

# IMPLEMENTIERUNG UND ANPASSUNG EINES DEEP NEURAL NETWORKS ZUR BESTIMMUNG VON DISPARITÄTEN AUS STEREOBILDERN MIT TENSORFLOW

## Bachelorarbeit

im Studiengang Medizintechnik  
Hochschule Koblenz, RheinAhrCampus  
Remagen

vorgelegt von

**Mengzhou Sun**  
Matrikelnummer: 532796

**Erstprüfer:** Prof. Dr. Dellen  
**Zweitprüfer:** Prof. Dr. Jaekel

22. April 2019

# Inhaltsverzeichnis

<b>1 Einleitung</b>	<b>3</b>
<b>2 Stereo Vision mit Abstandsmessung</b>	<b>7</b>
2.1 Die Mathematische Prinzip der Stereo Vision . . . . .	9
2.2 Traditionelle Methode: Semi-Global-Matching . . . . .	11
2.2.1 Erklärung der Idee von Semi-Global-Matching . . . . .	11
2.2.2 Erklärung die Formel von Semi-Global-Matching . . . . .	11
<b>3 Deep Learning Methode für Stereo Vision</b>	<b>14</b>
3.1 Erklärung die Rolle von CNN und Transposed-CNN beim Deep-Learning . . . . .	15
3.2 Die Ideen und Struktur von GC-Net . . . . .	17
3.2.1 Implementierung mit Tensorflow . . . . .	19
3.2.2 Die Änderung von GC-Net und eigenes Netzwerk . . . . .	23
<b>4 Trainingsergebnisse und Testergebnisse der Klassifizierungsmethode</b>	<b>25</b>
4.1 Der Loss-Wert und Accuracy der Klassifizierungsmethode . . . . .	25
4.2 Der Vergleich zwischen Regressionsmethode und Klassifizierungsmethode . . . . .	29
<b>5 Zusammenfassung und Problem</b>	<b>29</b>
<b>6 Anhang</b>	<b>30</b>
<b>Literaturverzeichnis</b>	<b>41</b>

## Abbildungsverzeichnis

1	Der Aufbau künstlicher Intelligenz[5] . . . . .	3
2	einfache Hardwarekomposition der binokularen Stereo Vision[7] . . . . .	5
3	linkes Original Bild[11] . . . . .	7
4	rechtes Original Bild [11] . . . . .	7
5	linke Disparitätskarte [11] . . . . .	8
6	rechte Disparitätskarte[11] . . . . .	8
7	Modell der Abstandsmessung[12] . . . . .	9
8	die verschiedene Tiefe Punkte [13] . . . . .	10
9	Beachte die Richtung auf MGM und SGM [18] . . . . .	12
10	Die ReLU-Funktion und ihre Ableitungen [45] . . . . .	15
11	die Struktur von CNN[23] . . . . .	15
12	Die Struktur von transponierten CNN[23] . . . . .	16
13	die Struktur von GC-Net[24] . . . . .	18
14	Der Effekt von GC-Nets Prediction[24] . . . . .	18
15	Linkes Original Bild[26] . . . . .	19
16	Rechtes Original Bild[26] . . . . .	20
17	Linkes Disparitäts-Bild[26] . . . . .	20
18	Rechtes Disparitäts-Bild[26] . . . . .	20
19	Genauigkeit verschiedener Algorithmen bei unterschiedlichen Fehlertoleranzen im Jahr 2012 [29] . . . . .	23
20	die Struktur von die Geändertes GC-Netzwerk[32] . . . . .	24
21	Änderung des Train-Set-Loss-Werts . . . . .	25
22	Änderung von disp_accuracy . . . . .	25
23	Experimentelle Daten im Papier[39] . . . . .	26
24	Erstes Beispiel der Disparitätsvorhersagen . . . . .	27
25	Zweites Beispiel der Disparitätsvorhersagen . . . . .	27

# 1 Einleitung

In den letzten Jahren ist AI (Artificial Intelligence) in der wissenschaftlichen Gesellschaft eine der beliebtesten Forschungsrichtungen und ein Hotspot der Forschung. AI hat zwei großen Richtungen: NLP (Neutral Language Processing) und CP (Computer Vision). Diese Arbeit handelt von Stereo Vision, einem Zweig der Computer Vision. Stereo Vision hat viele Anwendungen in der Industrie. Zum Beispiel Robotersteuerung, Autonomes Fahren und 3D-Rekonstruktion. Dieses Programm basiert auf einem Python-Compiler, um den Abstand zwischen einem Objekt im Bild und dem Kamerasystem zu berechnen. Diese Methode ist in zwei Papieren zu finden: ['Computing the Stereo Matchings Cost with a Convolutional Neural Network'] [1] und ['End-to-End Learning of Geometry and Context for Deep Stereo Regression'] [2]. Diese Methode stammt aus GC-Net. Dieses Neural-Net hat zwei Inputs (linkes Bild und rechtes Bild). Der Output ist ein Disparitäts-Bild. Abbildung 1 zeigt den Aufbau künstlicher Intelligenz (AI). Wie die Abbildung zeigt, ist das untere Bild das Tiefen-Bild. Das ist auch das Ergebnis, dass das Programm letztendlich erhalten sollte. Man kann aus dem Original-Bild mit Hilfe mathematischer Formeln die Disparitäts-Karte bekommen [3, 4]. Darauf wird im zweiten Kapitel eingegangen. Letzteres besteht aus zwei Teilen. Der erste Teil ist der mathematische Zusammenhang zwischen Disparitäts-Bild und Originalbild. Der zweite Teil behandelt die traditionelle Methode (Semi-Global-Matching) für Stereo Vision. Das dritte Kapitel hat drei Teile. Der erste Teil wird die Ideen von PWC-Net und GC-Net für die Stereo Vision einfach erklären. Der zweite Teil ist der Aufbau der CNN (Convolutional Neural Network) und wie dieses Network funktioniert. Der dritte Teil zeigt das Ergebnis. Das beinhaltet Run-Zeit, Loss-Wert, Accuracy und LR (learning rate). Ich verwende die Kerasgerüststruktur unter Tensorflow und in Kapitel 2 werde ich die Verwendung von Keras und Tensorflow separat einführen. Durch den Vergleich mit den Ergebnissen anderer Experimente kann man durch die Änderung der Networksparameter die Programms-Accuracy steigern. Im Anhang befindet sich außerdem mein Python-Code für Stereo-Vision. Darin gibt es noch Anmerkungen zum Code. Am Ende ist das Literaturverzeichnis.

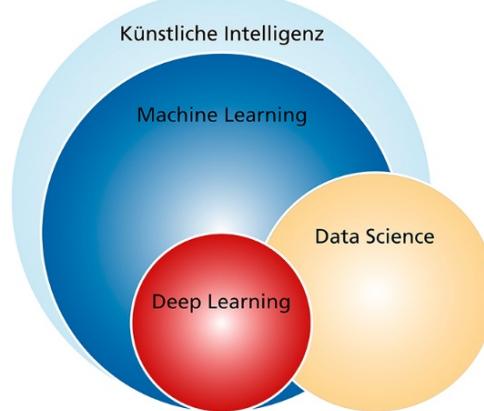


Abbildung 1: Der Aufbau künstlicher Intelligenz[5]

Stellen Sie sich die ganze Struktur der Bachelorarbeit vor und stellen Sie sich auch die Entwicklung und Bedeutung der Stereo Vision in den letzten Jahren vor. Stereo Vision kombiniert die von den beiden Augen erhaltenen Bilder und betrachtet den Unterschied zwischen ihnen, so dass wir ein klares Tiefengefühl erhalten, die Übereinstimmung zwischen den Merkmalen feststellen und die Punkte desselben räumlichen physischen Punkts in verschiedenen Bildern abbilden können. Diesen Unterschied nennen wir das Disparitäts-Bild.

Das binokulare Stereo-Vision-Messverfahren hat viele Vorteile [42], zum Beispiel: eine hohe Effizienz, eine angemessene Präzision, eine einfache Systemstruktur, niedrige Kosten und es ist sehr gut geeignet für die Online-Produkterkennung und Qualitätskontrolle im Fertigungsbereich. Bei der Messung von sich bewegenden Objekten (einschließlich Tieren und menschlichen Körpern) ist die stereoskopische Methode eine effektivere Messmethode, da die Bildaufnahme augenblicklich erfolgt. Das binokulare Stereo-Bildverarbeitungssystem stellt eine der Schlüsseltechnologien des Computer Vision dar. Die Entfernungsinformation von räumlichen 3D-Szenen ist auch der grundlegendste Inhalt in der Computer

Vision-Forschung.

Die bahnbrechende Arbeit im Bereich der binokularen Stereo Vision begann Mitte der 1960er[43] Jahre. Roberts vom MIT extrahierte die dreidimensionale Struktur einfacher regulärer Polyeder wie Würfel, Keile und Prismen aus digitalen Bildern. Es beschrieb die Form und räumliche Beziehung von Objekten und erweiterte die einfache zweidimensionale Bildanalyse in der Vergangenheit auf komplexe dreidimensionale Situation. Es war die Geburtsstunde der Stereo-Vision-Technologie. Mit der Vertiefung der Forschung reicht der Forschungsbereich von der Extraktion von Merkmalen wie Kanten und Ecken, der Analyse geometrischer Elemente wie Linien, Ebenen und Flächen bis zur Analyse von Bildhelligkeit, Textur, Bewegung und Abbildungsgeometrie, sowie der Etablierung verschiedener Datenstrukturen und Inferenzregeln. In den frühen achtziger Jahren fasste Marr die Forschungsergebnisse der Bildverarbeitung, Psychophysik, Neurophysiologie und klinischen Psychiatrie aus Sicht der Informationsverarbeitung zusammen und schuf den theoretischen Rahmen des Visual Computing. Diese grundlegende Theorie hat die Entwicklung der stereoskopischen Sichttechnologie stark vorangetrieben: In diesem Bereich wurde ein komplettes System von der Bilderfassung bis zur endgültigen visuellen Oberflächenrekonstruktion von 3D-Szenen gebildet. Das ist ein sehr wichtiger Zweig.

Nach Jahrzehntelanger Entwicklung hat sich die Stereo-Vision-Technik in den Bereichen Robotik-Vision, Luftbild-Mapping, Reverse Engineering, militärische Anwendung, medizinische Bildgebung und industrielle Inspektion immer weiter verbreitet. Abbildung 2 zeigt eine einfache Hardwarekomposition der binokularen Stereo Vision.

