der Kamera zu jedem Zeitpunkt der Kameraaufnahme benötigt. Für diesen Zweck wurde in Python eine ROS-Node geschrieben, die sich auf die Bild-Nachrichten der Hella-Kamera und die Punktwolken des LiDAR-Scanners subscribed und diese abspeichert. Beim Abspielen einer bag-Datei können so die Daten einfach extrahiert werden. Vor dem Abspeichern wird die Punktwolke bereits in das Hella-Koordinatensystem transformiert, um mit den Daten einfacher rechnen zu können. Um den benötigten Speicherplatz zu verringern, wurden alle Punkte, die hinter der Kamera liegen, nicht gespeichert, da diese grundsätzlich nicht im Kamerabild sichtbar sein können. Dies halbiert die Größe der zu speichernden Punktwolke. Zusätzlich wird zum Zeitpunkt jeder Bild- und Punktwolkennachricht die globale Transformation der Hella-Kamera in Bezug auf das „map“-Koordinatensystem gespeichert, um daraus später die Transformationsmatrizen zwischen zwei beliebigen Kameraaufnahmezeitpunkten berechnen zu können. Es wurde eine bag-Datei vollständig abgespielt, wodurch ein Datensatz aus 7510 aufeinanderfolgenden Bildern entstanden ist.

Der beschriebene Ansatz zur Tiefenbestimmung wurde in Python mit den Bibliotheken numpy und open-cv implementiert. Zur Berechnung des optischen Flusses wird der Dense Robust Local Optical Flow Algorithmus aus der open-cv Bibliothek mit zwei Bildern mit einem zeitlichen Abstand von 0,2 s genutzt. Der zeitliche Abstand ist dabei leicht variabel, da die Wiederholungsrate der Kamera nicht konstant ist. Die Transformationsmatrix zwischen den Kamerakoordinatensystemen wird über die gespeicherte globale Transformationsmatrix errechnet. Das Gleichungssystem für jedes Pixel wird in Form einer 3x4-Koeffizientenmatrix aufgestellt und mit dem Befehl „numpy.linalg.solve“ der numpy-Bibliothek gelöst. Für eine schnellere Implementierung wird nicht über jede Koeffizientenmatrix iteriert und einzeln gelöst, sondern es wird ein Tensor erstellt, der die Koeffizientenmatrizen für alle Bildpunkte enthält. Dieser Tensor kann als solcher an den Löser für lineare Gleichungssysteme gegeben werden, wodurch langsame Iteration in Python vermieden wird. Vorher wird durch Berechnung des Skalarprodukts noch eine Prüfung auf Parallelität der Geraden durchgeführt, da das Gleichungssystem in diesem Fall nicht eindeutig lösbar ist. Das Ergebnis ist ein Tiefenbild mit der Dimension der Eingabebilder, das an jeder Stelle die berechnete Entfernung für das Pixel des ersten Bildes der Bildsequenz enthält.

Um die Ergebnisse zu prüfen, wird aus den abgespeicherten Punktwolken ein Tiefenbild errechnet. Dafür wird für jeden Punkt der Punktwolke die Distanz zur Kamera (Koordinatenursprung) berechnet. Anschließend werden die Punkte durch Multiplikation mit der Kameramatrix projiziert und die Pixelposition im Bild errechnet. Für jedes Pixel des Tiefenbildes wird dann der kleinste Distanzwert aller Punkte genommen, die auf dieses Pixel projiziert werden.

3.3 Tiefenbestimmung mit neuronalem Netz

Ein alternativer Ansatz ist es, die Entfernungen aus einem Einzelbild mit einem neuronalen Netz schätzen zu lassen. Die Grundidee ist, die Tiefenschätzung als eine Regression auf jedem Pixel zu formulieren und das Modell dementsprechend zu optimieren. Dafür existieren unterschiedliche Ansätze. Im Rahmen dieser Arbeit wird ein Ansatz getestet, der ein neuronales Netz mit supervised learning auf Paaren von Bildern und ihren entsprechenden Tiefenbildern trainiert. Zu diesem Zweck muss ein Modell ausgewählt und implementiert werden und ein passender Datensatz zum Trainieren erstellt werden.

3.3.1 Datensatz

Der Datensatz für das Trainieren eines neuronalen Netzes wurde auf ähnliche Weise erstellt wie der für den optischen Fluss. Dabei werden nicht alle Kamerabilder und Punktwolken des LiDAR-Scanners genutzt, sondern nur im Abstand von einer Sekunde die Punktwolke mit dem zeitlich nächsten Kamerabild gespeichert. Der Grund dafür ist, dass die Bilder später manuell geprüft werden und daher die Anzahl an Bildern im Datensatz begrenzt ist. Aufeinanderfolgende Bilder weisen eine starke Ähnlichkeit miteinander auf, wodurch bei gleicher Menge an Gesamtbildern im Datensatz eine kleinere Menge unterschiedlicher Szenarien abgebildet wird. Die Daten werden analog zum Datensatz für den optischen Fluss abgespeichert, wobei die Position der Kamera in einem Referenzkoordinatensystem nicht benötigt wird. Anstelle dessen wurden die GPS-Koordinaten zu den Aufnahmezeitpunkten gespeichert. Die erhaltenen Daten sind noch nicht zum Trainieren eines neuronalen Netzes geeignet und werden im Folgenden als Rohdaten bezeichnet.

Die aus den Rohdaten projizierbaren Tiefenbilder stimmen nicht immer gut mit den Kamerabildern überein. Drei wichtige Gründe dafür sind:

Schnell bewegte Objekte, wie etwa andere Fahrzeuge. Sie bewegen sich in der Zeit zwischen Aufnahme des LiDAR-Scans und Aufnahme des Kamerabildes selbst. Dadurch kommt es teilweise zu einer verschobenen Darstellung dieser Objekte im Tiefenbild. Dies ist in Abbildung 14a dargestellt. Diese Bewegung kann nicht ohne weiteres herausgerechnet werden, da dafür nicht nur die genaue Bewegung dieser Objekte bekannt sein müsste, sondern auch die Punkte der Punktwolke, welche diese Objekte repräsentieren.

Starke Lenkbewegungen, wie etwa beim Abbiegen. Der LiDAR-Scanner rotiert kontinuierlich um eine Achse, damit die gesamte Umgebung des Autos erfasst wird. Wenn während dieser Rotation das Auto selbst auch rotiert, wird die reale Winkelgeschwindigkeit des LiDAR-Scanners verändert. Das Prinzip ist in Abbildung 14b gezeigt. Dies führt zu einer Stauchung bzw. Zerrung der Punktwolke.

Eigenbewegung eines Objektes, welche die Rotation des LiDAR-Scanners ausgleicht. Befindet sich ein Objekt in relativer Nähe zum Scanner, so kann es vorkommen, dass die Rotation des LiDAR-Scanners und die Bewegung des Objektes sich genau „ausgleichen“. Das Objekt bleibt

trotz der Rotation des Scanners genau im Blickfeld. Das Objekt erscheint dann verzogen. Dies ist in Abbildung 14c veranschaulicht.

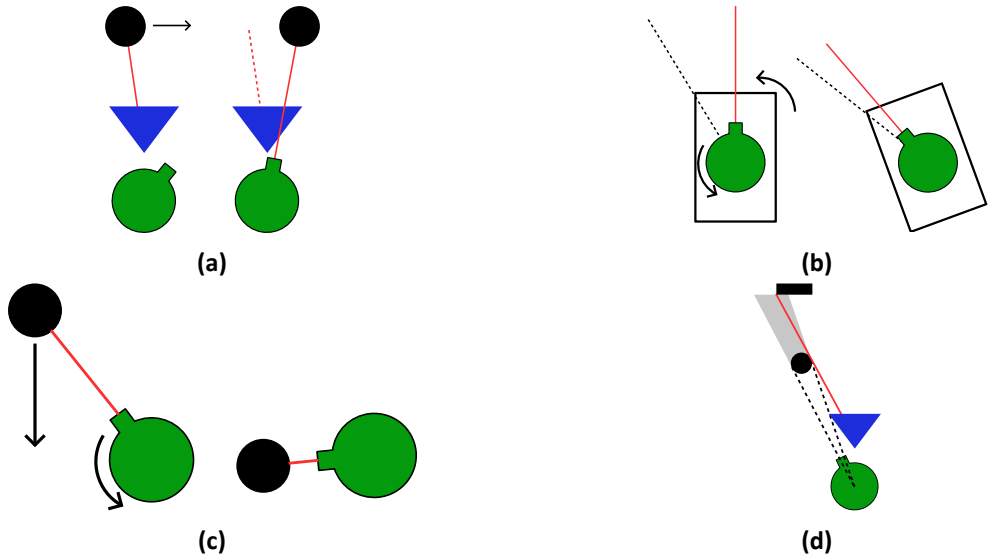
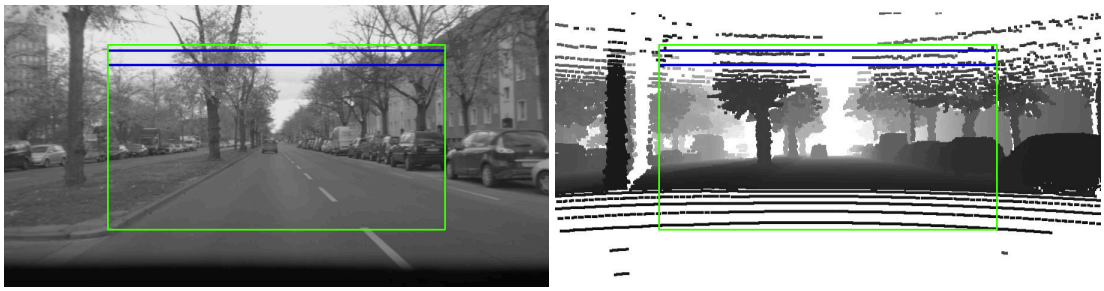


Abbildung 14: Veranschaulichung der potenziell auftretenden Probleme für die Erstellung von Tiefenbildern. Blau: Kamera, grün: LiDAR-Scanner, schwarz: Objekte. **a**: Veranschaulichung der Entstehung von Positionsdifferenzen im LiDAR-Bild durch Bewegung von Objekten zwischen den Aufnahmezeitpunkten. **b**: Veranschaulichung der Entstehung von Objektverzerrungen im LiDAR-Bild durch Rotation des Fahrzeugs bei der Aufnahme. **c**: Veranschaulichung der Entstehung von Objektverzerrungen im LiDAR-Bild durch Bewegung des Objektes mit der Rotation des LiDAR-Scanners. **d**: Veranschaulichung der Schattenentstehung durch Versatz zwischen LiDAR-Scanner und Kamera.

Aus diesen Gründen wurden alle erhaltenen Punktwolken manuell auf Übereinstimmung mit dem Kamerabild geprüft. Dafür wurden die Punktwolken in das Kamerabild projiziert, das Bild darübergelegt und die Übereinstimmung visuell geprüft. Bei einer Verschiebung der Kanten des Kamerabildes zum Tiefenbild wurde das Bild-Tiefenbild-Paar aussortiert. Für eine effiziente manuelle Aussortierung wurde in Python ein Hilfswerkzeug geschrieben, das die Bild-Tiefenbild-Paare mit variabler Transparenz übereinandergelegt darstellt und per Knopfdruck ein Aussortieren der Daten ermöglicht.

Aus den gefilterten Rohdaten wurden anschließend die Daten zum Trainieren des Modells erstellt. Dafür wurde der in Abbildung 15a gezeigte Kamera-Bildausschnitt gewählt und auf eine Auflösung von 352x192 Pixeln reduziert. Die Tiefenbilder werden auf die halbe Auflösung reduziert, da die Ausgabe des Modells die halbe Auflösung der Eingabe hat. Im Vergleich zum Originalbild wurde sowohl die Höhe als auch die Breite deutlich reduziert. Dies ist nötig, um die Vorhersage des Modells ausreichend schnell zu erhalten. Je größer das Bild, desto größer auch die Anzahl der für die Vorhersage benötigten Rechenoperationen und somit größere Latenz. Die Beschneidung der Höhe ist weiterhin damit zu begründen, dass der obere und untere Bildrand deutlich weniger relevant ist, als der zentrale Bereich. Sie sind auch nur teilweise durch den LiDAR-Scanner abgedeckt, was in Abbildung 15b erkennbar ist. Daher kann das Modell in diesen Bereichen auch nicht gut trainiert werden. Ein weiterer Grund für die Beschneidung der Seiten ist, dass es am Bildrand häufiger zu den oben erwähnten „Verzerrungen“ der Objekte kommt. Bei einem zur Seite größeren Bildausschnitt müssten also mehr Bilder aussortiert werden.

Durch den Versatz zwischen LiDAR-Scanner und Kamera kommt es bei dem in die Kamera projizierten Tiefenbild zu Schatteneffekten. Die Entstehung der Schatten ist schematisch in Abbildung 14d dargestellt. Die Schatten entstehen vor allem hinter Objekten wie Schildern und Ampeln und führen dazu, dass Trainingsdaten auf einer Seite dieser Objekte fehlen. Dies kann



(a) (b)

Abbildung 15: Darstellung des für den Datensatz gewählten Bildausschnittes (grün) und der für die Himmelerkennung genutzten Start-Bildzeilen. **a**: Im Kamerabild. **a**: Im Tiefenbild.

beim Trainieren des Modells zu unerwünschten Effekten führen, insbesondere einer zu breiten Vorhersage für diese Objekte. Um die Schatten zu minimieren, wird auf allen fehlenden Werten im Bild eine 3x3 Max-Filterung durchgeführt. Es wird also in einem 3x3 Umfeld um ein fehlendes Pixel nach dem maximalen Entfernungswert gesucht und sofern einer existiert, wird dieser für das Pixel eingetragen.

Ein Problem der projizierten Tiefenbilder ist, dass im Bereich, in dem sich der Himmel befindet, die Entfernungswerte fehlen. Auf den fehlenden Werten kann das Modell nicht trainiert werden. Die Bereiche des Bildes, die häufig den Himmel darstellen, können also nur selten trainiert werden, nämlich nur in den Fällen, in denen der Himmel verdeckt ist. Dies passiert typischerweise durch Objekte, die der Kamera nahe sind, wie über der Straße hängende Verkehrsschilder, Ampeln oder auch Tunneldecken. Das Modell lernt daher, diese Bereiche immer als ‚nahe' vorherzusagen, auch wenn keine Objekte in diesem Bereich sind. Um diesem Effekt entgegenzuwirken wurde eine Himmelerkennung implementiert. Dafür wird zuerst nach Kandidatpositionen für den Himmel gesucht, indem im oberen Bildbereich in zwei Zeilen (in Abbildung 15a in blau dargestellt) für jedes Pixel in einem 10x10 Feld um das Pixel geprüft wird, ob alle Werte im Tiefenbild nicht gesetzt sind. Ist dies der Fall, wird der Pixel als Startposition für den Himmel markiert. Die Bereiche mit starken Kanten im Kamerabild werden durch Anwendung des Laplacian of Gaussian gefunden. Die Kanten werden binarisiert, indem die stärksten 50 % (Werte nach Anwendung des Kantendetektionsfilters) behalten werden und die schwächeren Kanten verworfen werden. Diese Maske wird mit der Information, ob an dieser Stelle im Tiefenbild ein Wert existiert durch die ODER-Operation verknüpft. Ausgehend von den gefundenen Himmel-Startpositionen wird auf dieser Maske nach zusammenhängenden Bereichen gesucht. Alle Pixel, die von einer Himmel-Startposition erreicht werden können, werden als Himmel eingestuft. Im Anhang in Bild 26 ist die Himmelerkennung in den Einzelschritten gezeigt. Dieser Algorithmus erkennt nicht immer alle Bereiche des Himmels, aber markiert in keinem Bild des Datensatzes einen Bereich als Himmel, der eigentlich keiner ist. Ein vollständiges Ablaufschema über die Erstellung des Datensatzes ist in Abbildung 16 dargestellt.

Für die Erstellung des Datensatzes aus den gefilterten Rohdaten wurde ein Python-Skript erstellt, in dem die Parameter wie Bildausschnitt, Zielauflösung und Parameter für die Himmelerkennung direkt eingestellt werden können. Da bereits die Rohdaten mit schlechter Übereinstimmung zwischen Bild und Tiefenbild aussortiert wurden, kann so durch Ausführung des Skriptes mit anderen Parametern einfach ein Datensatz mit anderen Eigenschaften, wie etwa einer anderen Auflösung, erstellt werden.