Die aktuelle Stereo Vision muss viele technische Probleme lösen. Derzeit ist die internationale Forschungsrichtung wie im Folgenden gezeigt [6]:

1. Durch die Erstellung eines effektiveren binokularen Stereovisionsmodells können die wesentlichen Eigenschaften der Stereovision ohne Bestimmtheit vollständiger wiedergegeben, mehr Informationen für den Abgleich bereitgestellt und die Schwierigkeit des Stereomatching verringert werden.
2. Forschung der neuen rechentheoretischen Theorie und übereinstimmender Auswahlkriterien und Algorithmusstrukturen für umfassende Stereovision. Um das Problem des Vorhandenseins von Graustufenverzerrung, geometrischer Verzerrung (Perspektive, Drehung, Skalierung usw.), Rauschstörungen oder spezieller Struktur (flacher Bereich) zu lösen, wiederholt man ähnliche Strukturen usw. und deckt das Übereinstimmungsproblem der Szene ab.
3. Der Algorithmus wurde für die Parallelisierung weiterentwickelt, er wurde beschleunigt, der Rechenaufwand reduziert und die Durchführbarkeit des Systems verbessert.
4. Der Algorithmus kann an die Anforderungen der externen Umgebung geschwächt werden, und das System kann für unterschiedliche Anforderungen erstellt werden.

Unter den vielen Faktoren, von der die universelle Entwicklung der Bildverarbeitung abhängt, gibt es sowohl technische als auch kommerzielle Aspekte, aber die Nachfrage in der Fertigung ist letztlich entscheidend. Die Entwicklung des verarbeitenden Gewerbes hat zu einer steigenden Nachfrage nach Machine Vision geführt und erreicht, dass Machine Vision in der Vergangenheit gesammelt, analysiert, übertragen und beurteilt wird und allmählich in eine offene Richtung geht. Dieser Trend ist für die Industrie sehr wichtig, weil es die Stereo Vision bringt, um die Weitere Innovationen auf dem Gebiet der Technologie. Die Nachfrage bestimmt das Produkt, und nur das Produkt, das die Nachfrage erfüllt, überlebt. Dies gilt auch für die Bildverarbeitung.

Binokulare Stereo-Vision wird derzeit in vier Bereichen eingesetzt: Roboternavigation, Erkennung von Parametern für Mikrobetriebssysteme, 3D-Messung und virtuelle Realität. Japan ist in dieser Hinsicht das führende Land. Das Adaptive Mechanical Systems Research Institute der Osaka University of Japan hat ein adaptives binokulares Vision-Servosystem entwickelt, das das Prinzip der binokularen Stereo-Vision verwendet, z. B. die relativ statischen drei Marker in jedem Bild als Referenz, um das Ziellbild in Echtzeit zu berechnen. Das kurze Feld wird verwendet, um die nächste Bewegungsrichtung des Ziels vorherzusagen, und eine adaptive Verfolgung des Ziels mit einem unbekannten Bewegungsmodus wird realisiert. Das System erfordert nur eine stationäre Referenzmarke in beiden Bildern, es sind keine Kameralparameter erforderlich. Das traditionelle Servosystem für die visuelle Nachführung muss im Voraus



Abbildung 2: einfache Hardwarekomposition der binokularen Stereo Vision[7]

die Bewegungsparameter der Kamera, die Optik und die Bewegung des Ziels kennen.

Am stärksten umstritten war die Verwendung der binokularen Stereo-Vision zur Entfernungsmessung im autonomen Bereich anstelle der LiDAR-Messung. Zuallererst sollten die zwei Techniken Stereo-Vision und LiDAR sich nicht widersprechen. Aber beide haben ihre eigenen Stärken und Schwächen. In einfachen Worten ausgedrückt: LiDAR hat eine hohe Präzision, aber die Installation ist komplex und es ist teuer. Die Schwierigkeit besteht darin, dass es keine visuellen Funktionen gibt, die Aufgaben wie Nachverfolgung und Erkennung einfacher machen. Der Vorteil des Stereo-Vision ist, dass es über visuelle Funktionen verfügt, die eine Kombination mit anderen Bildverarbeitungsalgorithmen ermöglichen. Die Genauigkeit ist jedoch geringer als bei LiDAR (kann jedoch die Anforderungen des automatischen Fahrens erfüllen) und die Hardwarestabilität ist hoch (um die relative Stabilität zwischen den beiden Kameras sicherzustellen). Dies kann das größte Hindernis für die tatsächliche Anwendung von binokulärer Sicht sein.

Viele der Unternehmen, die autonomes Fahren entwickeln, verwenden drei Komponenten: Laser-Radar, Millimeterwellen-Radar und differentielles GPS. Der Gesamtpreis dieser drei Hauptkomponenten liegt bei über 100.000 Euro. Das Problem ist, dass es unrealistisch ist, sich auf diesen Preis zu verlassen, um ein selbstfahrendes Auto herzustellen: Das Millimeterwellenradar wurde in den letzten Jahren im Preis reduziert, und die Massenproduktion für automatisches Cruisen und Antikollision kann auf unter 2.000 Euro gesenkt werden. Im Prinzip kann differentielles GPS mit einem gewöhnlichen GPS hergestellt werden. Das Laserradar ist jedoch immer der Preis, der nicht verringert werden kann und hat sich zu einer wichtigen Preisschwelle für selbstfahrende Autos entwickelt. Betrachten wir verschiedene Vision Ansätze: Beim Autofahren ist nicht unbedingt eine binokulare Sicht erforderlich. Israels Mobileye Systeme basieren auf einer einzigen Kamera und das maschinelle Lernen, um die Hindernisentfernung zu errechnen, kann als Structure From Motion kombiniert mit maschinellem Lernen betrachtet werden. Binokulare Sicht hat auch eine Reihe von Schwierigkeiten bei der Anwendung, aber im Allgemeinen kann man sagen, dass Entfernung, Erkennungsgenauigkeit, Abtastfrequenz, Spiegel Loch, Lebensdauer und Preis die sechs wichtigen Aspekte für den Sieg über das Laserradar sind. Bei einigen Autoherstellern erreicht die USB 3.0-Industriekamera eine Auflösung von 1080p oder sogar 4k, und 120fps sind nicht schwierig, was einige tausend Dollar ausmacht. Sie kann im Auto installiert werden, muss nicht wie ein Laserradar auf dem Fahrzeug aufgesetzt werden, es gibt keine beweglichen Teile. Selbst wenn zwei Module verwendet werden, ist es nicht teuer. Die zwei wichtigsten Punkte hier sind, dass die Kamera eine große Anzahl von Möglichkeiten bietet und dass die Kamera billig genug ist. Dies macht kamerabasierte Computer

Vision in autonomen Fahrzeugen zu einer vielversprechenden Lösung. Die Schwierigkeit des binokularen Stereo-Vision liegt in der Berechnung des Abgleichs. In Unternehmen, die binokulare Stereo-Vision verwenden, ist es daher häufig erforderlich, eine leistungsstarke binokulare Synthese zu entwickeln. Der Bereich, für den der Algorithmus optimiert werden kann, ist bereits sehr klein, weshalb er aus Sicht von GPU, FPGA und ASIC der Fokus der binokularen Synthese ist. [8-10]

## 2 Stereo Vision mit Abstandsmessung

Stereo Vision bedeutet mit Hilfe von zwei Original-Bilden (links und rechts) eine Disparitäts-Bild zu bekommen. Das Disparitäts-Bild ist ein Matrix mit Zahlen von 0 bis 255. Man kann durch die Pixel den Abstand zwischen Kamerasystem und dem Objekt im Bild berechnen. Eigentlich kann man auch mit vielen anderen Methoden den Abstand messen werden, z.B.: Laser-Messung und Mikrowellenmessung. Aber mit diesen Methoden gibt es noch viele Probleme. Beispielsweise sind sie teuer und der Messbereich ist klein. Mit Hilfe der Original- und Disparitäts-Bilder können diese Probleme einfach und gut gelöst werden.



Abbildung 3: linkes Original Bild [11]



Abbildung 4: rechtes Original Bild [11]



Abbildung 5: linke Disparitätskarte [11]



Abbildung 6: rechte Disparitätskarte[11]

Die oberen Bilder sind das Original Bild und das Disparität-Bild. Im Folgenden wird die mathematische Beziehung zwischen Tiefenkarte und Disparitätskarte erklärt. Ein Stereo-Bild bezieht sich auf eine signifikante Pixeldifferenz oder Bewegung zwischen einem Paar stereoskopischer Bilder. Um dies zu erfahren,

versuchen Sie, eines Ihrer Augen zuzumachen, dann schnell wieder aufzumachen und das andere zuzumachen. Objekte, die sich in Ihrer Nähe befinden, scheinen einen erheblichen Sprung zu machen, während Objekte, die weiter entfernt sind, sich nur wenig bewegen. Diese Bewegung ist eine Lücke. In der 3D-Computergrafik ist das ein Disparitäts-Bild oder ein Bildkanal, der Informationen über die Entfernung der Oberfläche des Szenenobjekts vom Ansichtspunkt enthält. Es ist einem Graustufenbild ähnlich, mit der Ausnahme, dass jeder Pixelwert die tatsächliche Entfernung des Sensors vom Objekt ist. Normalerweise werden RGB-Bilder und Tiefenbilder registriert, daher besteht eine Eins-zu-Eins-Entsprechung zwischen Pixeln. Durch die Berechnung der Parallaxe der beiden Bilder wird die Abstandsmessung der vorderen Szene (der vom Bild erfasste Bereich) direkt vorgenommen, ohne zu bestimmen, welcher Art von Hindernis vorne liegt. Daher kann für jede Art von Hindernis die notwendige Warnung oder Bremsung entsprechend der Änderung der Abstandsinformation durchgeführt werden. Das Prinzip der binokularen Kamera ist dem des menschlichen Auges ähnlich. Das menschliche Auge kann die Entfernung des Objekts wahrnehmen, da die beiden Augen einen Unterschied in dem Bild darstellen, das von demselben Objekt dargestellt wird, auch Parallaxe" genannt. Je weiter weg das Objekt ist, desto kleiner ist die Parallaxe, je näher, desto größer ist die Parallaxe. Die Größe der Parallaxe entspricht der Entfernung zwischen Objekt und Auge, weshalb 3D-Filme dazu führen können, dass Menschen eine stereoskopische Wahrnehmung haben.

## 2.1 Die Mathematische Prinzip der Stereo Vision

Zuerst sollte man durch die Abbildung 7 das ganze mathematische Modell der Stereo Vision kennenlernen.

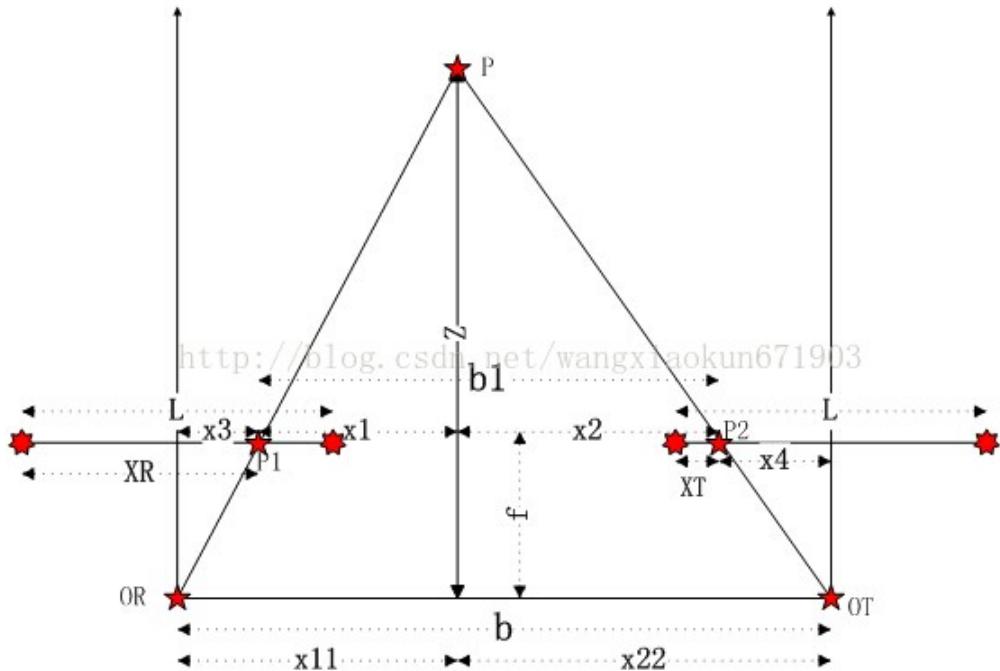


Abbildung 7: Modell der Abstandsmessung[12]

P ist der Punkt im Raum, P1 und P2 sind die Abbildungspunkte von Punkt P auf der linken und rechten Bildebene, f ist die Brennweite, OR und OT sind die optischen Mittelpunkte der linken und rechten Kamera. Wie in der oberen Abbildung gezeigt, sind die optischen Achsen der linken und rechten Kamera parallel.  $X_R$  und  $X_T$  sind die Abstände zwischen den beiden Bildpunkten auf der linken und rechten Bildebene vom linken Bildrand. Nun wird die mathematische Beziehung zwischen Parallaxe und Tiefe gezeigt.

$$\frac{b}{z} = \frac{(X_T + b) - X_R}{z - f} \quad (1)$$

Kann abgeleitet werden:

$$Z = \frac{b * f}{X_T - X_R} = \frac{b * f}{d} \quad (2)$$

Ableitungsprozess:  $x_1 + x_2 = b_1$ ,  $x_{11} + x_{22} = b$  Nach dem Prinzip ähnlicher Dreiecke:

$$\frac{X_{11}}{Z} = \frac{Z_1}{Z - f} \quad (3)$$

$$\frac{X_{22}}{Z} = \frac{Z_2}{Z - f} \quad (4)$$

Kann abgeleitet werden:

$$\frac{X_{22} + X_{11}}{Z} = \frac{b}{z} = \frac{Z_1 + Z_2}{Z - f} = \frac{b_1}{z - f} \quad (5)$$

$$b_1 = b - X_3 - X_4 = b - (X_R - \frac{L}{2}) - (\frac{L}{2} - X_T) = b - X_R + X_T = (b + X_T) - X_R \quad (6)$$

Wenn wir die Formel (5) mit Formel (6) verbinden, bekommen wir die Formel

$$\frac{b}{z} = \frac{(b + X_T) - X_R}{Z - f} \quad (7)$$

Nach dieser Formel kann man die Tiefe durch die Brennweite und den Abstand berechnen. Die untere Abbildung verdeutlicht: Je näher der Abstand von der Bildoberfläche ist, desto größer ist die Parallaxe in der linken und der rechten Kamera und je weiter weg von der Bildoberfläche, desto geringer ist die Parallaxe in der linken und der rechten Kamera. Die Position des P-Punkts in der folgenden Abbildung ist das, was wir berechnen müssen.

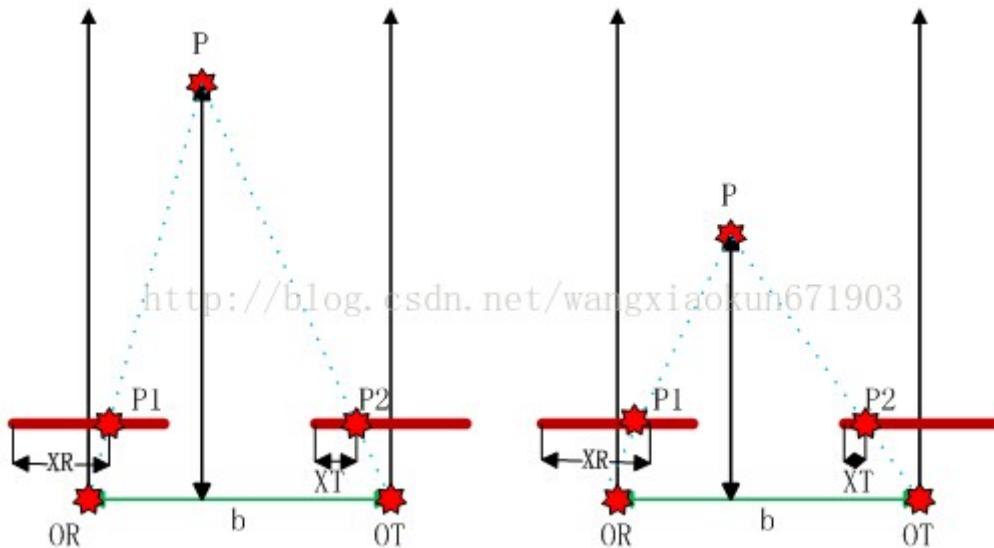


Abbildung 8: die verschiedene Tiefe Punkte [13]

Nach der Formel  $z = (b * f) / d$  kann man den Abstand (z) ausrechnen, b ist der Abstand zwischen der linken und der rechten Kamera. f ist die Brennweite. d ist Parallaxe zweier entsprechender Punkte. Im

Dreidimensionalen muss man die Punkteposition bestimmen. Wir können die dreidimensionalen Koordinaten der Punkte mit Hilfe der mathematischen Formel im unteren Bild bestimmen.

$$\begin{cases} X_c = \frac{B*X_{left}}{Disparity} \\ Y_c = \frac{B*Y}{Disparity} \\ Z_c = \frac{B*f}{Disparity} \end{cases} \quad (8)$$

Im nächsten Teil wird gezeigt, wie man mit Hilfe der traditionellen Methode die Parallaxe von zwei entsprechende Punkten berechnen kann.

## 2.2 Traditionelle Methode: Semi-Global-Matching

### 2.2.1 Erklärung der Idee von Semi-Global-Matching

Semi-Global-Matching (abgekürzt SGM) ist ein Semi-Global-Matching-Algorithmus, der zur Berechnung von Disparitäten in der binokularen Sicht verwendet wird. Die Implementierung in OpenCV ist semi-globales Block-Matching (SGBM).

Die Idee von SGBM ist wie unten gezeigt:

Durch Auswählen der Ungleichheit jedes Pixels zur Bildung einer Ungleichheitskarte und Einstellen einer globalen Energiefunktion in Bezug auf die Ungleichheitskarte wird die Energiefunktion minimiert, um die optimale Ungleichheit für jedes Pixel zu erreichen.

Die Energiefunktion ist wie folgt:

$$E(D) = \sum_p \left\{ (C(P, D_p) + \sum_{q \in N_p} (P_1 I)[|D_p - D_q| = 1] + \sum_{q \in N_p} (P_2 I)[|D_p - D_q| > 1]) \right\}$$

D bezieht sich auf die Disparitätskarte. E (D) ist die Energiefunktion, die der Disparitätskarte entspricht.

p, q repräsentiert ein Pixel im Bild.

Np bezieht sich auf das benachbarte Pixel des Pixels p (wird allgemein als verbunden betrachtet).

C (p, Dp) bezieht sich auf die Kosten des Pixels, wenn die aktuelle Pixeldisparität Dp ist.

P1 ist ein Strafkoeffizient, der für diejenigen Pixel gilt, bei denen der Disparity-Wert des benachbarten Pixels des Pixels p um eins von dem Disparity-Wert von p abweicht.

P2 ist ein Strafkoeffizient, der für jene Pixel gilt, bei denen der Disparity-Wert des Pixels p benachbarten Pixels um mehr als eins von dem Disparity-Wert von p abweicht.

Die Funktion I [ ] gibt 1 zurück, wenn das Argument in der Funktion true ist, andernfalls 0.

Die Verwendung der obigen Funktion zum Finden der optimalen Lösung in einem zweidimensionalen Bild ist ein NP-vollständiges Problem, das zu zeitaufwändig ist, sodass das Problem mit mehreren eindimensionalen Problemen, nämlich linearen Problemen, angenähert wird. Und jedes eindimensionale Problem kann mit dynamischer Programmierung gelöst werden. Da 1 Pixel 8 benachbarte Pixel hat, wird es im Allgemeinen in 8 eindimensionale Probleme zerlegt [14-17].

### 2.2.2 Erklärung die Formel von Semi-Global-Matching

Betrachten Sie die Richtung von links nach rechts wie unten gezeigt:

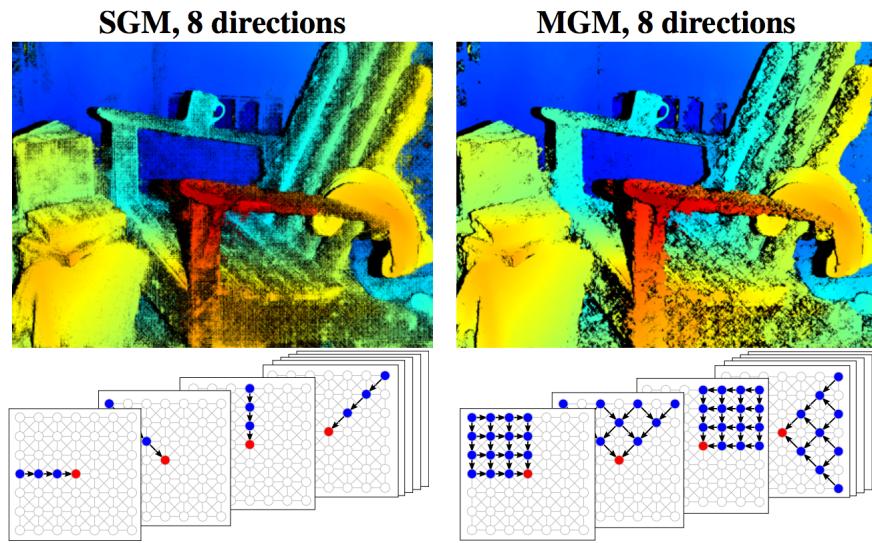


Abbildung 9: Beachte die Richtung auf MGM und SGM [18]

Dann bezieht sich die Ungleichheit jedes Pixels nur auf das Pixel auf der linken Seite und hat die folgende Formel:

$$L_r = C(p, d) + \min(L_r(p - r, d), L_r(p - r, d - 1) + P_1, L_r(p - r, d + 1) + P_2, \min L_r(p - r, k))$$

$r$  bezieht sich auf eine Richtung, die auf das aktuelle Pixel  $p$  zeigt, was hier als benachbartes Pixel links vom Pixel  $p$  verstanden werden kann.