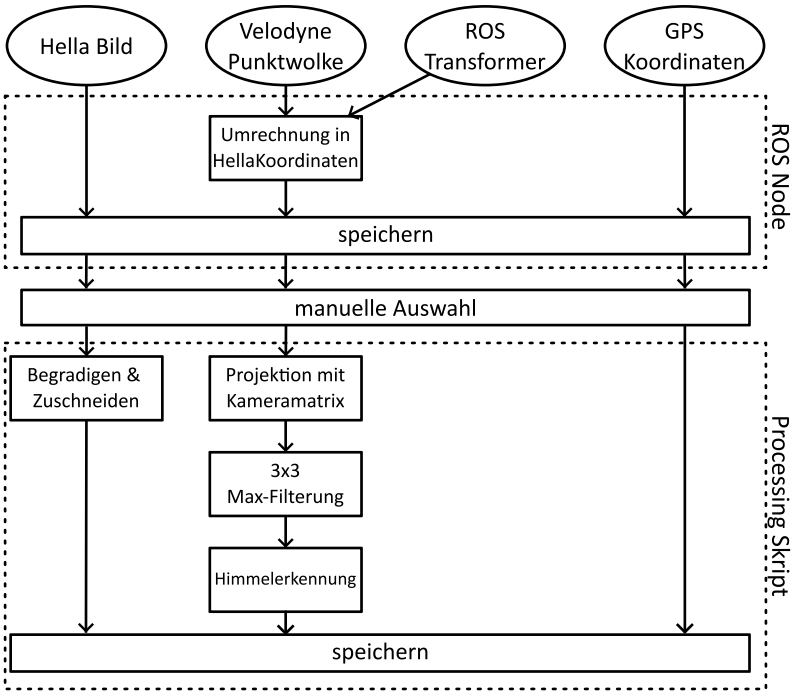


Abbildung 16: Ablaufschema zur Erstellung des Datensatzes zum Trainieren des neuronalen Netzes.

3.3.2 Modellarchitektur & Training

Als Referenz für die Architektur des Netzwerks wurde die Veröffentlichung von Hu et al. aus dem Jahr 2019 genutzt [14]. Das von Ihnen entworfene Modell erzielt sehr gute Ergebnisse auf dem NYU Depth V2-Datensatz und ist auch mit neueren Veröffentlichungen vergleichbar [8, 24]. Da in der Veröffentlichung verschiedene Backbones für den Encoder getestet und verglichen wurden, die sich in ihrer Komplexität stark unterscheiden, eignet sich das Modell für eine Abwägung zwischen Geschwindigkeit der Vorhersage und der erzielten Genauigkeit. Ein weiterer interessanter Aspekt ist das Abweichen von der häufig verwendeten UNet-Struktur für den Enocder-Decoder. Diese wird unter anderem auch in anderen Veröffentlichungen zum Thema monokulare Tiefenschätzung erfolgreich eingesetzt [1, 8]. Stattdessen nutzen Hu et al. ein eigenes Modul, das sogenannte Multiscale Feature Fusion Modul (MFF), um die Zwischenergebnisse des Encoders weiterzuverarbeiten. Hier bietet sich ein Vergleich an, bei dem das MFF durch eine UNet-Struktur ersetzt wird.

Die Struktur des Netzwerks ist in Abbildung 17 dargestellt. Es setzt sich aus 4 Komponenten zusammen, dem Encoder (blau), dem Decoder (braun), dem Multiscale Feature Fusion Modul (MFF, gelb) und dem Refinement Modul (grau). Zuerst wird die Eingabe durch eine einfache Convolution mit Stride zwei auf die halbe Auflösung reduziert. Anschließend wird sie im vierstufigen Encoder jeweils um die Hälfte der Auflösung reduziert, bei gleichzeitiger Erhöhung der Anzahl der Featuremaps. Die Ausgabe des Encoders wird nach einer Convolution im Decoder wieder schrittweise auf die ursprüngliche Dimension gebracht. Dabei wird der UpProjectionBlock von Laina et al. genutzt [20]. Die Ausgaben der einzelnen Schritte des Encoders werden ebenfalls mit dem UpProjectionBlock auf die ursprüngliche Auflösung skaliert. Die vier Teile werden konkateniert und mit einer Convolution verarbeitet. Dieser Teil ist das MFF. Die Ausgabe des Decoders und die Ausgabe des MFF werden im Refinement Modul konkateniert und mit drei Convolutions zur endgültigen Ausgabe des Modells verarbeitet. Die Ausgabe des Modells besitzt die halbe Auflösung der RGB-Eingabebilder. Eine Übersicht über die Dimensionen der Ebenen des Modells ist in Tabelle 1 gegeben.

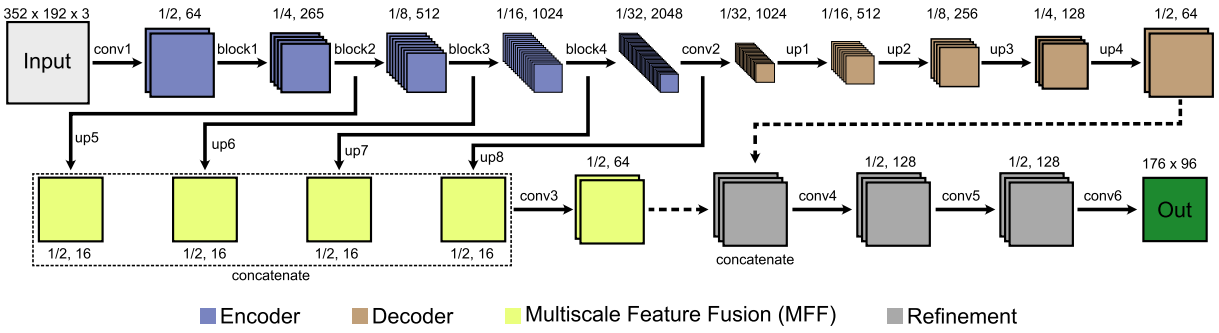


Abbildung 17: Struktur des neuronalen Netzes von Hu et al. [14].

Tabelle 1: Übersicht der Dimension der Ein- und Ausgabe der Ebenen des Modells. „block“ und „up“ Ebenen bestehen aus mehreren convolution Ebenen.

Ebene	Ausgabe Auflösung	Eingabe Channels	Ausgabe Channel
conv1	176 x 96	3	64
block1	88 x 48	64	256
block2	44 x 24	256	512
block3	22 x 12	512	1024
block4	11 x 6	1024	2048
conv2	11 x 6	2048	1024
up1	22 x 12	1024	512
up2	44 x 24	512	256
up3	88 x 48	256	128
up4	176 x 96	128	64
up5	176 x 96	256	16
up6	176 x 96	512	16
up7	176 x 96	1024	16
up8	176 x 96	2048	16
conv3	176 x 96	64	64
conv4	176 x 96	128	128
conv5	176 x 96	128	128
conv6	176 x 96	128	1

Das Modell wurde in Python mit TensorFlow und der Keras API implementiert. Dabei wurde sich an der Referenzimplementierung der Veröffentlichung von Hu et al. [14] orientiert, die in PyTorch geschrieben wurde. Weiterhin wurde für einen Vergleich das MFF-Modul entfernt und durch UNet-artige, direkte Verbindungen zwischen dem Encoder und Decoder ersetzt.

Für das Encoder-Backbone wurden in der Veröffentlichung drei verschiedene Architekturen verglichen, ResNet50, DenseNet161 und SENet154. Alle Backbones wurden mit auf ImageNet vortrainierten Gewichten initialisiert. Dabei erzielten die Modelle mit ResNet- und DenseNet-Backbone nahezu identische Ergebnisse und der SENet-Encoder war in allen Metriken etwa 5% - 10% besser. Daher wurde im Rahmen dieser Bachelorarbeit das ResNet50-Backbone und SENet154-Backbone implementiert. Der Grund für die Nutzung von ResNet50 im Gegensatz zu DenseNet169 ist, dass die auf ImageNet vortrainierten Gewichte für ResNet50 in TensorFlow verfügbar sind, für DenseNet169 allerdings nicht. Ein Vergleich zwischen der Nutzung von vortrainierten Gewichten und dem Training auf zufällig initialisierten Gewichten wird in Abschnitt 4.2.2 durchgeführt.

Für das Training wurden verschiedene Datenaugmentierungen implementiert. In der Literatur werden typischerweise die Rotation um den Bildmittelpunkt, Spiegelung an der vertikalen Achse und Verschiebungen im Farbraum genutzt. Da die Rotation invalide Daten in den Bildecken erzeugt, werden meist kleine Rotationsgrade genutzt [1, 14]. Für die Verschiebungen im Farbraum wird dabei in den HSV-Farbraum konvertiert, wodurch Farbton, Sättigung und Helligkeit einfach und separat voneinander modifiziert werden können. Die Kanäle im HSV-Farbraum entsprechen nicht direkt dem menschlichen Empfinden, so können zwei unterschiedliche Farben mit demselben „Value"-Wert ein unterschiedliches Helligkeitsempfinden auslösen. Für die Augmentierung ist dies aber ausreichend. Im Training wurden horizontale Spiegelungen mit 50 % Wahrscheinlichkeit durchgeführt, das Bild zufällig um einen Wert im Bereich [-5°, 5°] rotiert, der Farbton (Hue) mit einem Zufallswert im Bereich [-20°, 20°] addiert und die Sättigung und Helligkeit (Value) mit einem Faktor in [0.8, 1.2] multipliziert.

Das Training wurde auf Google Colab mit einer Nvidia Tesla V100 Grafikkarte und auf dem HPC Cluster „Curta" der Freien Universität Berlin durchgeführt [2]. Als Optimizer wurde Adam [17] mit einer Lernrate von 0.0001, $\beta_1 = 0.9$ und $\beta_2 = 0.999$ gewählt. Dies entspricht den Angaben der Veröffentlichung [14]. Der Epsilon-Wert wurde von 10^{-8} auf 10^{-6} erhöht, um Instabilitäten bei kleinen Gradienten zu vermeiden. Die Batchsize wurde von 8 auf 12 erhöht, da dies zu einem stabileren Training führte. Größere Batchsizes konnten aufgrund von begrenztem VRAM nicht getestet werden. Als Metrik für das Training wurde der absolute relative Fehler (REL) genutzt. Nach jeder Epoche wurde der REL auf dem Validierungsdatensatz bestimmt. Nach dem Training wurden die Modellparameter gewählt, die auf dem Validierungsdatensatz den geringsten Fehler aufwiesen.

4 Experimente & Evaluation

4.1 Tiefenbestimmung mit optischem Fluss

Der in Abschnitt 3.2.1 beschriebene Algorithmus zur Tiefenbestimmung mit optischem Fluss wurde auf den Daten einer bag-Datei mit insgesamt 7510 verschiedenen Bildpaaren getestet. Dabei wurde der relative absolute Fehler (REL) in Bezug auf die LiDAR-Daten errechnet. Über alle Instanzen gemittelt beträgt dieser 0,736, das bedeutet, im Durchschnitt wird die Entfernung um 73,6 % falsch eingeschätzt. Für die Berechnung wurden alle Datenpunkte, bei denen das Fahrzeug absolut still stand, nicht mit einbezogen. Alle fehlenden Werte in der Vorhersage oder der Ground Truth wurden ebenfalls nicht in die Fehlerberechnung mit einbezogen. Als Vergleich wurde der Mittelwert über alle gemessenen Tiefenbilder des Datensatzes errechnet und als Vorhersage getestet. Der REL für diese Baseline beträgt 0,711. Im allgemeinen Fall kann der Algorithmus also keine besseren Ergebnisse als der Durchschnittswert liefern.

Weiterhin wurde der Fehler in einem zentralen Bildausschnitt, der dem zum Trainieren des neuronalen Netzes identisch ist (siehe Abbildung 15a), berechnet. Dabei wurde der Algorithmus auf dem Gesamtbild ausgeführt, aber die Ränder für die Fehlerberechnung nicht beachtet. Hier beträgt der REL nur 0,682. Dies lässt sich dadurch begründen, dass der optische Fluss am Bildrand nur ungenau bestimmt werden kann. Die Bildränder im ersten Bild sind im zweiten Bild bereits nicht mehr sichtbar, wodurch sich der optische Fluss nicht genau bestimmen lässt. Der daraus resultierende Effekt, dass die Bildränder sehr ungenau eingeschätzt werden, ist in Abbildung 18 gut zu erkennen.

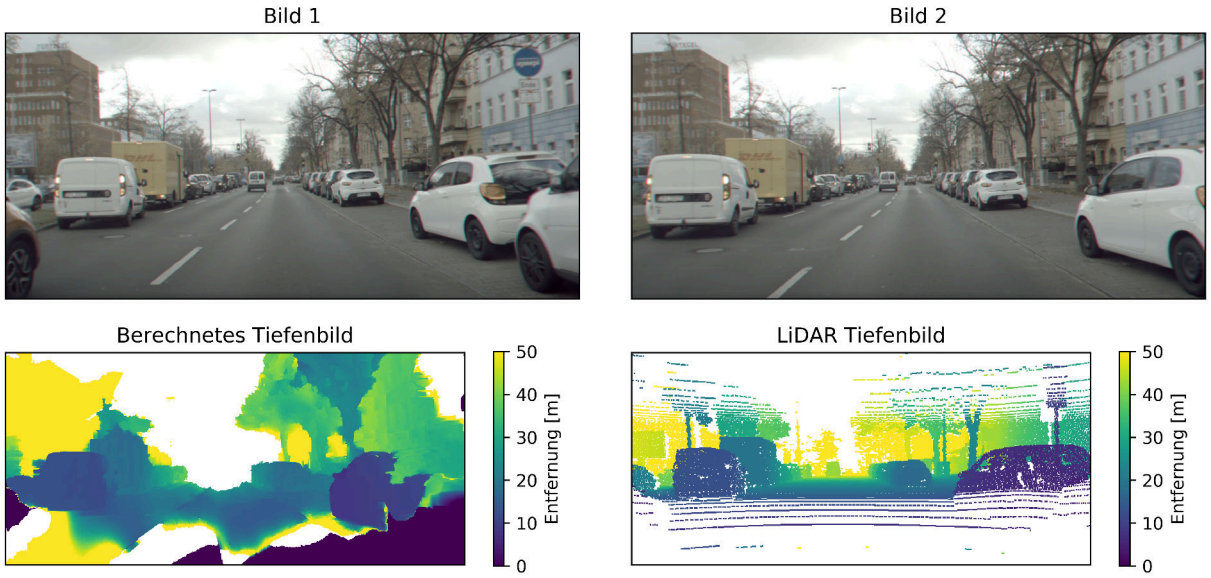


Abbildung 18: Exemplarische Veranschaulichung der mit dem optischen Fluss berechneten Entfernung und der dazugehörigen Ground Truth. Der durchschnittliche REL für das gesamte Bild ist 0,723, im zentralen Bildbereich beträgt er 0,308.

Der Fehler wurde weiterhin in Geschwindigkeitsbereiche unterteilt. Diese Unterteilung ist grafisch in Abbildung 19 dargestellt. Dabei stellen die Balken jeweils den mittleren REL für alle Datenpunkte in dem Geschwindigkeitsbereich dar. Jeder Balken repräsentiert einen Bereich von 0,5 m/s. In Blau ist der REL für das Gesamtbild dargestellt, in Orange der REL für den zentralen Bildausschnitt. Es scheint einen klaren Zusammenhang zwischen Geschwindigkeit und Qualität der Vorhersage zu geben. Es ist eine bestimmte Mindestgeschwindigkeit notwendig, um mit dem Algorithmus bessere Vorhersagen zu erhalten. Dies entspricht der Erwartungshaltung,

da die Grundidee des Algorithmus auf dem durch die Fahrt verursachten optischen Fluss beruht. Aufgrund dieser Tatsache wurde der REL zusätzlich nur für Geschwindigkeiten über 2 m/s berechnet. In diesem Fall beträgt er 0,628, bzw. 0,536 für den zentralen Bildausschnitt.