$L_r(p, d)$  stellt den minimalen Kostenwert entlang der aktuellen Richtung (d.h. von links nach rechts) dar, wenn die Ungleichheit des aktuellen Pixels  $p$   $d$  ist.

Dieses Minimum ist das Minimum, das aus den vier möglichen Kandidatenwerten ausgewählt wird:

1. Der kleinste Kostenwert des vorherigen Pixels (linkes benachbartes Pixel), wenn der Wert der Ungleichheit  $d$  ist.
2. Wenn das vorherige Pixel (linkes benachbartes Pixel) einen Wert von  $d-1$  hat, seinen minimalen Kostenwert + Strafkoeffizienten  $P_1$ .

3. Wenn das vorherige Pixel (linkes benachbartes Pixel) einen Wert von  $d + 1$  hat, seinen minimalen Kostenwert + Strafkoeffizienten P1.
4. Die Disparität des vorherigen Pixels (links benachbartes Pixel) nimmt den Wert des anderen, seinen minimalen Kostenwert + den Strafkoeffizienten P2.

MGM ist ein wesentlich mehr globales Matching für Stereovision. Außerdem muss der Kostenwert des aktuellen Pixels  $p$  von den minimalen Kosten abgezogen werden, wenn das vorherige Pixel einen anderen Disparitätswert annimmt. Dies liegt daran, dass  $L_r(p, d)$  mit der Rechtsverschiebung des aktuellen Pixels wächst. Um zu verhindern, dass der Wert überläuft, belassen Sie den Wert auf einem kleinen Wert [19, 20].

### 3 Deep Learning Methode für Stereo Vision

Das Konzept des Deep Learning beruht auf dem Studium künstlicher neuronaler Netzwerke. Ein mehrschichtiges Perzepron mit mehreren verborgenen Schichten ist eine Deep Learning Struktur. Beim Deep Learning werden Funktionen auf niedriger Ebene kombiniert, um abstraktere Attributkategorien oder -merkmale für Repräsentationsdarstellungen zu bilden, um verteilte Funktionsdarstellungen von Daten zu ermitteln. Das Konzept des Deep Learning wurde 2006 von Hinton et al. vorgeschlagen. Basierend auf dem Deep Trusted Network (DBN) wird ein unüberwachtes, greedy Schicht-für-Schicht-Trainingsalgorithmus vorgeschlagen, in der Hoffnung auf die Lösung tiefgreifender strukturbbezogener Optimierungsprobleme zu treffen. Darüber hinaus ist das von Lecun et al. vorgeschlagene neuronale Faltungsnetz der erste echte mehrschichtige Strukturlernalgorithmus, der räumlich relative Beziehungen verwendet, um die Anzahl der Parameter zu reduzieren, um die Trainingsleistung zu verbessern [21, 22].

Deep Learning ist eine Methode, die auf der Darstellung von Daten beim maschinellen Lernen basiert. Beobachtungen (z. B. ein Bild) können auf verschiedene Arten dargestellt werden, beispielsweise als ein Vektor jedes Pixelintensitätswerts oder abstrakter als eine Reihe von Kanten, Regionen einer bestimmten Form und dergleichen. Es ist einfacher, Aufgaben von Instanzen (z. B. Gesichtserkennung oder Gesichtsausdruckerkennung) mithilfe bestimmter Darstellungsmethoden zu erlernen. Der Vorteil des Deep Learning besteht in der Verwendung von nicht überwachten oder halbüberwachten Merkmallernen und hierarchischen Merkmalsextraktionsalgorithmen anstelle von manuellen Erfassungsfunktionen. Das neuronale Netzwerk besteht aus einer großen Anzahl von Neuronen, die miteinander verbunden sind. Nachdem jedes Neuron eine lineare Kombination von Eingaben akzeptiert hat, beginnt es mit einer einfachen linearen Gewichtung und fügt dann jedem Neuron eine nichtlineare Aktivierungsfunktion für die nichtlineare Transformation und Ausgabe hinzu. Die Verbindung zwischen jeweils zwei Neuronen repräsentiert einen gewichteten Wert, der als Gewicht bezeichnet wird. Unterschiedliche Gewichtungen und Aktivierungsfunktionen führen zu unterschiedlichen Ausgaben des neuronalen Netzwerks.

Während des Trainings stehen viele Optimierungsfunktionen zur Auswahl, beispielsweise: RMSprop, Adam, SGD, Adadelta und viele mehr. Da dieses Experiment hauptsächlich RMSprop verwendet, sollten es auf die folgenden Funktionen und Methoden konzentrieren. Der RMSProp-Algorithmus wird als Root-Mean-Square-Prop-Algorithmus bezeichnet. RMSprop ähnelt dem Gradientenabstiegs-Algorithmus mit Impuls. Der RMSprop-Optimierer beschränkt die Schwingungen in vertikaler Richtung. Wir können unsere Learning Rate erhöhen und unser Algorithmus könnte durch größere Schritte in horizontaler Richtung schneller konvergieren. Der Gradientenabstieg beschreibt, wie die Gradienten berechnet werden. Die folgenden Gleichungen zeigen, wie die Gradienten für RMSprop und den Gradientenabstieg mit dem Moment berechnet werden. Der Wert des Momentes wird durch Beta angegeben und ist normalerweise auf 0,9 eingestellt. [44]

Neben den Optimierungsfunktionen sind auch Aktivierungsfunktionen ein wichtiger Bestandteil des neuronalen Netzwerks. Z.B.: Tanh, Siegmoid, ReLU und so weiter. Jeder Neuronenknoten im neuronalen Netzwerk akzeptiert den Ausgabewert der oberen Neuronenschicht als den Eingabewert des Neurons und gibt den Eingabewert an die nächste Schicht weiter. Der Neuronenknoten der Eingabeschicht übergibt den Eingabeattributwert direkt an die nächste Schicht. In einem mehrschichtigen neuronalen Netzwerk besteht ein funktionaler Zusammenhang zwischen dem Ausgang des oberen Knotens und dem Eingang des unteren Knotens. Diese Funktion wird als Aktivierungsfunktion (auch als Erregungsfunktion bezeichnet) bezeichnet. In dem Experiment ist die Hauptfunktion die ReLU-Funktion, daher führt es hauptsächlich deren Funktion und mathematische Bedeutung ein. Die mathematische Formel dazu ist [45]:

$$\text{ReLU} = \max(0, x)$$

Die ReLU-Funktion stellt eigentlich eine Maximum-Funktion dar. Beachten Sie, dass es sich hierbei nicht um eine vollständige Übersicht handelt, wir können jedoch einen Sub-Gradienten verwenden, wie in der Abbildung oben dargestellt. Obwohl einfach, ist ReLU eine wichtige Errungenschaft der letzten Jahre, mit den folgenden Hauptvorteilen:

1. Das Problem des Verschwindens von Gradienten (im positiven Intervall) wurde gelöst.
2. Die Berechnungsgeschwindigkeit ist sehr schnell und muss nur beurteilt werden, ob die Eingabe größer als 0 ist.
3. Die Konvergenz ist viel schneller als Sigmoid und Tanh.

Das Bild der ReLU-Funktion und ihrer Ableitungen ist unten dargestellt:

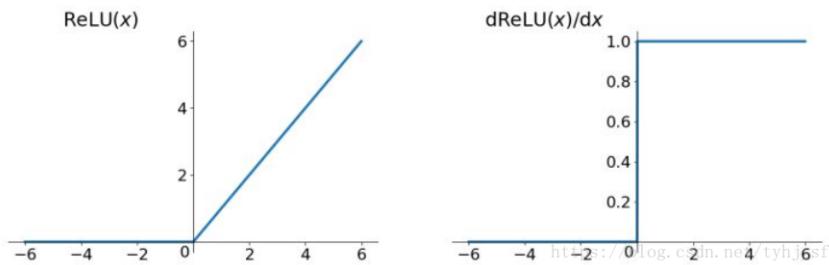


Abbildung 10: Die ReLU-Funktion und ihre Ableitungen [45]

### 3.1 Erklärung die Rolle von CNN und Transposed-CNN beim Deep-Learning

In unserem Projekt verwenden wir hauptsächlich die Faltungsoperation, die auch als Faltungs-Layer bezeichnet wird, die bekannteste CNN. Daher ist es notwendig, die Struktur und Funktion von CNN klar zu erklären. Convolutional Neural Networks (CNN) ist eine Art neuronales Feedforward-Netzwerk mit Faltungsberechnung und Tiefenstruktur, einem der repräsentativen Algorithmen des Tiefenlernens. Da das CNN in der Lage ist, die Shift-Invariante-Klassifikation durchzuführen, wird es auch als "Shift-Invariante Künstliche Neuronale Netz (SIANN)" bezeichnet. Das Studium der Convolutional Neural Network begann in den 1980er und 1990er Jahren. Das Zeitverzögerungsnetzwerk und LeNet-5 waren die ersten neuronalen Faltungsnetzwerke. Nach dem 21. Jahrhundert wurde die Theorie des Deep Learning eingeführt. Mit der Verbesserung der numerischen Computerausrüstung wurden Convolutional Neural Networks rasch entwickelt und in der Computer Vision, in der Verarbeitung natürlicher Sprache und in anderen Bereichen weit verbreitet. Das Convolutional Neural Network konstruiert den visuellen Wahrnehmungsmechanismus der Kreatur, der beaufsichtigtes Lernen und unüberwachtes Lernen durchführen kann. Die Struktur wird im folgenden Bild gezeigt:

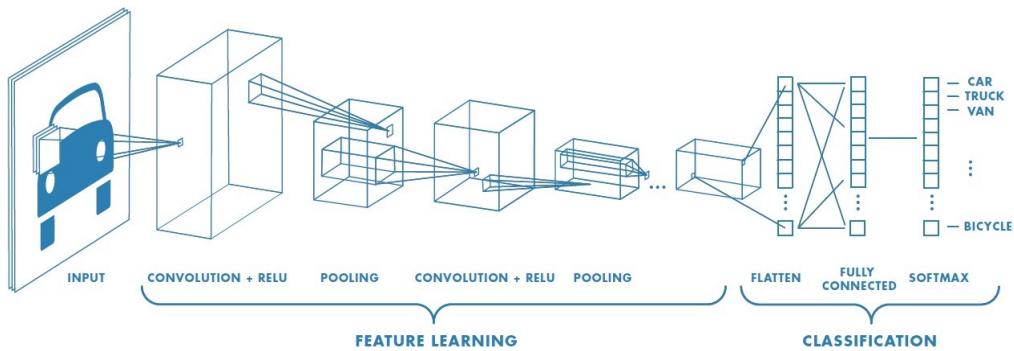


Abbildung 11: die Struktur von CNN[23]

Anhand des oberen Bilds kann man wissen: Die Convolutional Neural Networks (CNN) bestehen aus zwei Schritten: Feature-learning und Classification.