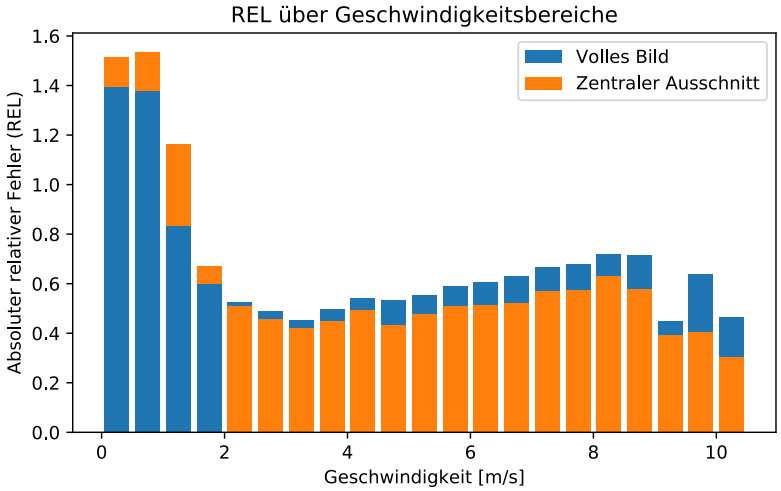


Abbildung 19: Darstellung des REL für verschiedene Geschwindigkeitsbereiche. In Blau ist der REL für das gesamte Bild dargestellt, in Orange der REL für den zentralen Bildausschnitt.

In Abbildung 18 sind zwei weitere Probleme des Algorithmus zu erkennen. Diese sind die Einschätzung von bewegten Objekten und die Einschätzung im und unmittelbar um den Bildmittelpunkt. Die bewegten Objekte erzeugen selbst einen optischen Fluss, wodurch die Einschätzung dieser Objekte verfälscht wird. Andere Verkehrsteilnehmer haben meist eine ähnliche Geschwindigkeit wie die des eigenen Fahrzeugs. Ihre relative Position im Bild verändert sich daher bei aufeinanderfolgenden Bildern nicht und der optische Fluss ist nahezu null. Die daraus berechneten Geraden für die Entfernungseinschätzung sind dementsprechend parallel, wodurch sich keine Entfernung berechnen lässt. Der Effekt im zentralen Bildbereich ist ein ähnlicher. Bei der Bewegung geradeaus ist der optische Fluss im Bildzentrum generell kleiner als am Bildrand. Die für den Bildmittelpunkt berechneten Geraden sind daher auch annähernd parallel, wodurch die Entfernung nicht genau berechnet werden kann. Weiterhin kommt es aufgrund des geringen optischen Flusses im Bildmittelpunkt zu größeren Ungenauigkeiten, da der optische Fluss aufgrund der Pixelauflösung des Bildes nur begrenzt genau bestimmt werden kann. Für einen besseren Überblick über die Vorhersage sind im Anhang ab Abbildung 27 weitere Beispiele gegeben.

In Tabelle 2 sind die Ausführungszeiten des Algorithmus auf einem Laptop mit einem Ryzen 5 2500U Prozessor für verschiedene Bildauflösungen gegeben. Selbst bei einer Reduktion der Auflösung auf ein Viertel des Originalbildes beträgt die erreichbare Wiederholungsrate nur knapp über 6 Hz. Dies liegt deutlich unter den 10 Hz, die durch den aktuell genutzten LiDAR-Scanner erreicht werden. Die Wiederholungsrate wird allerdings vor allem durch die Berechnung des optischen Flusses beschränkt. Da diese durch die Bibliothek open-cv durchgeführt wird, ist die langsame Wiederholungsrate nicht auf eine ineffiziente eigene Implementierung zurückzuführen.

Insgesamt liefert der Ansatz der Tiefenbestimmung mit optischem Fluss keine zufriedenstellenden Ergebnisse. Selbst bei Betrachtung des zentralen Bildbereichs und Geschwindigkeiten über 2 m/s kann die Vorhersage durch ein Durchschnittsbild nur geringfügig übertroffen werden. Hinzu kommen die Probleme bei der Vorhersage bewegter Objekte und dem Bereich um den Bildmittelpunkt, sowie die langsame Ausführungszeit. Besonders die Bildmitte, in der sich häufig andere Verkehrsteilnehmer befinden, scheint für das autonome Fahren relevant. Die anderen

Tabelle 2: Ausführungszeit des Algorithmus für verschiedene Bildauflösungen.

	Bildauflösung		
	910 x 420	455 x 210	227 x 105
Berechnung optischer Fluss	1,3 s	0,34 s	0.14 s
Aufstellung LGS	0,063 s	0,015s	0.0028 s
Lösung LGS	0,18 s	0,071 s	0.014 s
Summe	1,543 s	0,426 s	0,1568 s

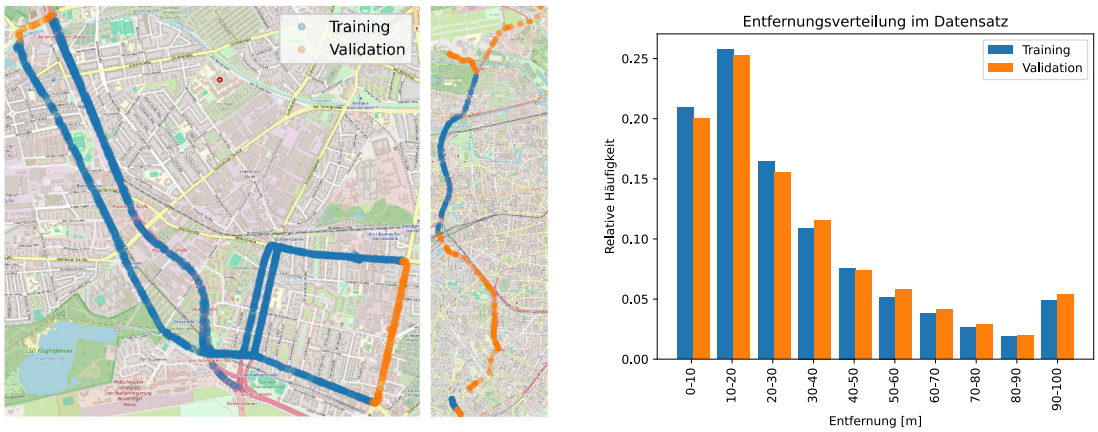
Verkehrsteilnehmer sind weiterhin meist in Bewegung und sollten trotzdem gut bestimmbar sein. Daher scheint dieser Ansatz für eine genaue Tiefenbestimmung nicht nutzbar.

4.2 Tiefenbestimmung mit neuronalem Netz

4.2.1 Datensatz

Der zum Trainieren des neuronalen Netzes erstellte Datensatz enthält nach der manuellen Auswahl 2910 Bild-Tiefenbild-Paare aus 10 verschiedenen Bags. Eine genaue Auflistung der Bags und der Anzahl der daraus gewählten Datenpunkte ist im Anhang in Tabelle 5 gegeben. Die Bags sind aus unterschiedlichen Jahreszeiten ausgewählt und verfügen alle über klare Sichtverhältnisse.

Für das Training wurde der Datensatz im Verhältnis 80/20 (2326/584) in zwei Teile untergliedert, den Trainingsdatensatz und den Validierungsdatensatz. Die Modelle werden auf dem Trainingsdatensatz trainiert und auf dem vorher ungesehenen Validierungsdatensatz evaluiert. Um eine Überanpassung zu vermeiden, wurden der Trainings- und Validierungsdatensatz nach Straßen getrennt, d.h. Aufnahmen aus einer Straße kommen nur in einem der beiden Datensätze vor. Dies soll verhindern, dass die Modelle die Struktur der Straßen auswendig lernen und so die Validierungsergebnisse besser erscheinen als eine Vorhersage auf ungesehenen Straßen. Die Auftrennung ist in Abbildung 20a abgebildet.



(a) (b)

Abbildung 20: a: Darstellung der GPS Koordinaten der Datenpunkte, getrennt in Validations- und Trainingsdatensatz. **b**: Histogramm der relativen Häufigkeiten der Entfernungen in den Tiefenbildern des Datensatzes, getrennt in Validations- und Trainingsdatensatz.

Bei der Aufteilung der Daten wurde weiterhin darauf geachtet, dass die Verteilung der unterschiedlichen Entfernungen im Tiefenbild in beiden Datensätzen ähnlich ist. Diese Verteilung

ist für beide Datensätze im Histogramm in Abbildung 20b dargestellt. Bei der Verteilung der Entfernungen im Datensatz ist ein Ungleichgewicht zu kleineren Entfernungen zu erkennen. Der Bereich von 90 m - 100 m weicht von diesem Trend leicht ab, da die als Himmel erkannten Bildbereiche in diesem Segment dazugezählt werden.

Ein weiterer wichtiger Aspekt ist die Verteilung der Werte der Tiefenbilder in Bezug auf die Bildbereiche. Generell können die Pixel drei Arten von Werten haben. Sie können als numerischer Wert eine gemessene Entfernung repräsentieren, sie können den Himmel darstellen, was im Tiefenbild durch den Wert „inf“ repräsentiert wird, oder es kann der Wert des Pixels fehlen. Ein fehlender Wert bedeutet, dass weder ein Messpunkt für dieses Pixel existiert, noch der Himmel an dieser Stelle erkannt wurde. Dies kann auftreten, wenn sich ein Objekt außerhalb der Reichweite des LiDAR-Scanners befindet. Fehlende Werte sind generell unerwünscht, da das Modell auf fehlenden Werten nicht trainiert werden kann. Die Verteilung dieser Wertekategorien für jedes Pixel des Datensatzes ist in Abbildung 21a - 21c abgebildet.

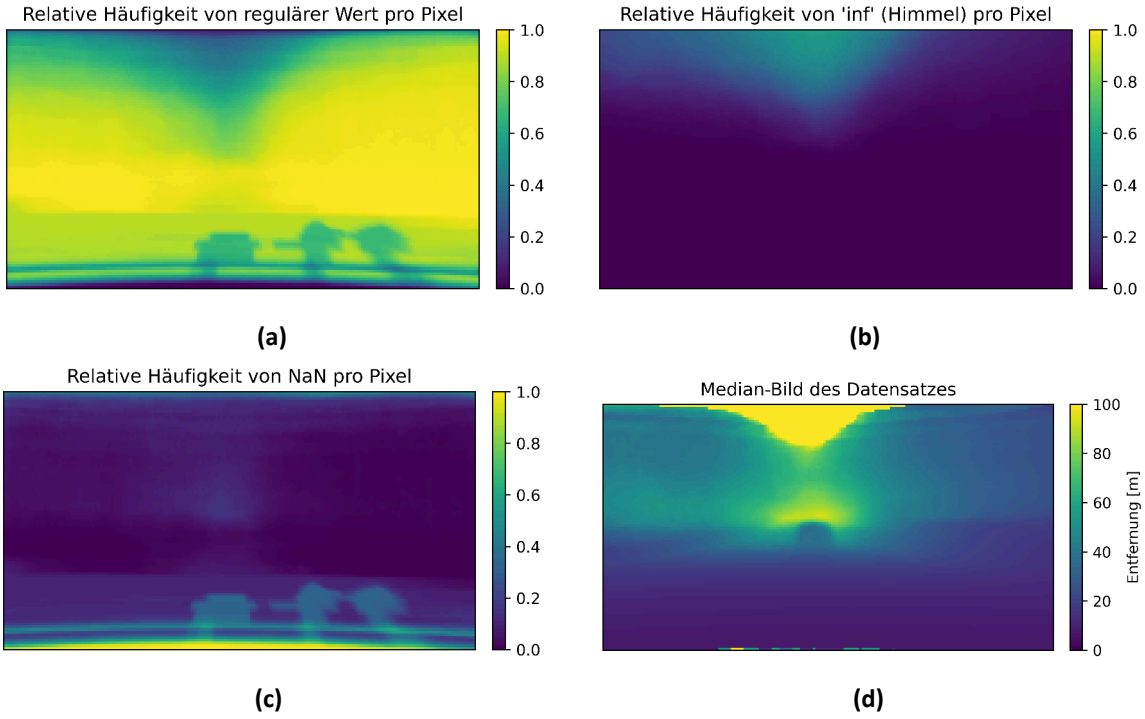


Abbildung 21: Übersicht über die Verteilung der möglichen Werte in Bezug auf die Pixel des Tiefenbildes. **a** zeigt die relative Häufigkeit der numerischen Werte. **b** zeigt die relative Häufigkeit des Wertes „inf“, welcher den Himmel repräsentiert. **c** zeigt die relative Häufigkeit des Wertes „NaN“, also einen fehlenden Wert. **d** zeigt das Median-Tiefenbild des Datensatzes.

Abbildung 21a zeigt, dass der Großteil des Bereiches des Tiefenbildes mit direkten Messwerten des LiDAR-Scanners gefüllt ist. Dabei ergeben sich im oberen und unteren Bildbereich aber deutliche Abweichungen. Im unteren Bildbereich kommt es vermehrt zu fehlenden Werten, wie in Abbildung 21c erkennbar ist. Dies liegt an den Scanlinien des LiDAR-Scanners, die zum oberen und unteren Bildrand weniger dicht liegen. Weiterhin sind in einigen Aufnahmen andere Sensoren vor dem LiDAR-Scanner auf dem Fahrzeug montiert, welche den Scan an diesen Stellen verdecken. Im oberen Bildbereich kommt es auch vermehrt zu fehlenden Werten. Dies liegt auch an der geringeren Dichte der Scanlinien und an der Tatsache, dass im oberen Bildbereich häufig keine Objekte vorhanden sind, sondern nur der Himmel sichtbar ist. Wird die Existenz des Himmels auch durch die implementierte Himmelerkennung registriert, so können die fehlenden Werte mit dieser Information ersetzt werden, wie in Abbildung 21b zu erkennen ist.

4.2.2 Modellvergleich

Insgesamt wurden drei verschiedene Modellarchitekturen getestet. Das Referenzmodell von Hu et al. wurde mit ResNet50-Backbone („ResNet Pretrain“) und SENet154-Backbone („SENet No Pretrain“) getestet. Für das ResNet50-Backbone wurden die auf ImageNet vortrainierten Gewichte genutzt. Um einen Vergleich mit dem SENet154 zu ermöglichen, für das keine vortrainierten Gewichte verfügbar waren, wurde auch das Modell mit ResNet50-Backbone ohne vortrainierte Gewichte („ResNet No Pretrain“) getestet. Weiterhin wurde für einen Vergleich das MFF Modul entfernt und stattdessen die Zwischenausgaben des Encoders an die Eingabe der entsprechenden Decoder-Ebene konkateniert („ResNet UNet Pretrain“). Dafür wurden die Anzahl der Featuremaps im jeweiligen Decoder-Block entsprechend erhöht. Diese Struktur ähnelt der Architektur von UNet [25]. Die Ergebnisse der allgemeinen Metriken sowie die Parameteranzahl sind in Tabelle 3 abgebildet.

Tabelle 3: Vergleich der verschiedenen Modelle mit der Baseline und der Referenz [14]. Das beste Modell auf dem selbsterstellten Datensatz ist jeweils hervorgehoben.

Modell	REL	RMSE	LOG10	$ACC_{1,25}$	$ACC_{1,25^2}$	$ACC_{1,25^3}$	Parameter
ResNet Pretrain	0,182	14,726	0,084	0,715	0,879	0,948	68 Mio.
ResNet No Pretrain	0,252	17,132	0,108	0,663	0,82	0,898	68 Mio.
SENet No Pretrain	0,206	15,116	0,092	0,702	0,857	0,927	157 Mio.
ResNet UNet Pretrain	**0,171**	**13,510**	**0,077**	**0,753**	**0,893**	**0,951**	81 Mio.
Baseline - Median	0,5262	22,688	0,155	0,518	0,693	0,826	-
Baseline - Mean	0,564	21,504	0,160	0,478	0,682	0,841	-
Referenz ResNet auf NYU Depth V2 [14]	0,126	0,555	0,054	0,843	0,968	0,991	68 Mio.

Als Baseline wurde für jedes Pixel des Tiefenbildes der Median und Mittelwert über dem Trainingsdatensatz vorhergesagt. Die Median-Baseline-Vorhersage ist in Abbildung 21d abgebildet und ist generell etwas besser als die Mittelwert-Baseline. Weiterhin wurden die Ergebnisse der Referenzveröffentlichung von Hu et al. [14] auf dem NYU Depth V2-Datensatz [22] eingefügt. Diese Werte dienen nur als Referenz, da Sie auf einem anderen Datensatz erreicht wurden. Insbesondere der skalierungsabhängige RMSE kann nicht verglichen werden.

Insgesamt setzen sich alle getesteten Modelle gut von der Baseline ab. Anhand der Metriken gibt sich eine klare Reihenfolge der Modelle. Die Modelle mit vortrainierten Backbones erzielen bessere Ergebnisse als solche mit zufällig initialisierten Backbones. Des Weiteren erweist sich das Ersetzen des von Hu et al. genutzten MFF durch UNet-artige direkte Verbindungen zwischen Encoder und Decoder als eine Verbesserung. Das „ResNet UNet Pretrain“ Modell erzielt auf allen Metriken die besten Ergebnisse. Es besitzt allerdings auch mehr Parameter als das äquivalente Modell mit MFF-Modul. Das „ResNet Pretrain“-Modell erzielt etwas schlechtere Ergebnisse als „ResNet UNet Pretrain“, liefert aber bessere Ergebnisse als „SENet No Pretrain“. Besonders auffällig ist aber der Unterschied zwischen den Modellen mit untrainierten SENet-Backbone und untrainiertem ResNet-Backbone. Das Modell „SENet No Pretrain“ erzielt in Bezug auf den REL fast 20 % (5 Prozentpunkte) bessere Ergebnisse und ist auch in allen anderen Metriken besser. Dies könnte bedeutet, dass die Modelle mit vortrainiertem SENet-Backbone noch bessere Ergebnisse als die mit vortrainiertem ResNet-Backbone erzielen können. Dies würde sich mit den Ergebnissen von Hu et al. decken [14]. Da in TensorFlow die auf ImageNet vortrainierten Gewichte für das SENet154-Backbone nicht verfügbar sind, konnte diesbezüglich allerdings kein Vergleich durchgeführt werden.

Weiterhin wurde die Struktur der Vorhersage mit der edge-accuracy [14] mit drei verschiedenen

Schwellwerten analysiert. Ein größerer Schwellwert bedeutet eine stärkere Kante im Bild. Die Daten sind in Tabelle 4 abgebildet.

Tabelle 4: Vergleich der edge-accuracy [14] der getesteten Modelle für verschiedene Schwellwerte. Größere Schwellwerte bedeuten, dass nur stärkere Kanten beachtet werden. Das beste Modell ist jeweils hervorgehoben.

Schwellwert	Modell	Precision	Recall	F1-Score
2,5	ResNet Pretrain	0,595	0,899	0,716
	ResNet No Pretrain	0,583	0,897	0,707
	SENet No Pretrain	0,607	0,908	**0,728**
	ResNet Unet Pretrain	**0,611**	0,893	0,725
	Baseline - Median	0,582	**0,924**	0,714
	Baseline - Mean	0,570	0,902	0,698
5	ResNet Pretrain	0,535	0,718	0,613
	ResNet No Pretrain	0,518	0,702	0,596
	SENet No Pretrain	0,555	0,723	0,628
	ResNet Unet Pretrain	**0,561**	0,716	**0,629**
	Baseline - Median	0,488	**0,766**	0,597
	Baseline - Mean	0,489	0,635	0,553
10	ResNet Pretrain	0,479	0,709	0,571
	ResNet No Pretrain	0,456	0,650	0,536
	SENet No Pretrain	0,497	0,707	0,584
	ResNet Unet Pretrain	**0,514**	0,703	**0,594**
	Baseline - Median	0,401	**0,721**	0,515
	Baseline - Mean	0,446	0,541	0,489

Das Modell „ResNet UNet Pretrain“ erzielt wieder die besten Ergebnisse. Allerdings sind die Unterschiede deutlich geringer als bei den anderen Metriken. Für schwache Kanten (Schwellwert 2,5) erzielt das „SENet No Pretrain“-Modell sogar einen besseren F1-Score. Auffällig ist auch der Unterschied in Precision und Recall zwischen der Baseline und den Modellen. Während die Modelle sich in Bezug auf Precision deutlich von der Baseline absetzen können, bleibt der Recall ein Schwachpunkt, bei dem die Baseline generell die besten Ergebnisse liefert. Das bedeutet, dass eine durch ein Modell vorhergesagte Kante eher korrekt ist, aber dadurch auch mehr Kanten nicht als solche identifiziert werden.

Ein Grund für das schwache Absetzen gegenüber der Baseline in Bezug auf die Kantenerkennung ist die Unschärfe der Vorhersagen der Modelle. Dies ist auch in den Beispielabbildungen 23, 24 und 25 erkennbar. Die allgemeine Struktur der Szene wird richtig erkannt, aber die Übergänge zwischen Objekten, welche in der Ground Truth typischerweise sprunghaft sind, werden durch die Modellvorhersagen über einen größeren Pixelbereich gestreckt. Dies lässt sich besonders gut in der jeweiligen Darstellung einer Pixelzeile erkennen.

Im Folgenden werden nun die Vorhersagen des Modells „ResNet UNet Pretrain“ genauer betrachtet. Bei den in diesem Abschnitt gezeigten Beispielen handelt es sich immer um die Vorhersagen dieses Modells.

Das Histogramm in Abbildung 22a zeigt die Verteilung des RELs für die Vorhersagen des Modells auf dem Validierungsdatensatz. Es ist gut erkennbar, dass der Großteil der Vorhersagen um den Durchschnittswert des RELs streuen, aber einige starke Ausreißer existieren. Weiterhin ist in Abbildung 22b, welche die Verteilung des RELs über Entfernungsbereiche darstellt, zu erkennen, dass die nahen Bereiche deutlich besser vorhergesagt werden als die weit entfernten Bereiche.

Eine Ausnahme davon ist die Einschätzung des Himmels, die durch den Balken über 100 m repräsentiert wird.

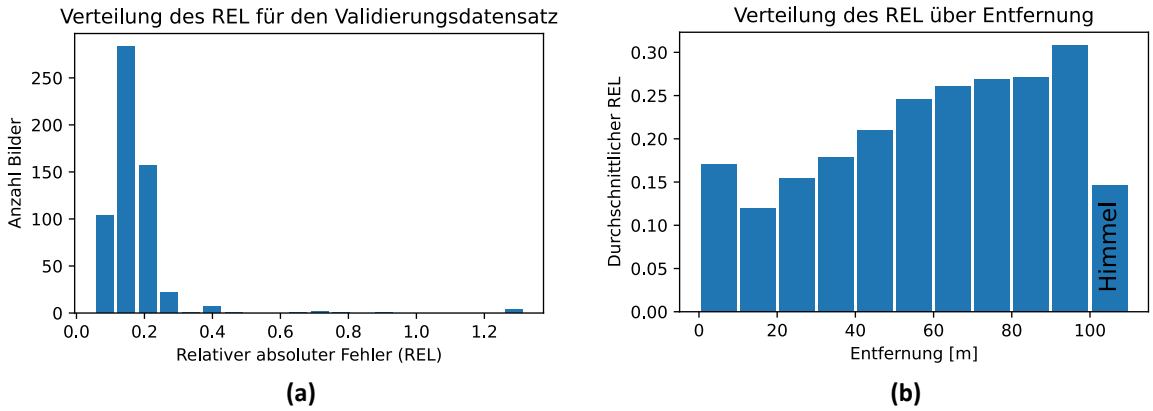


Abbildung 22: Verteilung des RELs für das UNet-artige Modell mit vortrainiertem ResNet50-Backbone. **a**: Histogramm des durchschnittlichen RELs für die Bilder des Validierungsdatensatzes. **b**: Histogramm des durchschnittlichen RELs für Entfernungssegmente. Der Balken über 100 m repräsentiert die Himmelwerte.

Ein möglicher Grund für die schlechteren Einschätzungen für ‚weite' Entfernungen könnte die in Abschnitt 4.2.1 gezeigte Verteilung der Entfernungen im Datensatz sein. Da die größeren Entfernungen im Datensatz deutlich weniger repräsentiert sind, kann das Modell für diesen Bereich nicht so gut trainiert werden wie für den Nahbereich. Ein weiterer Aspekt ist die Auflösung des Bildes, auf dem das Modell die Vorhersage trifft. Je größer die Entfernung eines Objektes ist, desto kleiner erscheint es im Bild und desto weniger Pixel nimmt es folglich ein. Eine höhere Auflösung führt dazu, dass auch weiter entfernte Objekte detaillierter dargestellt werden können, was die Vorhersage verbessern könnte. Dies würde aber auch zu langsameren Ausführungszeiten führen, da mehr Rechenoperationen ausgeführt werden müssen.

Für das Modell wurde auch die Ausführungszeit der Vorhersage auf allen Bildern des Datensatzes getestet. Dabei wurden immer Einzelbilder vorhergesagt. Eine Vorhersage von mehreren Bildern gleichzeitig führt üblicherweise zu kürzeren Ausführungszeiten pro Bild, ist aber im Kontext des autonomen Fahrens unrealistisch, da die Tiefeninformationen in Echtzeit benötigt werden. Die Ausführungszeit wurde auf einem MacBook Pro M1 getestet. Sie beträgt im Mittelwert 0.061 s mit einer Standardabweichung von 0.009 s. Das Modell kann also mit einer Wiederholungsrate von etwa 16 Hz ausgeführt werden. Da die Wiederholungsrate des LiDAR-Scanners etwa 10 Hz beträgt, ist eine Wiederholungsrate von 16 Hz als sehr gutes Ergebnis zu werten. Vergleichsweise beträgt die gemessene Ausführungszeit des Referenzmodells von Hu et al. mit ResNet50-Backbone 0.083 s mit einer Standardabweichung von 0.005 s. Das modifizierte Modell erreicht also nicht nur bessere Ergebnisse, sondern auch eine schnellere Ausführungszeit. Es ist hierbei anzumerken, dass die Ausführungszeit stark hardwareabhängig ist. Die Ausführungszeit kann also auf anderen Geräten variieren. Weiterhin wurde nur die reine Vorhersagezeit getestet, bei einer realen Benutzung müssten zumindest die Bilder noch begradigt und beschnitten werden, sowie die Ergebnisse in einem nutzbaren Format, z.B. als Punktwolke, ausgegeben werden.

In den Abbildungen 23, 24 und 25 sind drei Beispiele der Vorhersage des Modells abgebildet. Im Anhang ab Abbildung 29 sind weitere Beispiele gegeben. Die Abbildung 23 zeigt eine Tunnelaufnahme und ist mit einem REL von 0,065 ein Beispiel für eine gute Vorhersage des Modells. Sowohl die beiden Fahrzeuge im mittleren Bildbereich als auch die Tunnelwände werden durch das Modell sehr gut vorhergesagt. Unter den besten Vorhersagen des Modells sind hauptsächlich

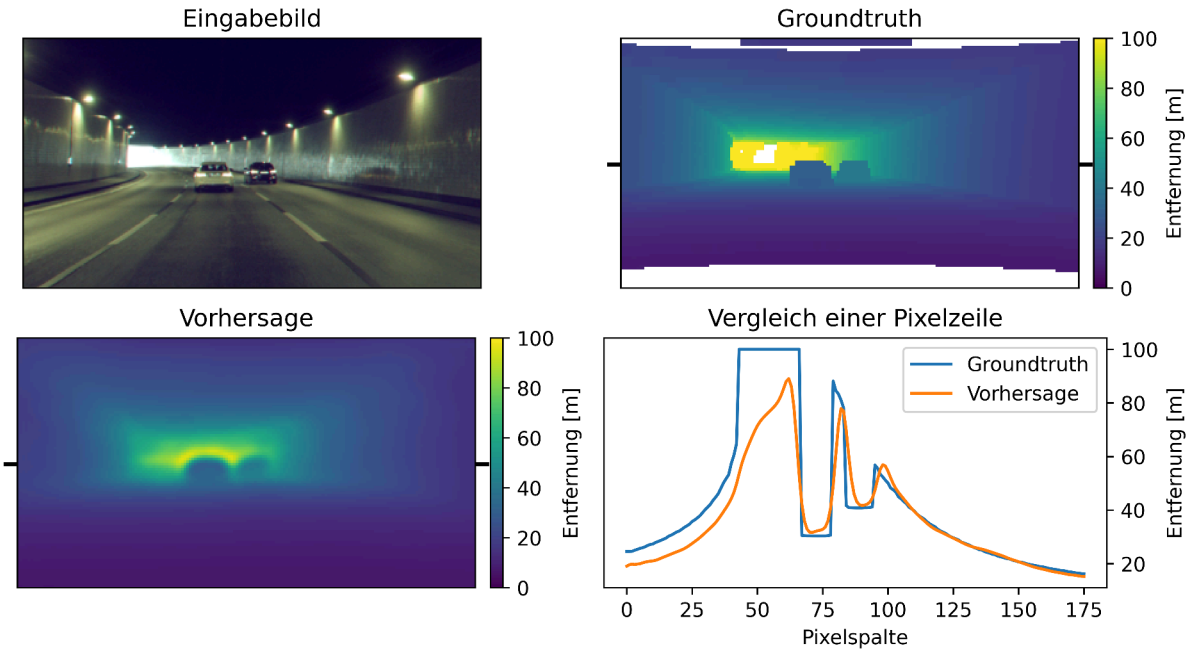


Abbildung 23: Darstellung einer sehr guten Vorhersage des Modells. Der REL beträgt 0,065. Oben links: Eingabebild. Oben rechts: Ground Truth Tiefenbild. Unten links: Vorhersage des Modells. Unten rechts: Vergleich zwischen Ground Truth und Vorhersage in einer Zeile des Bildes. Die Zeile ist in der Ground Truth und Vorhersage markiert.

Tunnelaufnahmen zu finden. Ein Grund dafür, dass Tunnel allgemein besser vorhergesagt werden können, ist, dass es weniger Unregelmäßigkeiten in den Szenen gibt. Anstelle von vielen starken Kanten im Tiefenbild, die durch schmale Objekte wie Bäume und Straßenschilder entstehen, sind am Bildrand nur die Tunnelwände vorhanden. Diese zusammenhängenden Flächen sind für das Modell deutlich besser vorhersagbar wodurch es weniger Abweichungen von der Ground Truth gibt. Die etwas schlechtere Belichtung scheint sich dabei nicht negativ auf die Vorhersage auszuwirken.

Die Abbildung 24 zeigt eine typische Szene im Straßenverkehr mit anderen Verkehrsteilnehmern, Fahrzeugen am Fahrbahnrand und einem Straßenschild. Der REL der Vorhersage des Modells beträgt 0,170, es handelt sich somit um eine durchschnittliche Vorhersage. Die Entfernung der Fahrzeuge wird dabei sehr genau eingeschätzt, was besonders bei Betrachtung der ausgewählten Pixelzeile sichtbar ist. Allerdings gibt es eine gewisse Unschärfe der Kanten, die bereits besprochen wurde. Die Feinstruktur der Szene zwischen den großflächigen Fahrzeugen kann durch das Modell nicht genau aufgelöst werden. Meist sind diese vorhergesagten Distanzen zu klein. Auch die Feinstrukturen der anderen Objekte der Szene sind nicht genau aufgelöst. So setzt sich das Straßenschild in der Vorhersage zwar vom Fahrzeug und der Hauswand ab, es verschwimmt aber mit dem leicht dahinterliegenden Baum. Insgesamt ist diese Vorhersage trotzdem als gut zu bewerten, da besonders die relevanten Aspekte der Szene, die anderen Verkehrsteilnehmer, gut eingeschätzt werden und die Gesamtstruktur der Szene trotz der Unschärfe gut wiedergegeben wird.

Die in Abbildung 22a erkennbaren, starken Ausreißer bezüglich des RELs lassen sich auf eine zusammenhängende Bildsequenz aus 9 Bildern zurückführen. Ein Beispiel aus dieser Sequenz ist in Abbildung 25 dargestellt. Sie geben einen Aufschluss über die Failcases des Modells. In der Abbildung, welche mit einem REL von 1,328 die schlechteste Vorhersage des Modells darstellt, befindet sich ein Bus sehr nahe vor der Kamera und nimmt dadurch den Großteil des Bildes ein.