Transposed-CNN: Die transponierten Convolutional Neural Networks (CNN) stellt die gleiche Konnektivität wie die normale Faltung dar, jedoch in Rückwärtsrichtung. Wir können damit Up-Sampling durchführen. Außerdem sind die Gewichte in der transponierten Faltung erlernbar. Daher benötigen wir keine vorgegebene Interpolationsmethode.

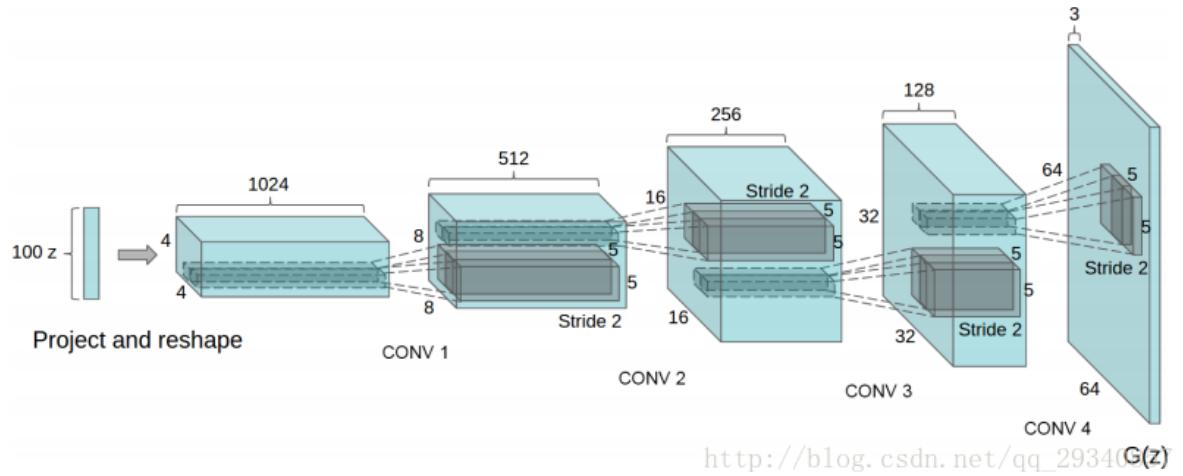


Abbildung 12: Die Struktur von transponierten CNN[23]

CNN und Transposed-CNN stellen ein Paar reziproker Algorithmen dar. Aus der obigen Abbildung ist ersichtlich, dass CNN zum Extrahieren der Bildmerkmale verwendet wird und transponiertes CNN ist das Merkmal des wiederhergestellten Bildes.

## 3.2 Die Ideen und Struktur von GC-Net

GC-Net ist ein Deep Learning Methode für Stereo Vision. Es verwendet hauptsächlich die Klassifizierung oder Regression, um den Pixelabgleich zu lösen. Das Netzwerk ist in vier Teile unterteilt. Die erste Teil reicht vom ersten Layer bis zum achtzehnten Layer mit 2D-conv (Unary features). Der zweite Teil ist Cost-Volume zu machen (zwei Features verbinden). Der dritte Teil ist Learning Regularization mit 3-D conv und 3-D Transposed conv. Der vierte Teil ist Soft-argmin für die Disparität zu regressieren, und dann mit Hilfe der Mae-Lossfunktion die Accuracy zu rechnen. Die untere Abbildung[24] ist eine Übersichtskarte für die Struktur von GC-Net.

Layer Description		Output Tensor Dim.
Input image		H×W×C
<b>Unary features (section 3.1)</b>		
1	5×5 conv, 32 features, stride 2	$\frac{1}{2}H \times \frac{1}{4}W \times F$
2	3×3 conv, 32 features	$\frac{1}{2}H \times \frac{1}{4}W \times F$
3	3×3 conv, 32 features	$\frac{1}{2}H \times \frac{1}{4}W \times F$
4-17	add layer 1 and 3 features (residual connection) (repeat layers 2,3 and residual connection) × 7	$\frac{1}{2}H \times \frac{1}{4}W \times F$
18	3×3 conv, 32 features, (no ReLu or BN)	$\frac{1}{2}H \times \frac{1}{4}W \times F$
<b>Cost volume (section 3.2)</b>		
Cost Volume		$\frac{1}{2}D \times \frac{1}{2}H \times \frac{1}{2}W \times 2F$
<b>Learning regularization (section 3.3)</b>		
19	3-D conv, 3×3×3, 32 features	$\frac{1}{2}D \times \frac{1}{4}H \times \frac{1}{4}W \times F$
20	3-D conv, 3×3×3, 32 features	$\frac{1}{2}D \times \frac{1}{4}H \times \frac{1}{4}W \times F$
21	From 18: 3-D conv, 3×3×3, 64 features, stride 2	$\frac{1}{4}D \times \frac{1}{4}H \times \frac{1}{4}W \times 2F$
22	3-D conv, 3×3×3, 64 features	$\frac{1}{4}D \times \frac{1}{4}H \times \frac{1}{4}W \times 2F$
23	3-D conv, 3×3×3, 64 features	$\frac{1}{4}D \times \frac{1}{4}H \times \frac{1}{4}W \times 2F$
24	From 21: 3-D conv, 3×3×3, 64 features, stride 2	$\frac{1}{8}D \times \frac{1}{8}H \times \frac{1}{8}W \times 2F$
25	3-D conv, 3×3×3, 64 features	$\frac{1}{8}D \times \frac{1}{8}H \times \frac{1}{8}W \times 2F$
26	3-D conv, 3×3×3, 64 features	$\frac{1}{8}D \times \frac{1}{8}H \times \frac{1}{8}W \times 2F$
27	From 24: 3-D conv, 3×3×3, 64 features, stride 2	$\frac{1}{16}D \times \frac{1}{16}H \times \frac{1}{16}W \times 2F$
28	3-D conv, 3×3×3, 64 features	$\frac{1}{16}D \times \frac{1}{16}H \times \frac{1}{16}W \times 2F$
29	3-D conv, 3×3×3, 64 features	$\frac{1}{16}D \times \frac{1}{16}H \times \frac{1}{16}W \times 2F$
30	From 27: 3-D conv, 3×3×3, 128 features, stride 2	$\frac{1}{32}D \times \frac{1}{32}H \times \frac{1}{32}W \times 4F$
31	3-D conv, 3×3×3, 128 features	$\frac{1}{32}D \times \frac{1}{32}H \times \frac{1}{32}W \times 4F$
32	3-D conv, 3×3×3, 128 features	$\frac{1}{32}D \times \frac{1}{32}H \times \frac{1}{32}W \times 4F$
33	3×3×3, 3-D transposed conv, 64 features, stride 2 add layer 33 and 29 features (residual connection)	$\frac{1}{16}D \times \frac{1}{16}H \times \frac{1}{16}W \times 2F$
34	3×3×3, 3-D transposed conv, 64 features, stride 2 add layer 34 and 26 features (residual connection)	$\frac{1}{8}D \times \frac{1}{8}H \times \frac{1}{8}W \times 2F$
35	3×3×3, 3-D transposed conv, 64 features, stride 2 add layer 35 and 23 features (residual connection)	$\frac{1}{4}D \times \frac{1}{4}H \times \frac{1}{4}W \times 2F$
36	3×3×3, 3-D transposed conv, 32 features, stride 2 add layer 36 and 20 features (residual connection)	$\frac{1}{2}D \times \frac{1}{2}H \times \frac{1}{2}W \times F$
37	3×3×3, 3-D trans conv, 1 feature (no ReLu or BN)	D×H×W×1
<b>Soft argmin (section 3.4)</b>		
Soft argmin		H×W

Table 1: Summary of our end-to-end deep stereo regression

Mein Programm basiert auf dem obigen Netzwerk an dem einige Änderungen vorgenommen wurden. Die erste Änderung besteht darin, dass ich die Regressionsmethode durch eine Klassifizierungsmethode ersetzt habe, und die zweite Änderung ist, dass die Position von Disparität in der vierten Dimension liegt statt in der zweiten Dimension. Es gibt drei wichtige Punkte in diesem Strukturdiagramm, der erste ist residual connection, der Output von Layer 1 muss den Output von Layer 3 addieren und so weiter, der zweite ist der Eingang des Layers 21 der Ausgang von 18 Layers und so weiter. Der dritte ist, dass Layer 18 und Layer 37 keine ReLU(Activationsfunktion) haben und BN (Batchnormalization). Der Zweck hiervon ist, das Hochfrequenzsignal und das Niederfrequenzsignal in das Bild einzufügen, so dass die Kanten im Bild offensichtlicher sind und der Effekt der Vorhersage besser ist. Die folgende Abbildung zeigt ein visuelles Flussdiagramm des GC-Netzwerk.

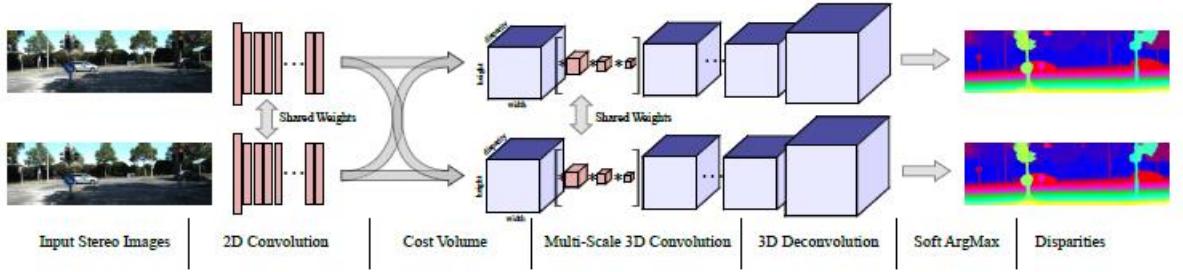


Figure 1: Our end-to-end deep stereo regression architecture, GC-Net (Geometry and Context Network).