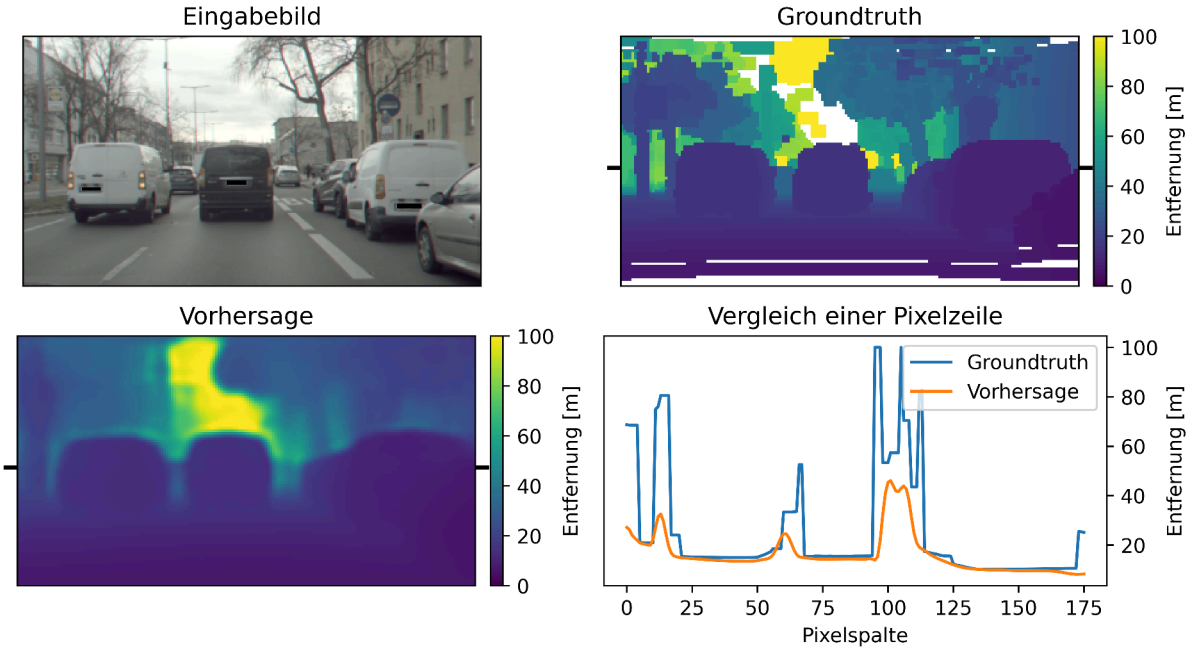


Abbildung 24: Darstellung einer durchschnittlichen Vorhersage des Modells. Der REL beträgt 0,170. Oben links: Eingabebild. Oben rechts: Ground Truth Tiefenbild. Unten links: Vorhersage des Modells. Unten rechts: Vergleich zwischen Ground Truth und Vorhersage in einer Zeile des Bildes. Die Zeile ist in der Ground Truth und Vorhersage markiert.

Die Ränder des Busses werden durch das Modell noch relativ gut eingeschätzt, die einfarbige zentrale Fläche wird aber deutlich zu weit geschätzt. Das Modell scheint Probleme bei der Vorhersage zu haben, wenn große Teile des Bildes verdeckt sind. Hierbei ist anzumerken, dass im Trainingsdatensatz keine ähnliche Szene vorhanden ist. Es handelt sich um eine Art von Szene, auf der das Modell nicht trainiert wurde. Eine Erweiterung des Trainingsdatensatzes um ähnliche Szenen könnte daher zu besseren Ergebnissen führen.

Ein weiteres Problem, dass in dieser Sequenz von Bildern auftritt, ist starkes Gegenlicht und dadurch entstehende Blendenflecke („lens flare“). Dies kann im Anhang in Abbildung 37 beobachtet werden. Die Blendenflecke verdecken einen Teil des Bildes. Dieser Bereich wird daher vom Modell schlecht eingeschätzt.

Insgesamt ist davon auszugehen, dass Bedingungen, welche die Sicht einschränken, zu einer schlechteren Vorhersage des Modells führen. Dies könnten nicht nur Gegenlicht, sondern auch starker Niederschlag oder Nebel sein. Diese Sichtbedingungen sind im Datensatz nicht repräsentiert. Daher kann auch nicht genauer geprüft werden, wie sich diese Bedingungen auf die Vorhersage auswirken. Im Datensatz sind allerdings Aufnahmen mit liegendem Schnee enthalten. Hier konnte keine auffällige Verschlechterung der Vorhersage festgestellt werden. Für ein einsatzfähiges System ist eine genaue Analyse der anderen Fälle aber notwendig.

Ein Vergleich zwischen den Ergebnissen der Tiefenschätzung mittels optischem Fluss und der Tiefenschätzung durch das neuronale Netz zeigt, dass die Vorhersagen des neuronalen Netzes deutlich besser sind. Dies wird nicht nur durch die Betrachtung der Metriken deutlich, sondern auch durch die Tatsache, dass die Vorhersage des neuronalen Netzes dicht ist, also für jedes Pixel des Bildes eine Vorhersage macht. Dennoch sind die erreichten Genauigkeiten noch verbesserungswürdig. Sie scheinen in dieser Form nicht für den Einsatz in einem autonomen Fahrzeug nutzbar.

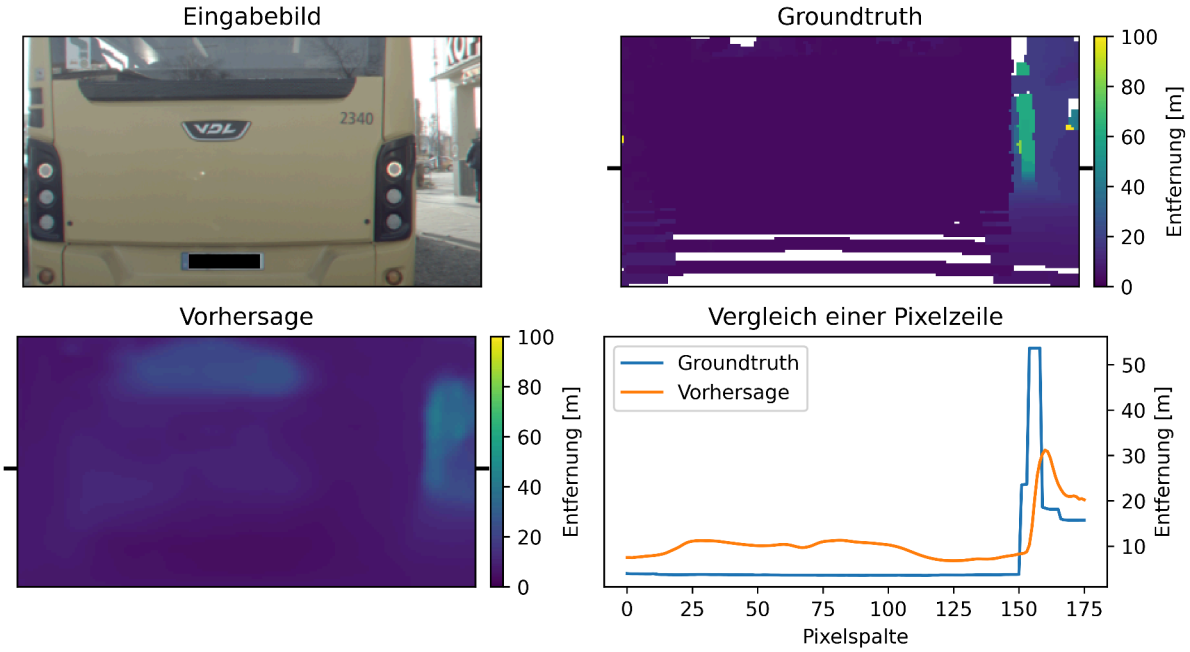


Abbildung 25: Darstellung der schlechtesten Vorhersage des Modells. Der REL beträgt 1,318. Oben links: Eingabebild. Oben rechts: Ground Truth Tiefenbild. Unten links: Vorhersage des Modells. Unten rechts: Vergleich zwischen Ground Truth und Vorhersage in einer Zeile des Bildes. Die Zeile ist in der Ground Truth und Vorhersage markiert.

5 Zusammenfassung und Ausblick

In dieser Arbeit wurden zwei Ansätze zur Tiefenschätzung aus Kamerabildern getestet. Der eine beruht auf dem optischen Fluss einer Bildsequenz und der bekannten Bewegung der Kamera, der andere nutzt ein neuronales Netz, um aus Einzelbildern die Tiefe jedes Pixels vorherzusagen. Die Ansätze wurden auf den Daten des „Made in Germany“-Fahrzeugs des Dahlem Center for Machine Learning and Robotics der Freien Universität Berlin getestet. Dafür wurden aus den aufgezeichneten Frontkamera-Aufnahmen und LiDAR-Daten des Fahrzeugs Datensätze für die beiden Methoden erstellt.

Der Ansatz, die Tiefe durch den optischen Fluss und die bekannte Eigenbewegung der Kamera zu schätzen liefert keine guten Ergebnisse. In Bezug auf die Genauigkeit der geschätzten Tiefe kann sich der Ansatz nur leicht von einer Mittelwertschätzung über den genutzten Datensatz absetzen. Die Wiederholungsrate liegt selbst bei starker Reduzierung der Bildauflösung deutlich unter 10 Hz, welche die Wiederholungsrate des genutzten LiDAR-Scanners ist. Daher erscheint dieser Ansatz nicht zielführend.

Für die Tiefenschätzung mit einem neuronalen Netz wurde ein von Hu et al. entworfenes Modell [14] mit ResNet50- und SENet154-Backbone getestet [12, 15]. Dabei konnten durch eine Veränderung der Struktur des Modells die Ergebnisse auf dem genutzten Datensatz sowohl in Bezug auf die Qualität als auch auf die Laufzeit verbessert werden. Der erzielte mittlere absolute relative Fehler beträgt 17 %, das bedeutet, dass die Entfernung jedes Pixels im Schnitt um 17 % falsch eingeschätzt wird. Die erzielbare Wiederholungsrate liegt mit etwa 16 Hz gut über der des LiDAR-Scanners. Diese Ergebnisse sind zwar deutlich besser als die mit dem optischen Fluss berechneten Tiefenschätzungen, jedoch scheint eine Fehleinschätzung von durchschnittlich 17 % für den Einsatz in einem autonomen Fahrzeug zu hoch.

Ein Vergleich der Backbones zeigt, dass ein auf ImageNet vortrainiertes ResNet50-Backbone die besten Ergebnisse liefert. Allerdings konnten für das SENet154-Backbone keine vortrainierten Gewichte geladen werden, da diese in TensorFlow nicht verfügbar sind. Im Vergleich ohne vortrainierte Gewichte erzielte das Modell mit SENet154-Backbone deutlich bessere Ergebnisse. Die Nutzung der vortrainierten Gewichte mit SENet154-Backbone könnte also zu einer Verbesserung der Ergebnisse führen. Sie könnten aus einem gespeicherten PyTorch-Modell nach TensorFlow konvertiert werden.

Es konnte weiterhin eine Korrelation zwischen der Nähe von Objekten und der Genauigkeit der Vorhersage festgestellt werden. Objekte, die näher an der Kamera sind, wurden besser eingeschätzt als solche, die weiter entfernt sind. Dies könnte auf ein Ungleichgewicht im Datensatz zurückzuführen sein, allerdings ist auch eine zu geringe Bildauflösung eine mögliche Ursache. Eine Erhöhung der Auflösung könnte also zu besseren Ergebnissen führen. Diese wurde reduziert, um die Geschwindigkeit der Vorhersage zu erhöhen. Im Kontext des autonomen Fahrens muss hier eine Abwägung zwischen Geschwindigkeit der Vorhersage und einem potenziellen Verlust an Genauigkeit getroffen werden.

Um die Ergebnisse der Vorhersage weiter zu verbessern, erscheint eine Erweiterung des Datensatzes sinnvoll. Andere Datensätze für die Tiefenschätzung, wie der NYU Depth V2 oder KITTI-Depth Datensatz, bestehen z.T. aus über 100.000 Bild-Tiefenbild-Paaren [22, 32]. Da das Erstellen eines großen Datensatzes sehr aufwändig ist, könnte eine Alternative sein, die Modelle auf diesen Datensätzen vorzutrainieren. Anschließend können die Modelle dann auf einem kleineren Datensatz, wie dem in dieser Arbeit erstellten, feinjustiert werden.

Auch das Testen von neueren Backbones kann zu einer Verbesserung der Ergebnisse beitragen.

Ein möglicher Kandidat ist das von Tan et al. eingeführte EfficientNet [31], welches bei der Klassifizierung auf ImageNet nicht nur bessere Ergebnisse als ResNet- und SENet-Architekturen liefert, sondern auch weniger Parameter und FLOPs benötigt.

Weiterhin könnten andere Modelle zu einer besseren Vorhersage führen. Ein vielversprechender Ansatz ist es, die Tiefenschätzung nicht als Regressionsproblem, sondern als Klassifizierung auf Tiefenbereichen zu betrachten [8, 9]. Auch die Nutzung neuer Architekturen wie dem Vision Transformer [19] könnte zu besseren Ergebnissen führen.

Literatur

[1] I. Alhashim und P. Wonka, „High Quality Monocular Depth Estimation via Transfer Learning," *ArXiv*, Jg. abs/1812.11941, 2018. Adresse: https://api.semanticscholar.org/CorpusID:57189389.

[2] L. Bennett, B. Melchers und B. Proppe, *Curta: A General-purpose High-Performance Computer at ZEDAT, Freie Universität Berlin*, 2020. Adresse: http://dx.doi.org/10.17169/refubium-26754.

[3] F. Bukhari und M. N. Dailey, „Automatic Radial Distortion Estimation from a Single Image," *Journal of Mathematical Imaging and Vision*, Jg. 45, Nr. 1, S. 31–45, 2013, ISSN: 1573-7683. DOI: 10.1007/s10851-012-0342-2. Adresse: https://doi.org/10.1007/s10851-012-0342-2.

[4] V. Dumoulin und F. Visin, *A guide to convolution arithmetic for deep learning*, 2018. arXiv: 1603.07285 [stat.ML].

[5] T. Durgut und E. E. Maraş, „Principles of self-calibration and visual effects for digital camera distortion," *Open Geosciences*, Jg. 15, Nr. 1, S. 20 220 552, 2023. DOI: doi:10.1515/geo-2022-0552. Adresse: https://doi.org/10.1515/geo-2022-0552.

[6] D. Eigen und R. Fergus, „Predicting Depth, Surface Normals and Semantic Labels with a Common Multi-scale Convolutional Architecture," in *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, S. 2650–2658. DOI: 10.1109/ICCV.2015.304.

[7] D. Eigen, C. Puhrsch und R. Fergus, „Depth Map Prediction from a Single Image Using a Multi-Scale Deep Network," in *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, Ser. NIPS'14, Montreal, Canada: MIT Press, 2014, S. 2366–2374.

[8] S. Farooq Bhat, I. Alhashim und P. Wonka, „AdaBins: Depth Estimation Using Adaptive Bins," in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, Juni 2021. DOI: 10.1109/cvpr46437.2021.00400. Adresse: http://dx.doi.org/10.1109/CVPR46437.2021.00400.

[9] H. Fu, M. Gong, C. Wang, K. Batmanghelich und D. Tao, „Deep Ordinal Regression Network for Monocular Depth Estimation," *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, S. 2002–2011, 2018. Adresse: https://api.semanticscholar.org/CorpusID:46968214.

[10] A. Géron, *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow : concepts, tools, and techniques to build intelligent systems / Aurélien Géron.* eng, Second edition. Beijing ; Boston ; Farnham ; Sebastopol ; Tokyo: O'Reilly, 2019, ISBN: 978-1-492-03264-9.

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

[12] K. He, X. Zhang, S. Ren und J. Sun, „Deep Residual Learning for Image Recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, S. 770–778. DOI: 10.1109/CVPR.2016.90.

[13] D. Hoiem, A. A. Efros und M. Hebert, „Automatic Photo Pop-Up," *ACM Trans. Graph.*, Jg. 24, Nr. 3, S. 577–584, Juli 2005, ISSN: 0730-0301. DOI: 10.1145/1073204.1073232. Adresse: https://doi.org/10.1145/1073204.1073232.

[14] J. Hu, M. Ozay, Y. Zhang und T. Okatani, „Revisiting Single Image Depth Estimation: Toward Higher Resolution Maps With Accurate Object Boundaries," in *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, Los Alamitos, CA, USA: IEEE Computer Society, Jan. 2019, S. 1043–1051. DOI: 10.1109/WACV.2019.00116. Adresse: https://doi.ieeecomputersociety.org/10.1109/WACV.2019.00116.

[15] J. Hu, L. Shen und G. Sun, „Squeeze-and-Excitation Networks," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, S. 7132–7141. DOI: 10.1109/CVPR.2018.00745.

[16] K. Karsch, C. Liu und S. B. Kang, „Depth Extraction from Video Using Non-parametric Sampling," in *Computer Vision – ECCV 2012*, A. Fitzgibbon, S. Lazebnik, P. Perona, Y. Sato und C. Schmid, Hrsg., Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, S. 775–788, ISBN: 978-3-642-33715-4.

[17] D. P. Kingma und J. Ba, „Adam: A Method for Stochastic Optimization," *CoRR*, Jg. abs/1412.6980, 2014. Adresse: https://api.semanticscholar.org/CorpusID:6628106.

[18] K. Kitani, *16-385 Computer Vision: Optical Flow*, Apr. 2017. Adresse: https://www.cs.cmu.edu/~16385/s17/Slides/14.1_Brightness_Constancy.pdf.

[19] A. Kolesnikov u. a., „An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale," 2021.

[20] I. Laina, C. Rupprecht, V. Belagiannis, F. Tombari und N. Navab, „Deeper Depth Prediction with Fully Convolutional Residual Networks," in *2016 Fourth International Conference on 3D Vision (3DV)*, Los Alamitos, CA, USA: IEEE Computer Society, Okt. 2016, S. 239–248. DOI: 10.1109/3DV.2016.32. Adresse: https://doi.ieeecomputersociety.org/10.1109/3DV.2016.32.

[21] B. D. Lucas und T. Kanade, „An Iterative Image Registration Technique with an Application to Stereo Vision," in *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*, Ser. IJCAI'81, Vancouver, BC, Canada: Morgan Kaufmann Publishers Inc., 1981, S. 674–679.

[22] P. K. Nathan Silberman Derek Hoiem und R. Fergus, „Indoor Segmentation and Support Inference from RGBD Images," in *ECCV*, 2012.

[23] V. Patil, C. Sakaridis, A. Liniger und L. Van Gool, „P3Depth: Monocular Depth Estimation with a Piecewise Planarity Prior," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[24] R. Ranftl, A. Bochkovskiy und V. Koltun, „Vision Transformers for Dense Prediction," in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021, S. 12 159–12 168. DOI: 10.1109/ICCV48922.2021.01196.

[25] O. Ronneberger, P. Fischer und T. Brox, „U-Net: Convolutional Networks for Biomedical Image Segmentation," in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, N. Navab, J. Hornegger, W. M. Wells und A. F. Frangi, Hrsg., Cham: Springer International Publishing, 2015, S. 234–241, ISBN: 978-3-319-24574-4.

[26] O. Russakovsky u. a., „ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, Jg. 115, Nr. 3, S. 211–252, 2015. DOI: 10.1007/s11263-015-0816-y.

[27] D. Scharstein, R. Szeliski und R. Zabih, „A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," in *Proceedings IEEE Workshop on Stereo and Multi-Baseline Vision (SMBV 2001)*, 2001, S. 131–140. DOI: 10.1109/SMBV.2001.988771.

[28] T. Senst, V. Eiselein und T. Sikora, „Robust Local Optical Flow for Feature Tracking," *IEEE Transactions on Circuits and Systems for Video Technology*, Jg. 22, Sep. 2012. DOI: 10.1109/TCSVT.2012.2202070.

[29] A. R. Smith, *A Pixel Is Not A Little Square , A Pixel Is Not A Little Square , A Pixel Is Not A Little Square ! (And a Voxel is Not a Little Cube) 1 Technical Memo 6*, 1995. Adresse: https://api.semanticscholar.org/CorpusID:17798361.

[30] P. Sturm, S. Ramalingam, J.-P. Tardif, S. Gasparini und J. Barreto, „Camera Models and Fundamental Concepts Used in Geometric Computer Vision," *Found. Trends. Comput.*

Graph. Vis., Jg. 6, Nr. 1–2, S. 41, Jan. 2011, ISSN: 1572-2740. DOI: 10.1561/0600000023. Adresse: https://doi.org/10.1561/0600000023.

[31] M. Tan und Q. Le, „EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks," in *Proceedings of the 36th International Conference on Machine Learning*, K. Chaudhuri und R. Salakhutdinov, Hrsg., Ser. Proceedings of Machine Learning Research, Bd. 97, PMLR, Juni 2019, S. 6105–6114. Adresse: https://proceedings.mlr.press/v97/tan19a.html.

[32] J. Uhrig, N. Schneider, L. Schneider, U. Franke, T. Brox und A. Geiger, „Sparsity Invariant CNNs," in *International Conference on 3D Vision (3DV)*, 2017.

[33] P. Wang, „Research on Comparison of LiDAR and Camera in Autonomous Driving," in *Journal of Physics Conference Series*, Ser. Journal of Physics Conference Series, Bd. 2093, Nov. 2021, 012032, S. 012 032. DOI: 10.1088/1742-6596/2093/1/012032.

[34] S. Xie, R. B. Girshick, P. Dollár, Z. Tu und K. He, „Aggregated Residual Transformations for Deep Neural Networks," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, S. 5987–5995, 2016. Adresse: https://api.semanticscholar.org/CorpusID:8485068.

A Anhang

Datensatz

Tabelle 5: Übersicht über die für den Datensatz des neuronalen Netzes genutzten Bags und die Anzahl gefilterter Datenpunkte.

Bag Name	Anzahl ausgewählt	Anzahl gesamt
20210210-120755+0100–thiel_3	298	917
20210323-112233+0100–reinickendorf_viereck	85	648
20210323-114248+0100–reinickendorf_autobahn	142	608
20210323-115917+0100–reinickendorf_autobahn_3_laps	410	1537
20211203-114745+0100–lap4	568	1460
20211203-135029+0100–back_to_fu	168	1477
20220311-121546+0100	114	798
20220726-115441+0200–back2fu	210	944
20221110-133557+0100–biglap2_part2	312	782
20221110-135019+0100–biglap3	603	1651

Himmelerkennung

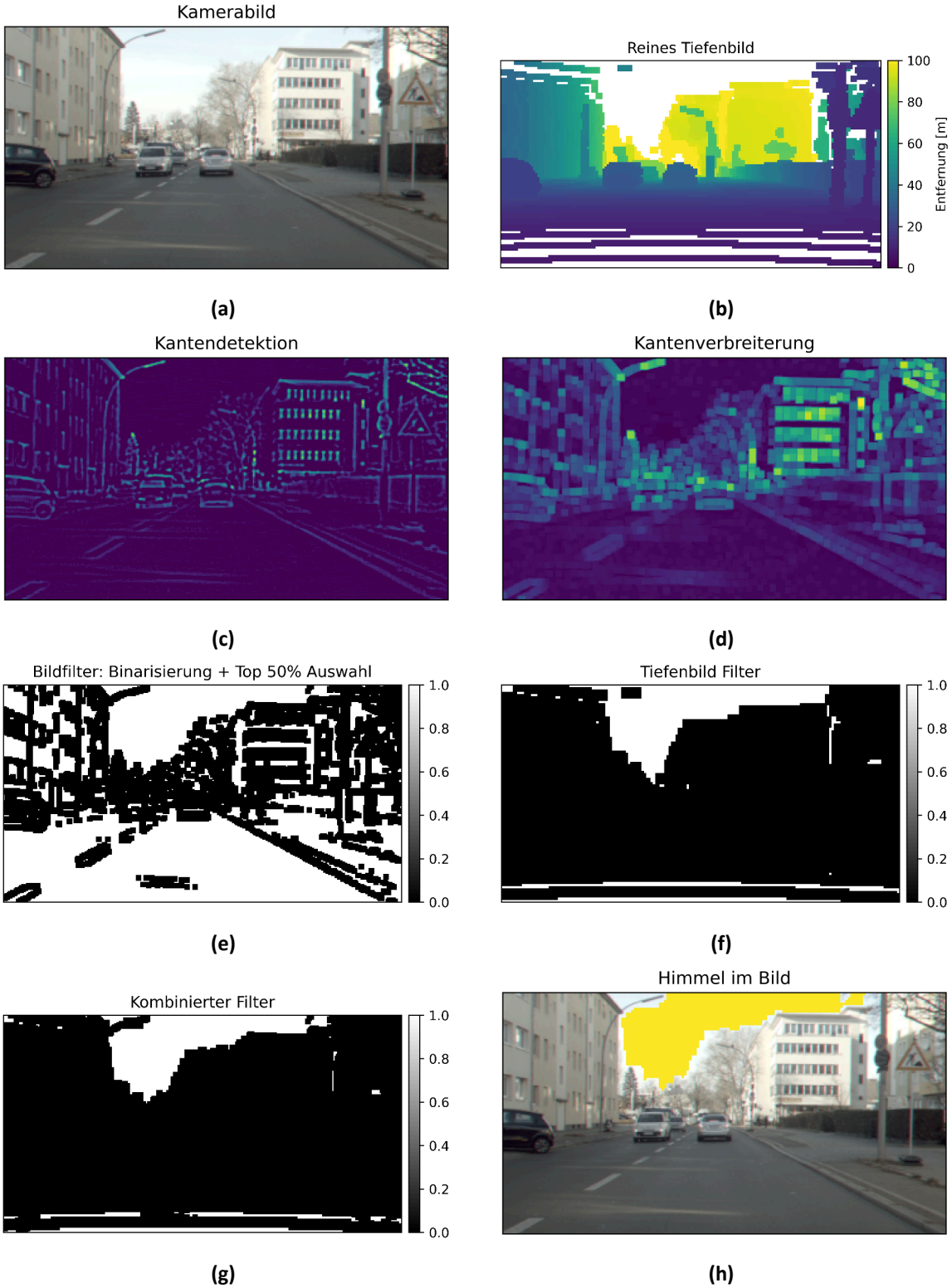


Abbildung 26: Darstellung der Schritte für die Himmelerkennung. Zuerst wird mit Kantenerkennung im Bild eine Maske erstellt (c-e). Diese wird mit der Maske des Tiefenbilds (f) kombiniert, wodurch die Gesamtmaske (g) entsteht. In dieser wird nach zusammenhängenden Bereichen ausgehend von den erlaubten Startpositionen gesucht, die als Himmel markiert werden (h).

Beispiele Tiefenbestimmung mit optischem Fluss

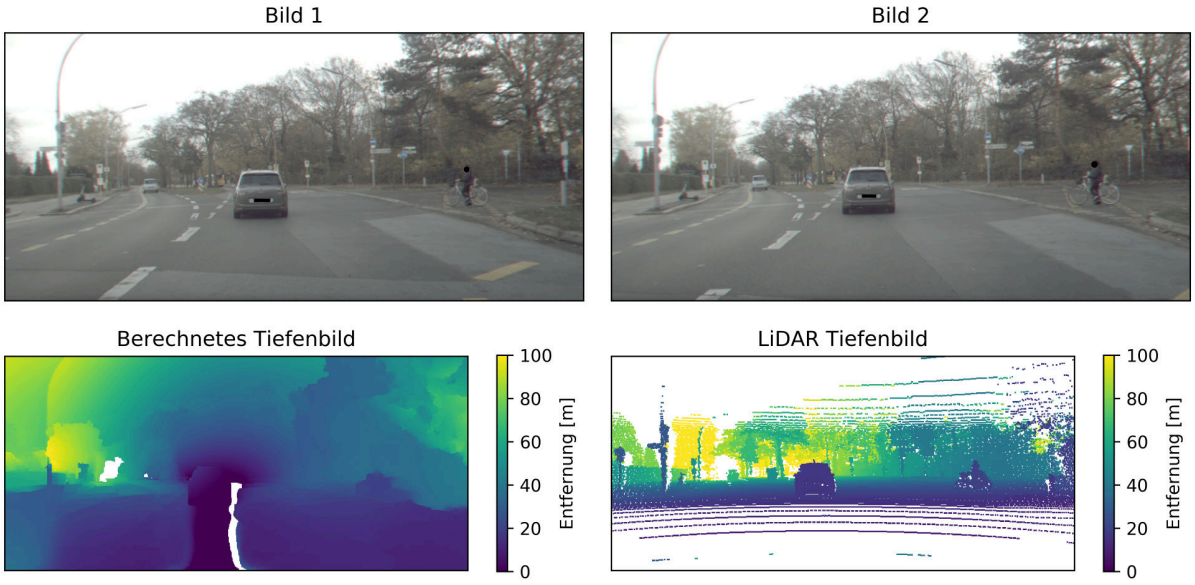


Abbildung 27: Beispiel der Tiefenbestimmung mit optischem Fluss. Der REL im Gesamtbild beträgt 0,484, der REL im zentralen Bildausschnitt beträgt 0,446.

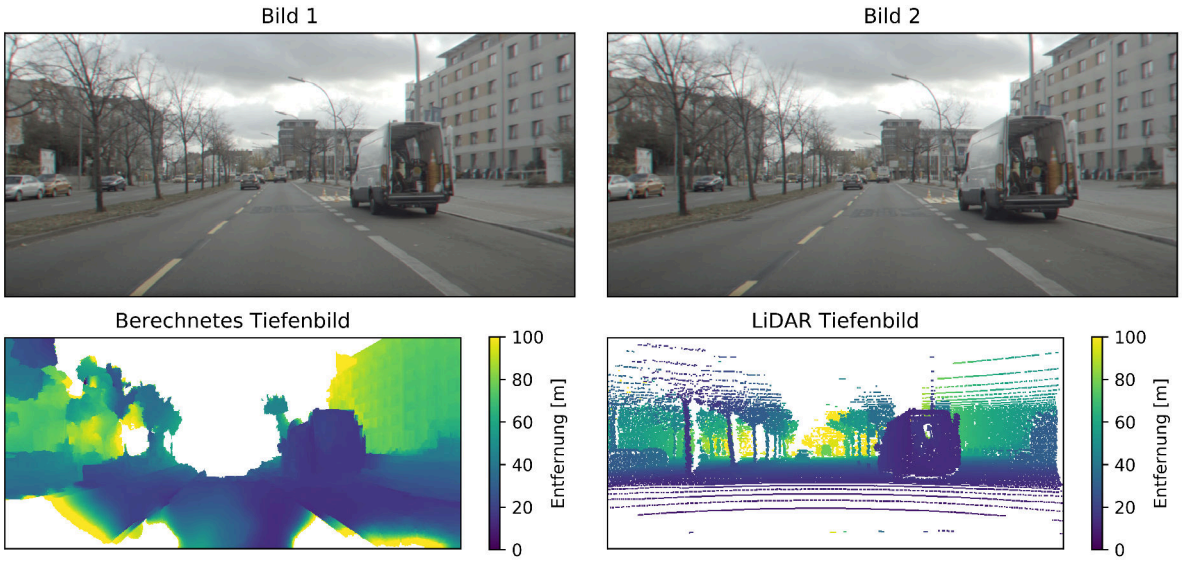


Abbildung 28: Beispiel der Tiefenbestimmung mit optischem Fluss. Der REL im Gesamtbild beträgt 0,642, der REL im zentralen Bildausschnitt beträgt 0,536.

Modellvorhersagen

Beispiele mit gutem REL

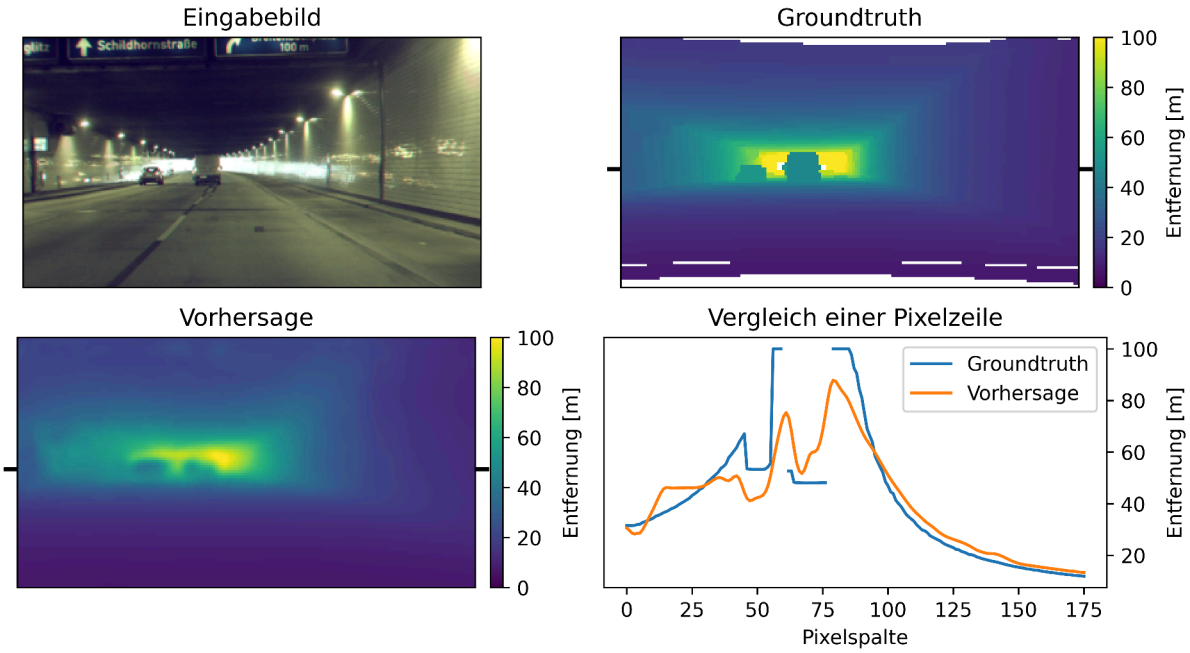


Abbildung 29: Beispiel einer Vorhersage des „ResNet UNet Pretrain“ Modells mit einem REL von 0,085.