Abbildung 13: die Struktur von GC-Net[24]

Es gibt insgesamt 38 Layer. Zuerst gibt es zwei Inputs: Linkes-Original-Bild und Rechtes-Original-Bild. Die Datenbank kommt von Sceneflow-Dataset [40, 41]. Die Bilder vom Sceneflow-Dataset werden im nächsten Kapitel gezeigt. Die Größe jedes Bilds ist Channel = 3, Höhe = 540, Weite = 960, aber in unserem Training ist das Gedächtnis(GPU) jedoch begrenzt. Wenn Sie das gesamte Bild trainieren, führt dies zu einer Überlastung des Trainings und zu einer Verlangsamung der Effizienz. Also nimmt man nur einen Teil des gesamten Bildes (Channel = 3, Höhe = 256, Weite = 512). Jeder Tensor hat 2\*2 Bilder: linkes Bild, rechtes Bild, linke Disparitätskarte und rechte Disparitätskarte. Der Input von diesem neuronalen Netzwerk sind das linke Originalbild und rechte Originalbild. Das Label ist die linke Disparitätskarte. Das Sceneflow-Dataset hat insgesamt 35454 Trainingsbilder. Das Sceneflow-Dataset hat zwei Vorteile: Der erste Vorteil ist, dass die Abweichung zwischen dem Label und der Abbildung beseitigt ist. Zweite Vorteil ist das, dass das Sceneflow-Dataset groß genug ist, um Over-Fitting zu vermeiden. Die Over-Fitting ist ein häufiges Problem für neuronale Netzwerke. Wenn wir das Sceneflow-Dataset als den Input möchten, müssen die Bilder vorher etwas verarbeitet werden. Im nächsten Kapitel wird die Programmstruktur und der wichtigste Python-Code gezeigt. Die folgende Abbildung zeigt die Disparity Prediction der Kitti-Datenbank im Rahmen des GC-Netzwerktrainings.

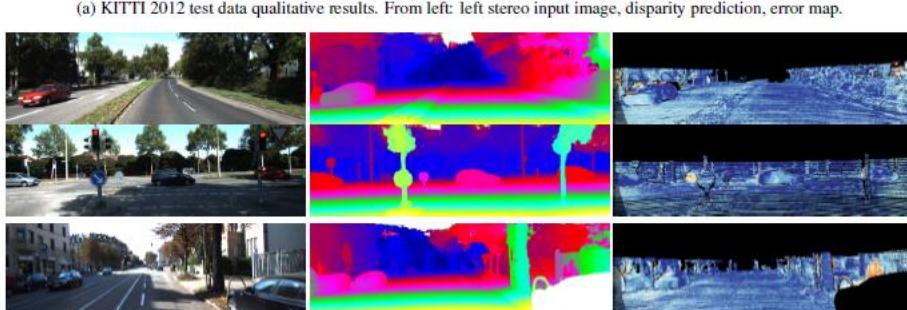
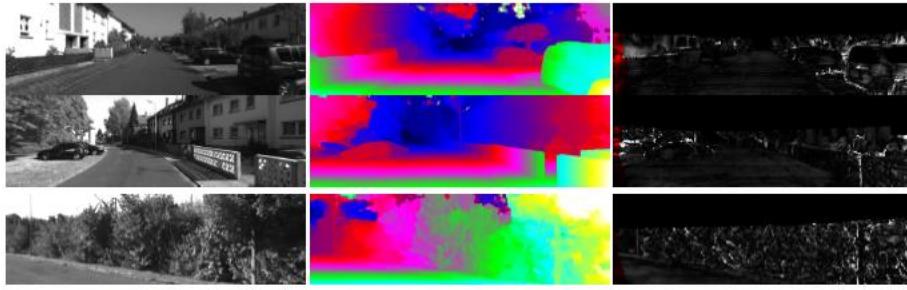


Abbildung 14: Der Effekt von GC-Nets Prediction[24]

Aus Tabelle 1 und Abbildung 11 kann man wissen, dass die grundlegende GC-Netzwerkstruktur darin besteht, dass die ersten 18 Schichten 3 x 3-CNN-Netzwerke sind. Es ist auch notwendig, die Tensoren der Ausgabe jeder ungeraden Schicht zu addieren und dann der geraden Schicht hinzuzufügen. Kombinieren Sie anschließend die beiden Ausgabeergebnisse und fügen Sie die Disparitätsdimension hinzu (192).

Zu dieser Zeit haben wir einen fünfdimensionalen Tensor erhalten Batch\_size, Height, Weight, Feature und Disparitätsdimension. Das ist das Cost-Volume. Legen Sie dann das Kostenvolumen in das CNN-Netzwerk und dann in das transposed CNN-Netzwerk. Schließlich wird die Soft-Argmin-Funktion verwendet, um jeder Pixel mit der entsprechenden Wahrscheinlichkeit zu multiplizieren, und addieren Sie den Durchschnitt als Output

### 3.2.1 Implementierung mit Tensorflow

TensorFlow ist eine End-to-End-Open-Source-Plattform für maschinelles Lernen und verfügt über ein umfassendes, flexibles Ökosystem an Tools, Bibliotheken und Community-Ressourcen, mit dem Forscher den neuesten Stand der Technik in ML und Entwickler auf einfache Weise entwickeln und bereitstellen können Angetriebene Anwendungen[25]. Für diese Arbeit wurde die keras unter tensorflow-Bibliothek mit Programmiersprache Python benutzt. Die Pythonprogramm hat insgesamt drei Teile: Hauptprogramm.py, GC-Net.py, Aufrufe1.py. Das Hauptprogramm hat die Aufgabe, die Bilder aus dem Dataset herauszunehmen und das Training und das Speichern des Datensatzes durchzuführen. Da ich einzeln trainiere, extrahiere ich jeweils nur einen Datensatz. Dieser Datensatz enthält insgesamt vier Tensoren. Der erste und der zweite Tensor entsprechen jeweils linke und rechte Disparitätskarten, und der dritte und vierte Tensor entsprechen jeweils linke und rechte Disparitätskarten. Die folgenden drei Bilder sind die, die ich dieses Mal verwenden werde, die Parallax-Karte und die Tiefenkarte. Abbildung 13 und 14 sind die Eingabe des Trainings und Abbildung 15 sind die Disparitätskarten unseres Vergleichs. Beim Kosten-Volumen. Wir sperren die rechte Disparitätskarte und lassen die linke Disparitätskarte mit der rechten Disparitätskarte interagieren.



Abbildung 15: Linkes Original Bild[26]



Abbildung 16: Rechtes Original Bild[26]

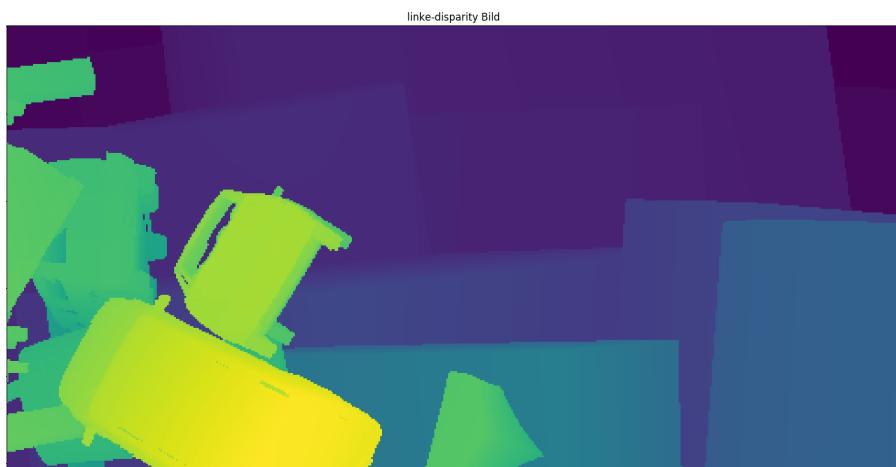


Abbildung 17: Linkes Disparitäts-Bild[26]

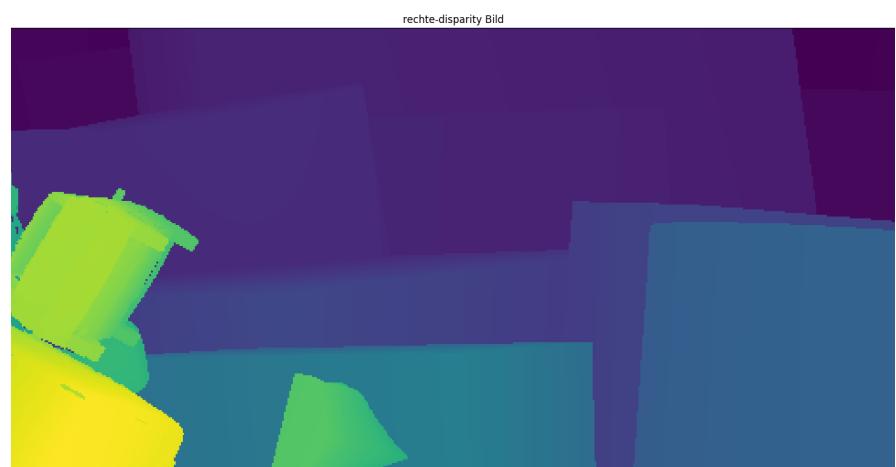


Abbildung 18: Rechtes Disparitäts-Bild[26]

Die Visualisierung des Trainingsprozesses kann mit dem von Tensorflow zur Verfügung gestellten Hilfsprogramm Tensorboard erfolgen. Sie wird in späteren Ergebnissen und Zusammenfassungen dargestellt. Am Ende werde ich den Code des Hauptprogramms anfügen. Das Aufrufprogramm ist der 38-schichtige Aufbau des gesamten Netzwerks, aber das Cost-Volumen und die letzte Softargmin-Funktion haben sich geändert. Es enthält Aufrufprogramm1 und Aufrufprogramm3. Sie sind für die Zusammensetzung der Cost-Volumen, die tf.Slice, tf.expand und tf.stack verwenden. Da ich die Keras-API im Tensorflow-API verwende, zeigt sich die Notwendigkeit, die verschiedenen Teile für das Datentraining in einem Modell zu kapseln. Wenn Sie also die Daten herausnehmen und ändern müssen, müssen Sie die Lambda-Funktion verwenden, um sie einzuwickeln. Zum Schluss werde ich den Code anhängen und erläutern. Aufrufprogramm2 wird hauptsächlich verwendet, um das Prediction-Bild anzuzeigen und die Genauigkeit des Bildes zu berechnen. Als nächstes werde ich die wichtiges Funktion zeigen[27, 28].

1. `tf.keras.layers.Conv2D` :

Diese Schicht erstellt einen Faltungskern, der mit der Ebeneneingabe gefaltet wird, um einen Ausgangstensor zu erzeugen. Wenn Sie diese Ebene als erste Ebene im Modell verwenden, geben Sie den Schlüsselwortparameter Input-Shape (Ganzzahl-Tupel ohne die Beispielachse) an, wie `input_shape = (128, 128, 3)`, für `data-format = channels-last`". 128x128 RGB-Bild

2. `tf.keras.layers.Conv2DTranspose`:

Transposed convolution layer. Das Bedürfnis nach transponierten Windungen ergibt sich im Allgemeinen aus dem Wunsch, eine Transformation zu verwenden, die in die entgegengesetzte Richtung einer normalen Windung geht, d.h. von etwas, das die Form der Ausgabe einer gewissen Windung hat, in etwas, das die Form seiner Eingabe hat, während eine Windung beibehalten wird Verbindungs muster, das mit der Faltung kompatibel ist.

3. `tf.keras.layers.concatenate`:

Als Eingabe wird eine Liste von Tensoren verwendet, die bis auf die Verkettungsachse alle dieselbe Form haben, und die Ausgabe ist ein einzelner Tensor, die Verkettung aller Eingaben.