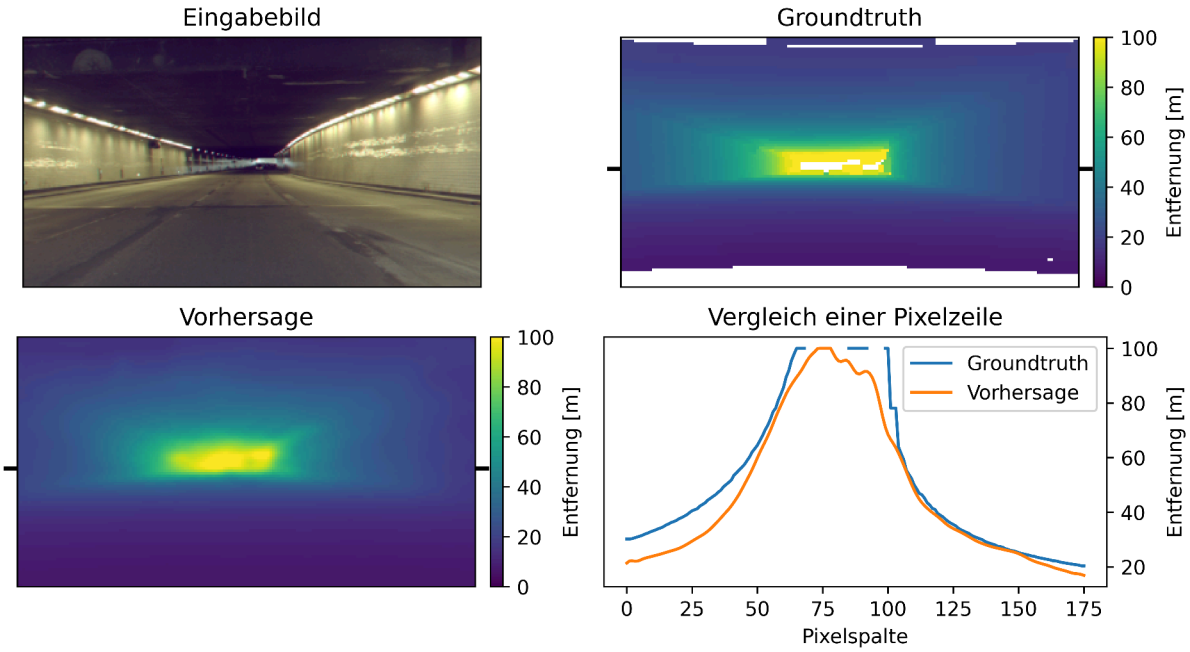


Abbildung 30: Beispiel einer Vorhersage des „ResNet UNet Pretrain“ Modells mit einem REL von 0,093.

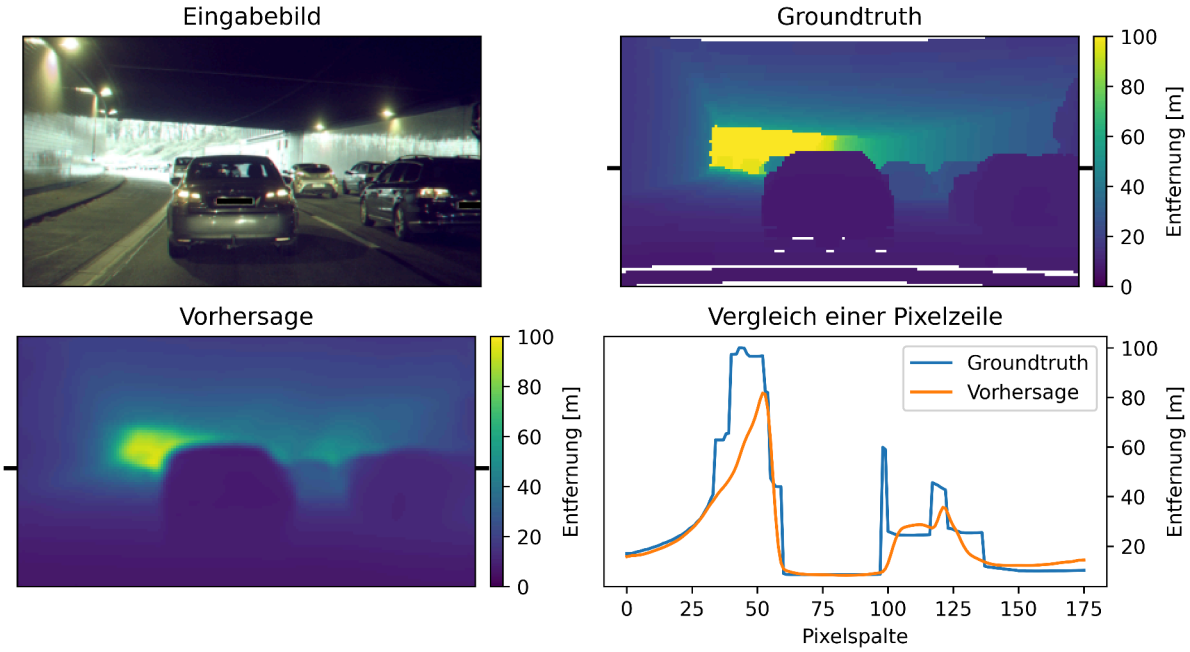


Abbildung 31: Beispiel einer Vorhersage des „ResNet UNet Pretrain“ Modells mit einem REL von 0,098.

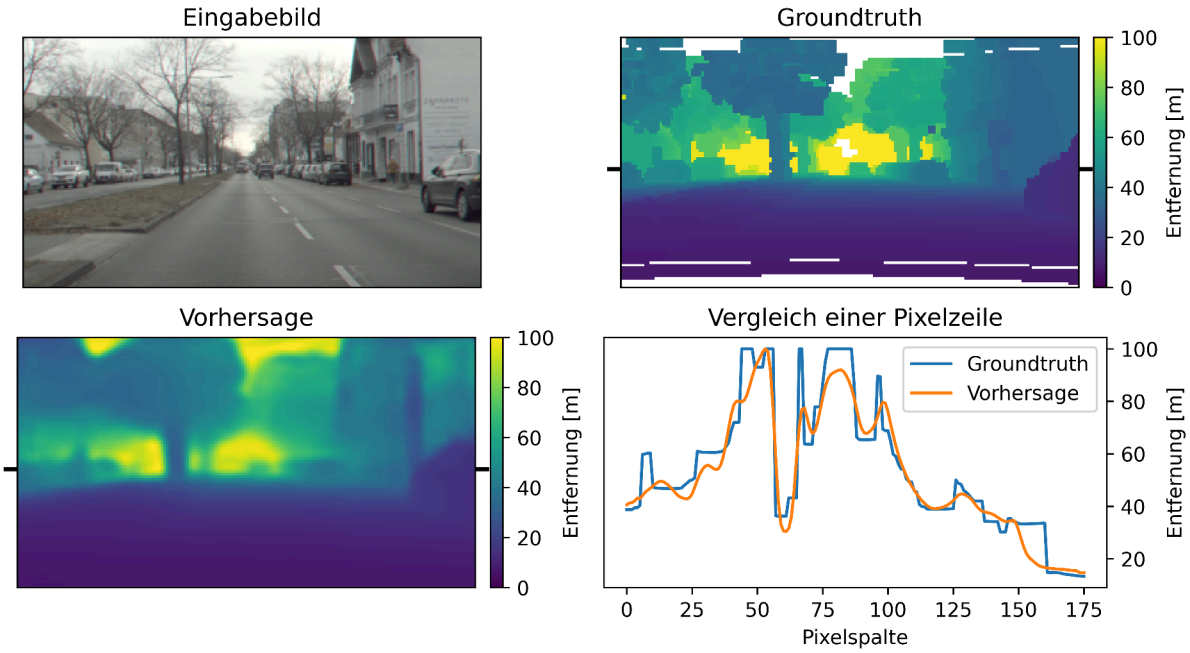


Abbildung 32: Beispiel einer Vorhersage des „ResNet UNet Pretrain“ Modells mit einem REL von 0,106.

Beispiele mit durchschnittlichem REL

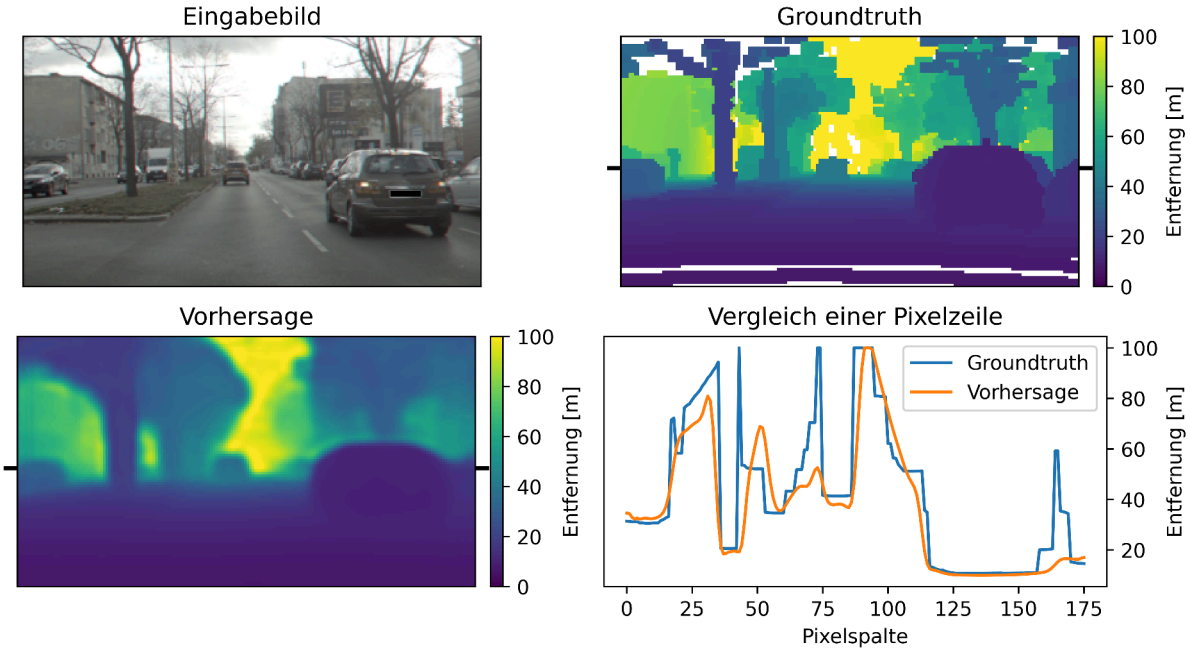


Abbildung 33: Beispiel einer Vorhersage des „ResNet UNet Pretrain“ Modells mit einem REL von 0,145.

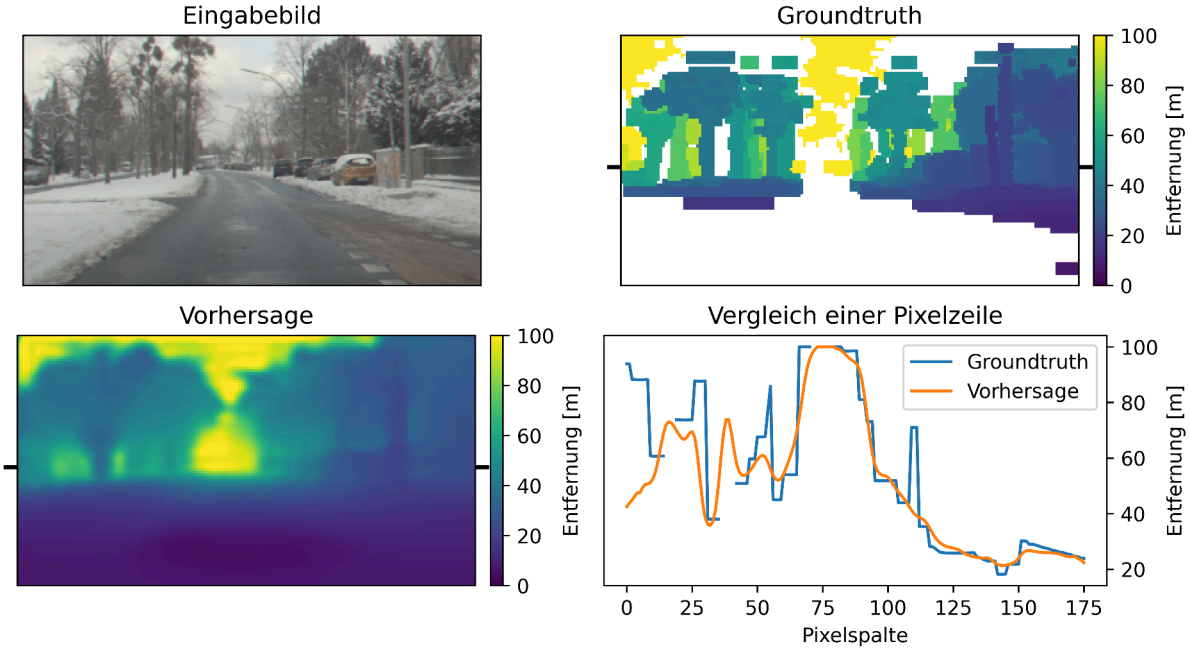


Abbildung 34: Beispiel einer Vorhersage des „ResNet UNet Pretrain“ Modells mit einem REL von 0,157.

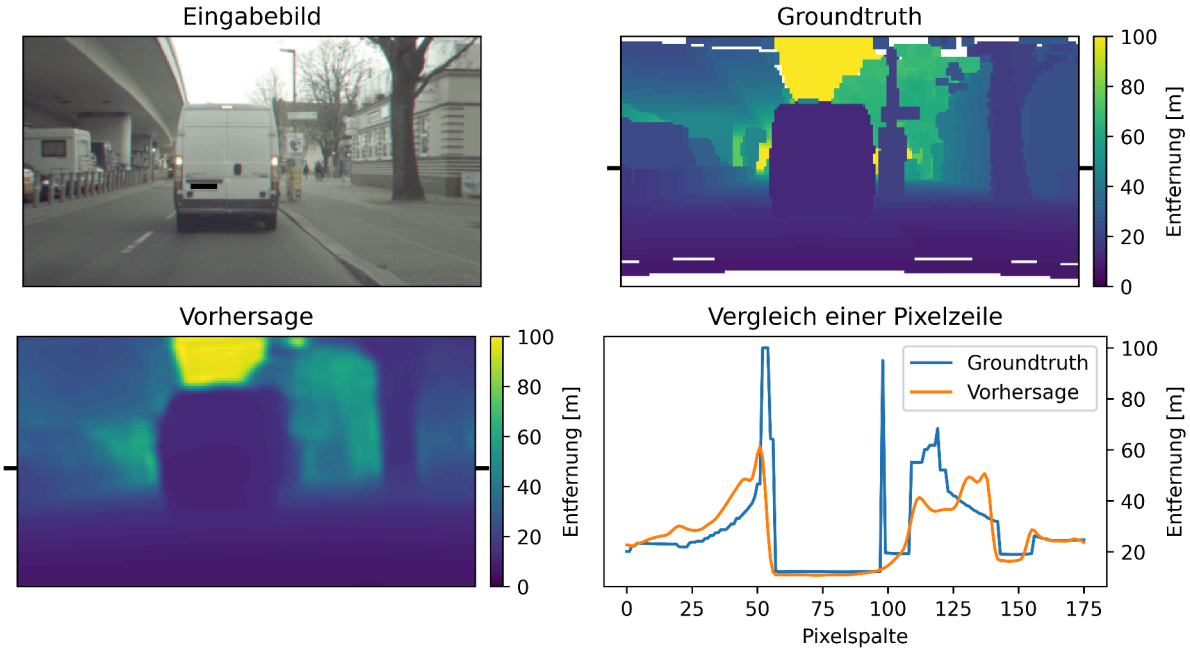


Abbildung 35: Beispiel einer Vorhersage des „ResNet UNet Pretrain“ Modells mit einem REL von 0,163.