4. `tf.Data.dataset.make_one_shot_iterator().get_next()`:

Wir können Tensoren nacheinander aus dem Datensatz entnehmen.

5. `tf.stack`:

Packt die Liste der Tensoren in Werten in einen Tensor mit einem um eins höheren Tensorwert, indem sie entlang der Achsendimension gepackt werden. Geben Sie eine Liste der Länge N der Tensoren der Form (A, B, C) an.

6. `tf.reduce_mean`:

Berechnet den Mittelwert von Elementen über die Dimensionen eines Tensors. Wenn `keepdims` wahr ist, wird der Rang des Tensors für jeden Eintrag in der Achse um 1 verringert. Wenn `keepdims` falsche ist, werden die reduzierten Abmessungen mit der Länge 1 beibehalten.

7. `tf.keras.layers.BatchNormalization`:

Normalisiert die Aktivierungen der vorherigen Schicht bei jeder Charge, d.h. wendet an und hält die mittlere Aktivierung nahe bei 0 und die Aktivierungsstandardabweichung nahe bei 1.

Die oben genannten Funktionen sind nur ein Teil davon, aber sie werden sehr gut mit dem Tensor verarbeitet, der häufig in Tensorflow verwendet wird. Ein weiterer Vorteil der Verwendung von Tensorflow besteht darin, dass Sie Tensorboard verwenden können, um Änderungen von Daten und Änderungen von Cost-Wert zu überwachen, Sie können Ihre Netzwerkstruktur in Form eines Flussdiagramms anzeigen, sodass Sie sie auf einen Blick sehen können. Da ich eine Klassifizierungsmethode verwende, muss ich den Fokus darauf legen, den Softmax-Klassifikator zu erklären. Für eine Eingabe x möchten wir wissen, um

welche der N Kategorien es sich handelt.

Angenommen, wir verfügen über ein Modell, das N Kategorien von Bewertungen für die Eingabe x ausgeben kann: Je höher die Bewertung, desto wahrscheinlicher ist es, dass x die Kategorie ist, und diejenige mit der höchsten Bewertung wird als die richtige Kategorie für x betrachtet. Der Softmax-Eingang hat viele Werte, und die diesen Werten entsprechende Wahrscheinlichkeit wird nach der Berechnung der folgenden Abbildung erhalten. Je besser das vorherige CNN-Netzwerk ausgelegt ist, desto näher liegt die Wahrscheinlichkeit bei 1. Ansonsten näher an 0. Es ist also eine sehr wichtige Klassifizierungsfunktion in der Klassifizierungsmethode.

Softmax[38] Die mathematische Formel wird definiert als:

$$\sigma(z)_j = \frac{e_j^z}{\sum_{k=1}^K e_k^z}$$

Wobei j = 1, ..., K ist (von 1 bis K)

Unter diesen ist  $Z_j$  die Ausgabe der Vorstufen\_Ausgabe\_Einheit des Klassifizierers. j steht für den Kategorieindex und die Gesamtzahl der Kategorien ist K. Z stellt das Verhältnis des Index des aktuellen Elements zur exponentiellen Summe aller Elemente dar. Softmax wandelt Ausgabewerte für mehrere Kategorien in relative Wahrscheinlichkeiten um, die einfacher zu verstehen und zu vergleichen sind.

### 3.2.2 Die Änderung von GC-Net und eigenes Netzwerk

Nach der Regressionsmethode in der Arbeit wurde sie von vielen Forschern praktiziert. Es gibt jedoch eine andere Methode, die nicht vollständig getestet wurde: der harten Klassifizierung. Obwohl das Klassifizierungsverfahren und das neuronale Netzwerk des GC-Netzwerks strukturell gleich sind, unterscheiden sich der endgültige Verlustwert und der Ausgabewert von dem Regressionsverfahren. Die Hauptstruktur ist ein neuronales Netzwerk, das die linken und rechten Originalbilder durch 38 Schichten leitet, wodurch ein dreidimensionaler Tensor erhalten wird, das Klassifizierungsverfahren besteht darin, eine Soft-Max-Aktivierungsfunktion für die Parallaxendimension auszuführen, um eine Wahrscheinlichkeit zu erhalten, die verschiedenen Disparitätswerten entspricht. Unser Verlustwert basiert auf der Spares\_cross\_entropy für die harte Klassifizierung. Der Grund für die Bezeichnung harte Klassifizierung ist, dass die Spares\_cross\_entropy [29, 30] die Label in der Form one-hot machen kann, deshalb liegen die Pixel der Prediction nur als int-Type (ganze Zahl) vor. Das vorhergesagte Bild, das durch die Regressionsmethode erhalten wird, ist ein Float-Typ, daher ist es genauer.

	>2 px		>3 px		>5 px		Mean Error		Runtime (s)
	Non-Occ	All	Non-Occ	All	Non-Occ	All	Non-Occ	All	
SPS-st [44]	4.98	6.28	3.39	4.41	2.33	3.00	0.9 px	1.0 px	2
Deep Embed [8]	5.05	6.47	3.10	4.24	1.92	2.68	0.9 px	1.1 px	3
Content-CNN [32]	4.98	6.51	3.07	4.29	2.03	2.82	0.8 px	1.0 px	0.7
MC-CNN [50]	3.90	5.45	2.43	3.63	1.64	2.39	0.7 px	0.9 px	67
PBCP [40]	3.62	5.01	2.36	3.45	1.62	2.32	0.7 px	0.9 px	68
Displets v2 [18]	3.43	4.46	2.37	3.09	1.72	2.17	0.7 px	0.8 px	265
GC-Net (this work)	<b>2.71</b>	<b>3.46</b>	<b>1.77</b>	<b>2.30</b>	<b>1.12</b>	<b>1.46</b>	<b>0.6 px</b>	<b>0.7 px</b>	0.9

(a) KITTI 2012 test set results [14]. This benchmark contains 194 train and 195 test gray-scale image pairs.

Abbildung 19: Genauigkeit verschiedener Algorithmen bei unterschiedlichen Fehlertoleranzen im Jahr 2012 [29]

	All Pixels			Non-Occluded Pixels			Runtime (s)
	D1-bg	D1-fg	D1-all	D1-bg	D1-fg	D1-all	
MBM [11]	4.69	13.05	6.08	4.33	12.12	5.61	0.13
ELAS [15]	7.86	19.04	9.72	6.88	17.73	8.67	0.3
Content-CNN [32]	3.73	8.58	4.54	3.32	7.44	4.00	1.0
DispNetC [34]	4.32	<b>4.41</b>	4.34	4.11	<b>3.72</b>	4.05	<b>0.06</b>
MC-CNN [50]	2.89	8.88	3.89	2.48	7.64	3.33	67
PBCP [40]	2.58	8.74	3.61	2.27	7.71	3.17	68
Displets v2 [18]	3.00	5.56	3.43	2.73	4.95	3.09	265
GC-Net (this work)	<b>2.21</b>	6.16	<b>2.87</b>	<b>2.02</b>	5.58	<b>2.61</b>	0.9

(b) KITTI 2015 test set results [35]. This benchmark contains 200 training and 200 test color image pairs. The qualifier ‘bg’ refers to background pixels which contain static elements, ‘fg’ refers to dynamic object pixels, while ‘all’ is all pixels (fg+bg). The results show the percentage of pixels which have greater than three pixels or 5% disparity error from all 200 test images.

Aus den ersten und zweiten Bildern sind ersichtlich, dass die Trainingszeit des GC-Netzwerks in allen Netzwerken mit dem gleichen Datensatz zu unterschiedlichen Zeiten mittel ist, die Fehlerrate jedoch am niedrigsten ist.

Die in diesem Artikel beschriebene Methode besteht darin, nach den ersten 18 Schichten einen fünfdimensionalen Tensor aufzubauen, [Batch\_size, Disparität, Weight, Höhe, Features], dann erhält man einen 3D-Tensor durch conv3d und convtranspos3d[Disparität, Weight, Höhe], unsere Idee ist die gleiche wie hier, aber später habe ich die Klassifizierungsmethode anstelle der Regressionsmethode verwendet. Ich benutze die Softmax-Funktion, um Disparitätswerte zu klassifizieren. Daher hat sich auch die Loss-Funktion geändert: Wenn es sich um eine Regressionsmethode handelt, wird die MAE(Mean-Absolut-Error)-Loss-Funktion verwendet. Für die Klassifizierungsmethode verwenden wir jedoch sparse\_categorical\_crossentropy. Ich werde die Vorteile des sparse\_categorical\_crossentropy im Abschnitt mit dem Vergleich beider Methoden erläutern.

Meine Netzwerkstruktur basiert auf dem GC-Netzwerk. Die größte Änderung betrifft die Gestaltung des Kosten-Volumens und die Änderung von der Klassifizierungsmethode zur Regressionsmethode. Durch Tensorboard erhielt ich das Netzwerkstrukturdiagramm, das ich mit Tensorflow entworfen habe.

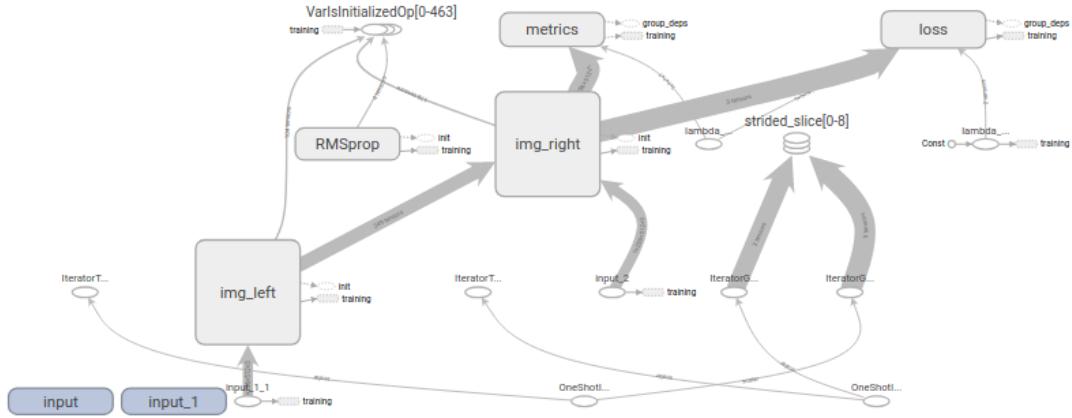


Abbildung 20: die Struktur von die Geändertes GC-Netzwerk[32]

Tensorboard-Diagramme haben zwei Arten von Verbindungen: Datenabhängigkeiten und Steuerungsabhängigkeiten. Die Datenabhängigkeit zeigt den Tensorfluss zwischen zwei Operationen und wird als ausgeföllter Pfeil dargestellt, während die Abhängigkeit der Steuerelemente als gestrichelte Linie dargestellt ist.