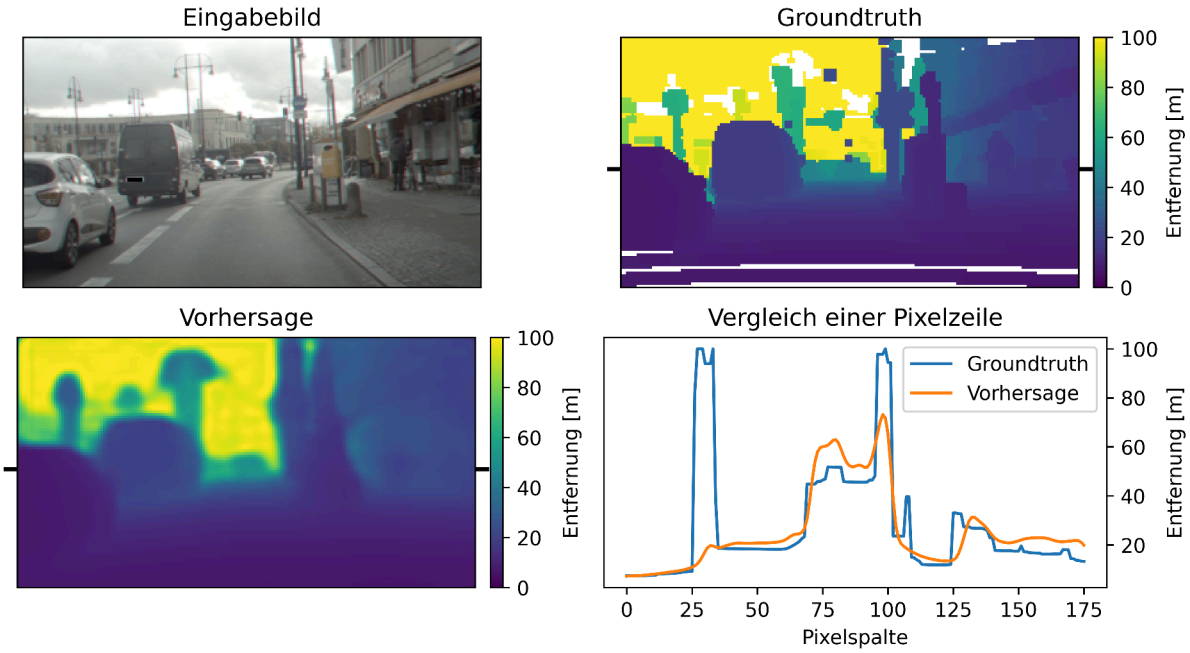


Abbildung 36: Beispiel einer Vorhersage des „ResNet UNet Pretrain“ Modells mit einem REL von 0,157.

Beispiele mit schlechtem REL

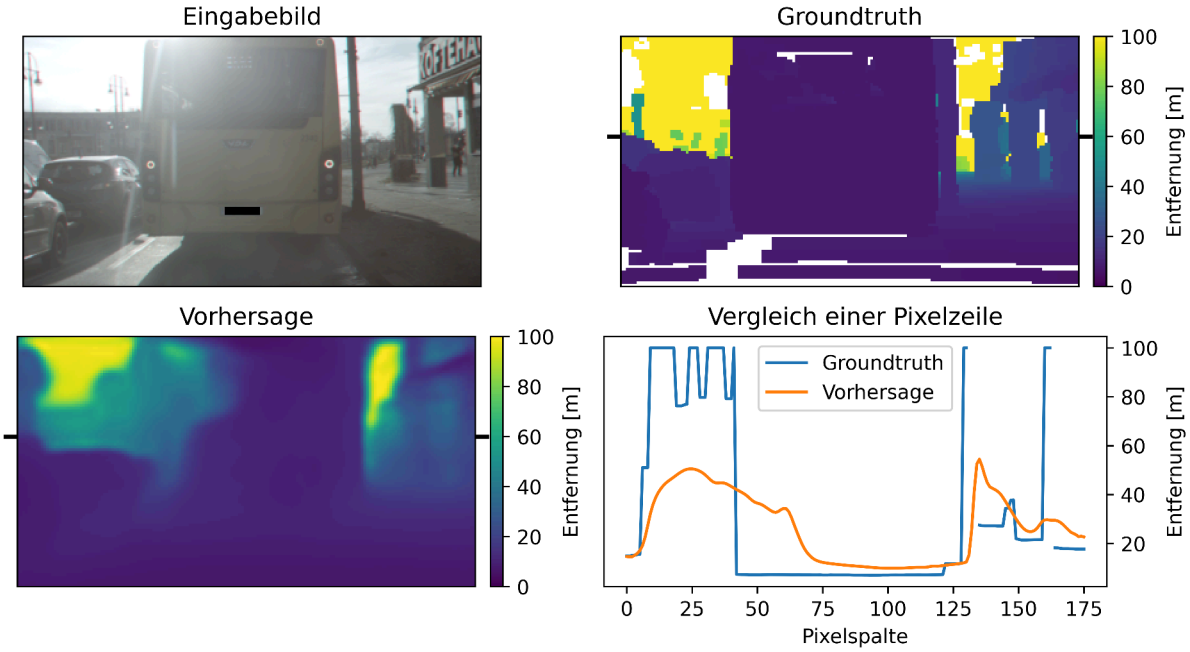


Abbildung 37: Darstellung der sechst-schlechtesten Vorhersage des Modells. Im Kamerabild sind starke Blendenflecke zu erkennen. Der REL beträgt 0,748. Oben links: Eingabebild. Oben rechts: Ground Truth Tiefenbild. Unten links: Vorhersage des Modells. Unten rechts: Vergleich zwischen Ground Truth und Vorhersage in einer Zeile des Bildes. Die Zeile ist in der Ground Truth und Vorhersage markiert.

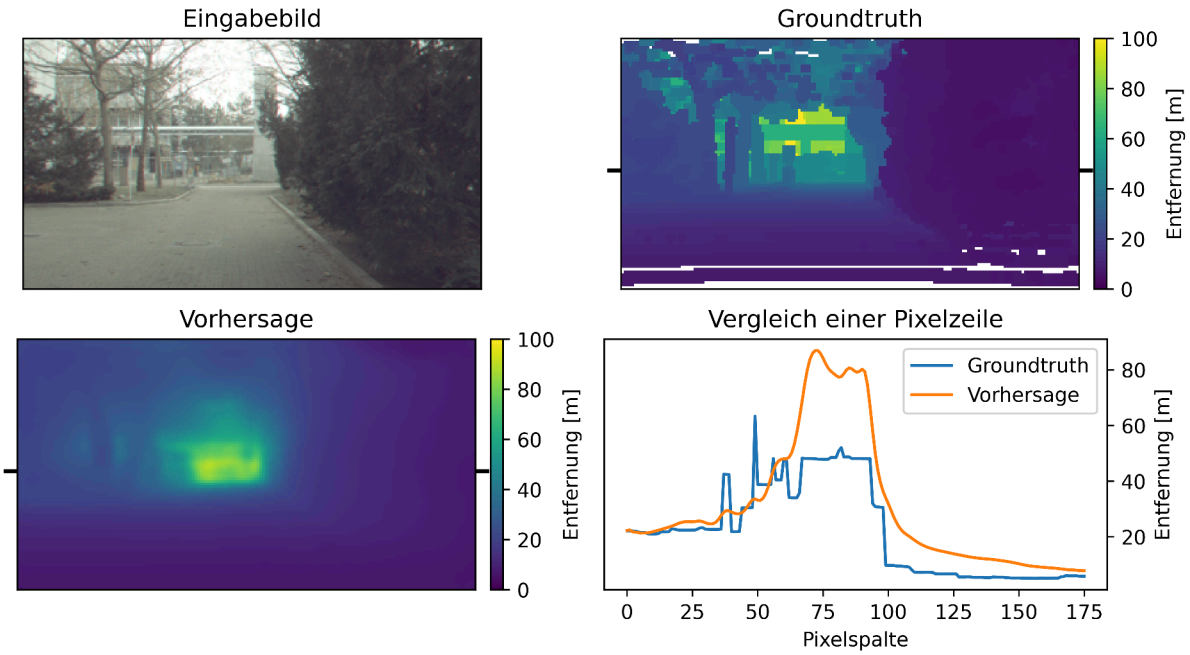


Abbildung 38: Beispiel einer Vorhersage des „ResNet UNet Pretrain“ Modells mit einem REL von 0,459.

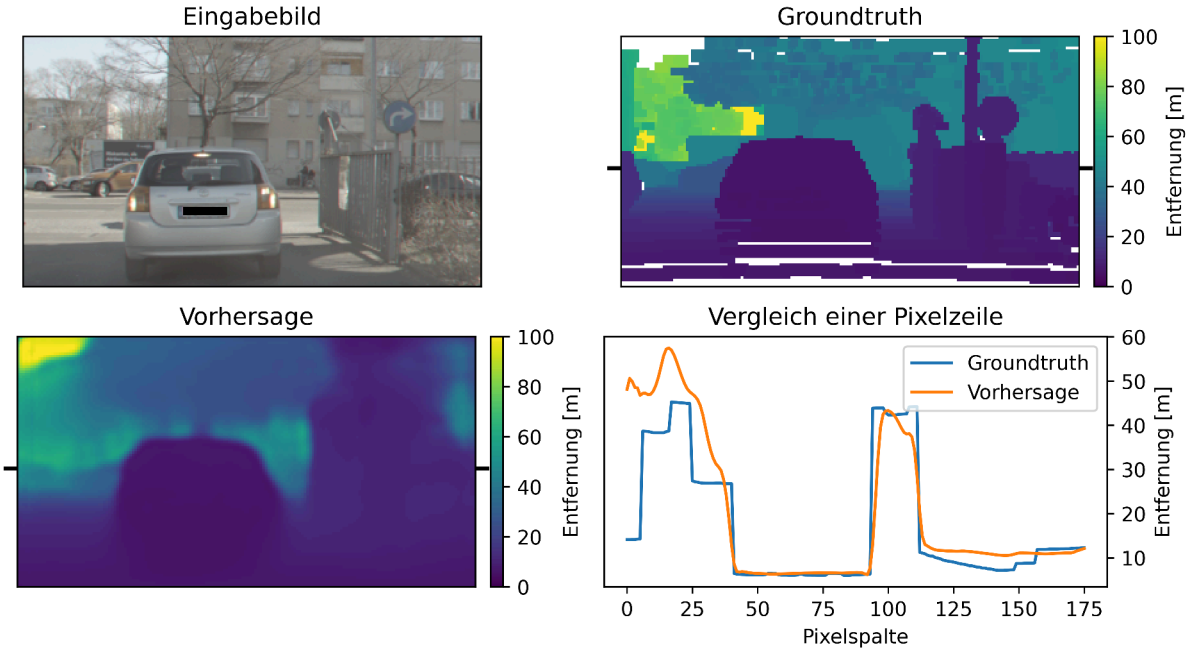


Abbildung 39: Beispiel einer Vorhersage des „ResNet UNet Pretrain“ Modells mit einem REL von 0,281.

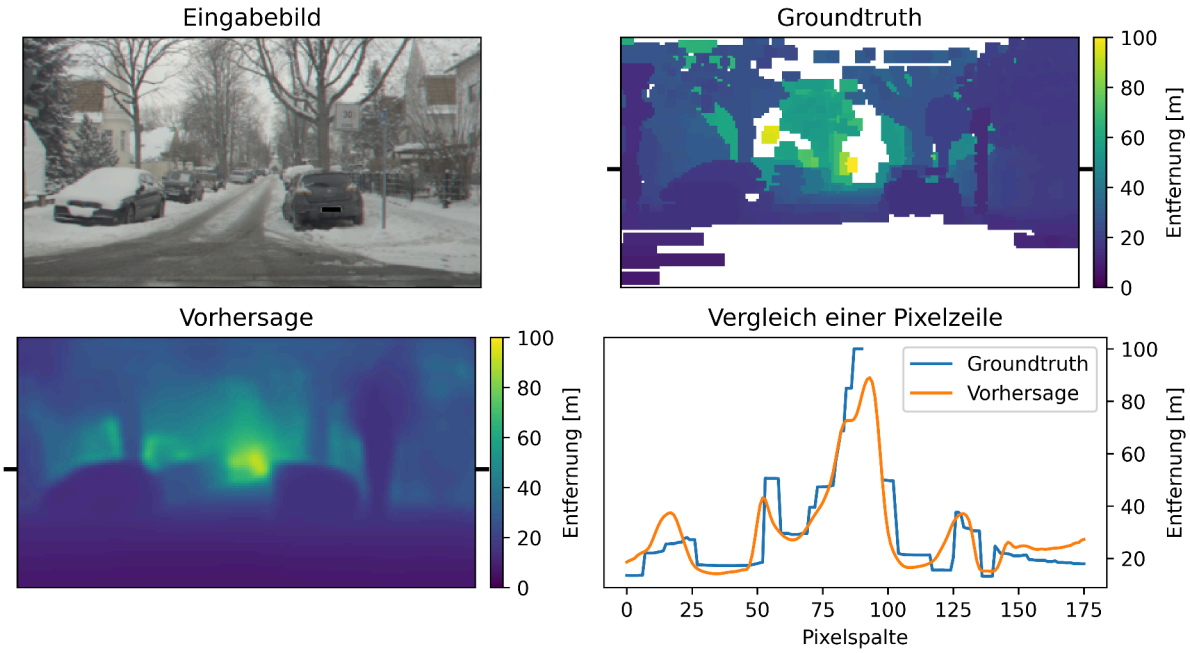


Abbildung 40: Beispiel einer Vorhersage des „ResNet UNet Pretrain“ Modells mit einem REL von 0,241.

Training

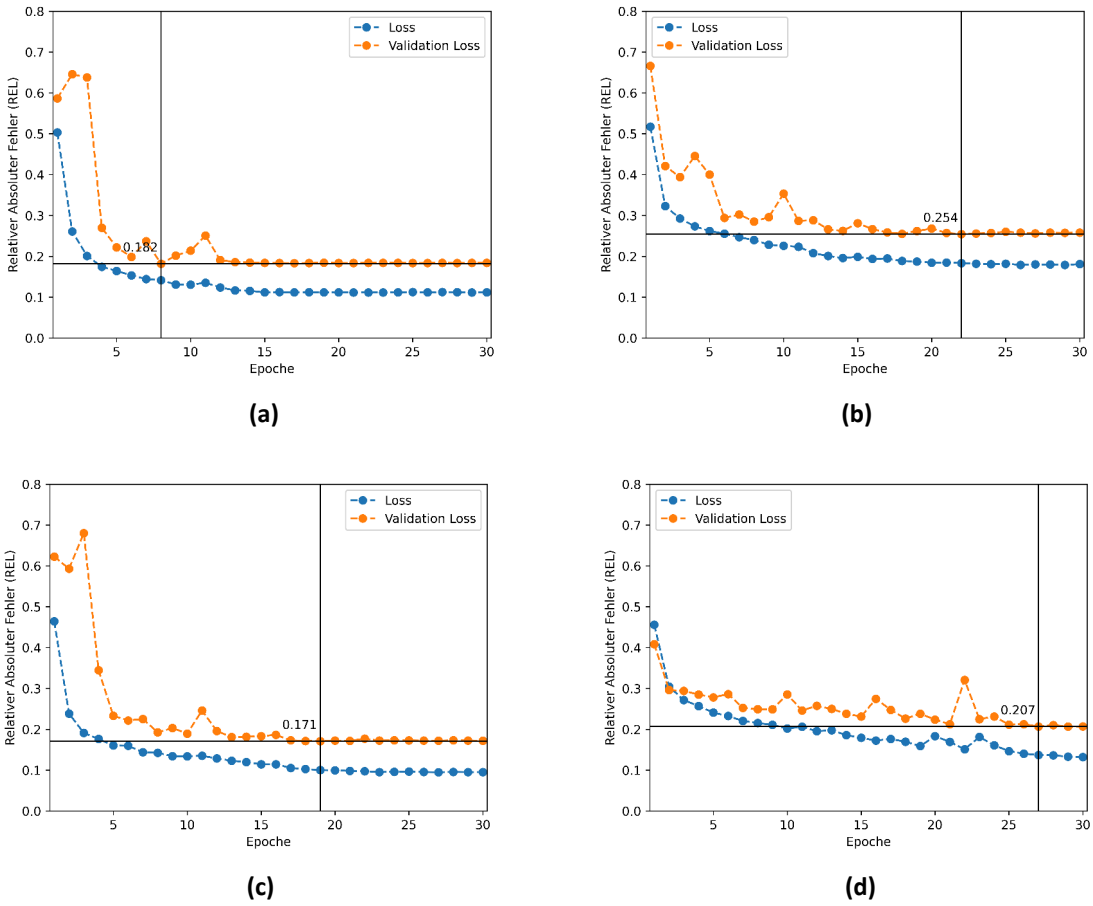


Abbildung 41: Übersicht über das Training der verschiedenen Modelle. **a**: ResNet Pretrain. **b**: ResNet No Pretrain. **c**: ResNet UNet Pretrain. **d**: SENet No Pretrain.