Um Unordnung zu vermeiden, unterteilt das visuelle Anzeigewerkzeug alle übergeordneten Knoten in den Hilfsbereich rechts und zeichnet keine Linien, um deren Kanten darzustellen. Wir verwenden das Knotensymbol, um die Verbindung anzuzeigen, ohne eine Linie zu verwenden. Kritische Informationen werden normalerweise nicht entfernt, wenn der sekundäre Knoten gelöscht wird, da diese Knoten normalerweise mit Buchhaltungsfunktionen verbunden sind. Das untere Bild zeigt das.

Der letzte Schritt ist das Trainieren: Da die Keras-Struktur das gesamte Modell trainieren soll, fügt man dem Modell die Trainingsverlustfunktion und Genauigkeit (Accuracy) sowie den Optimierer hinzu. Die Loss-Funktion ist Spares\_categorical\_crossentropy. Learning-rate ist 0.0001. Die Optimierer ist Adam. Batch\_size ist 1, Da dieses Einzeltraining Verlust- und Genauigkeitsänderungen besser beobachten kann, ist dies effizienter. Es sind 5000 Epochen. Die Steps pro Epoche sind 30. So wurden insgesamt 150.000 Trainingsschritte durchgeführt. Im vierten Teil werden die Auswirkungen des Trainings und die Auswirkungen des Tests gezeigt.

## 4 Trainingsergebnisse und Testergebnisse der Klassifizierungsmethode

### 4.1 Der Loss-Wert und Accuracy der Klassifizierungsmethode

Die Genauigkeit (Accuracy) und Loss-Wert sind zwei wichtigen Kriterien für die Prediction. Insbesondere wenn das neuronale Netzwerk optimiert ist, kann die Genauigkeit verwendet werden, um die Genauigkeit der Prediction widerzuspiegeln. Wie in der folgenden Abbildung dargestellt, wird die Genauigkeit unter der Klassifizierungsmethode angezeigt.

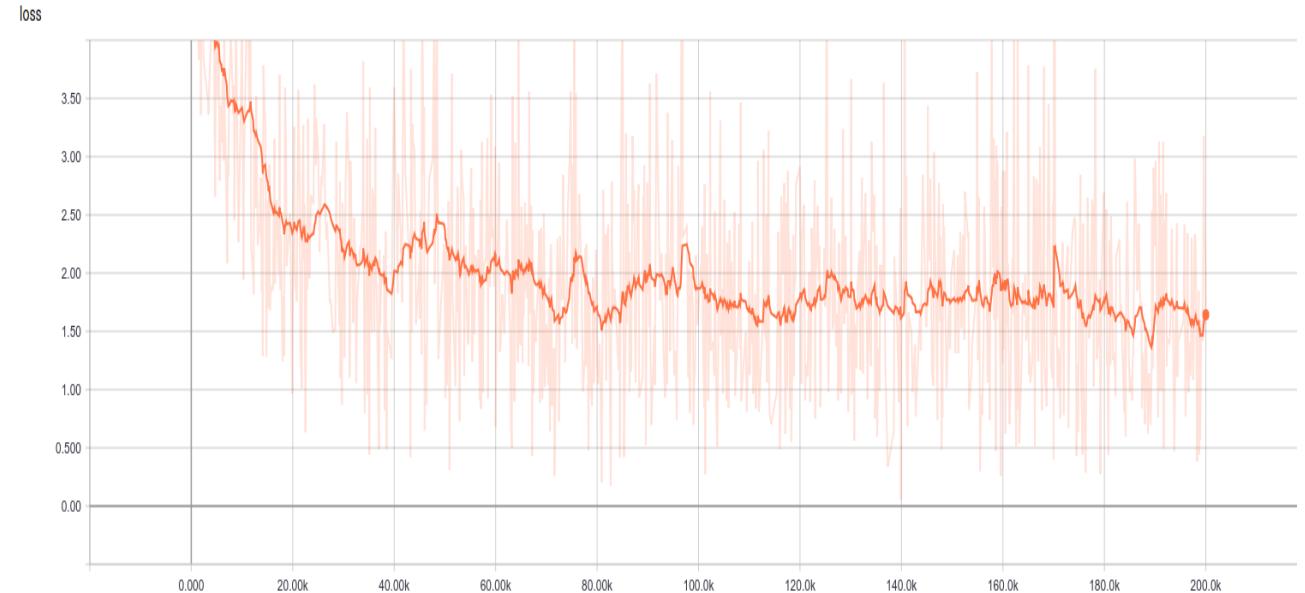


Abbildung 21: Änderung des Train-Set-Loss-Werts

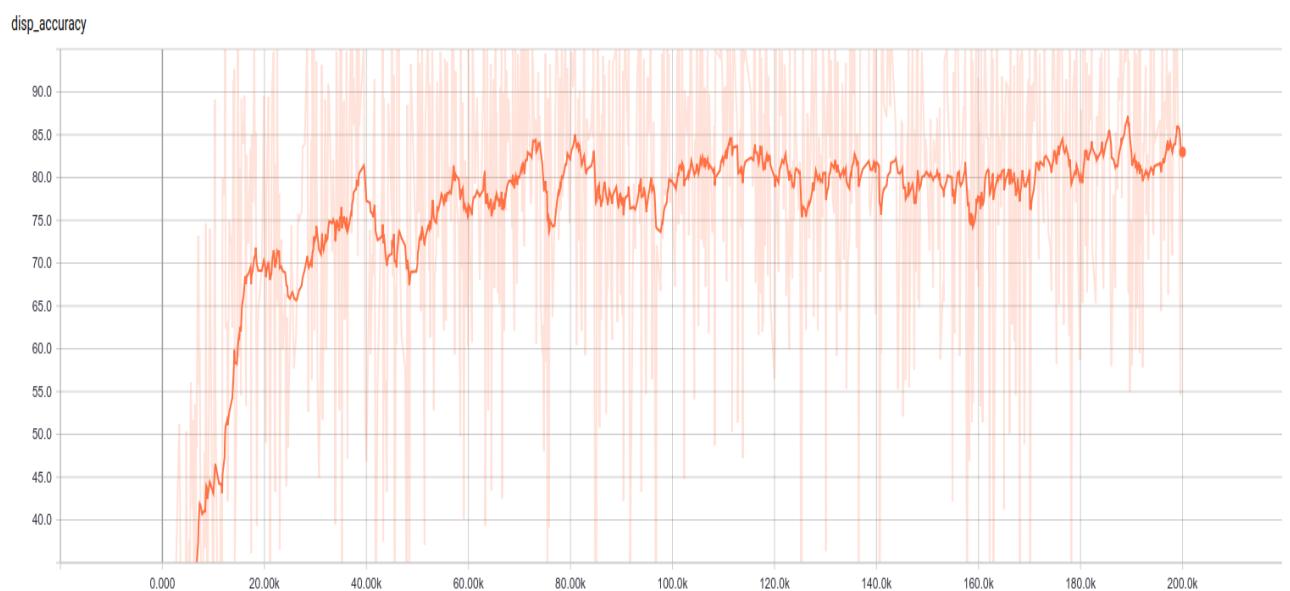


Abbildung 22: Änderung von disp\_accuracy

Wie oben gezeigt, führte ich 30 Iterationen durch und trainierte jedes Mal 5000 Epochen. Es ist sehr intuitiv zu sehen, dass sich die Genauigkeit am Anfang sehr schnell ändert und die Änderung von 0 auf 70 in nur fünf Iterationen erfordert. Die nachfolgende Genauigkeit ändert sich jedoch sehr langsam von 70 Prozent auf 82,5 Prozent bei 25 Iterationen. Dies entspricht auch der Genauigkeitsänderung bei der Klassifizierung im Papier. Gleichzeitig habe ich mich auch während des Trainings dem Verifikationsset angenähert. Die Änderung der Verlustfunktion ist auch von den ersten 3,95 auf die letzten 1,68 offensichtlich. Die helle Kurve in der obigen Abbildung ist der tatsächliche Verlustwert und der Genauigkeitsänderungswert, die dunkle Kurve ist nach dem Einstellen der Glätte. Man erkennt, dass der anfangs die Veränderung schnell vorstatten geht und dann allmählich verlangsamt wird, was auch dem Gesetz des Tiefenlernens entspricht, da es immer schwieriger wird, kleinere Verlustfunktionswerte zu finden. Die folgende Abbildung zeigt das Diagramm der experimentellen Genauigkeitsänderung.

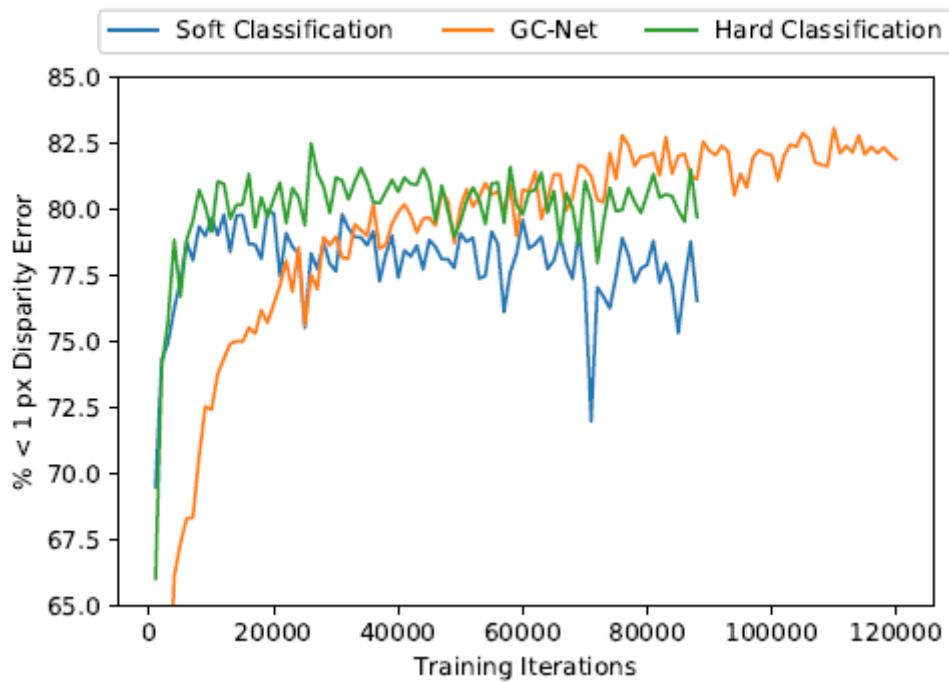


Abbildung 23: Experimentelle Daten im Papier[39]

Die von mir verwendete Klassifizierungsmethode ist in der obigen Abbildung eine harte Klassifizierung. Die Änderung der grünen Linie ist im Wesentlichen die gleiche wie meine eigene Genauigkeitsänderung. Zum Schluss erreicht die Accuracy als höchsten Wert 82,5 Prozent, so kann man den Erfolg des Experiments beurteilen. Die Folgenden Abbildungen sind die zwei prädiktiven Effekte.

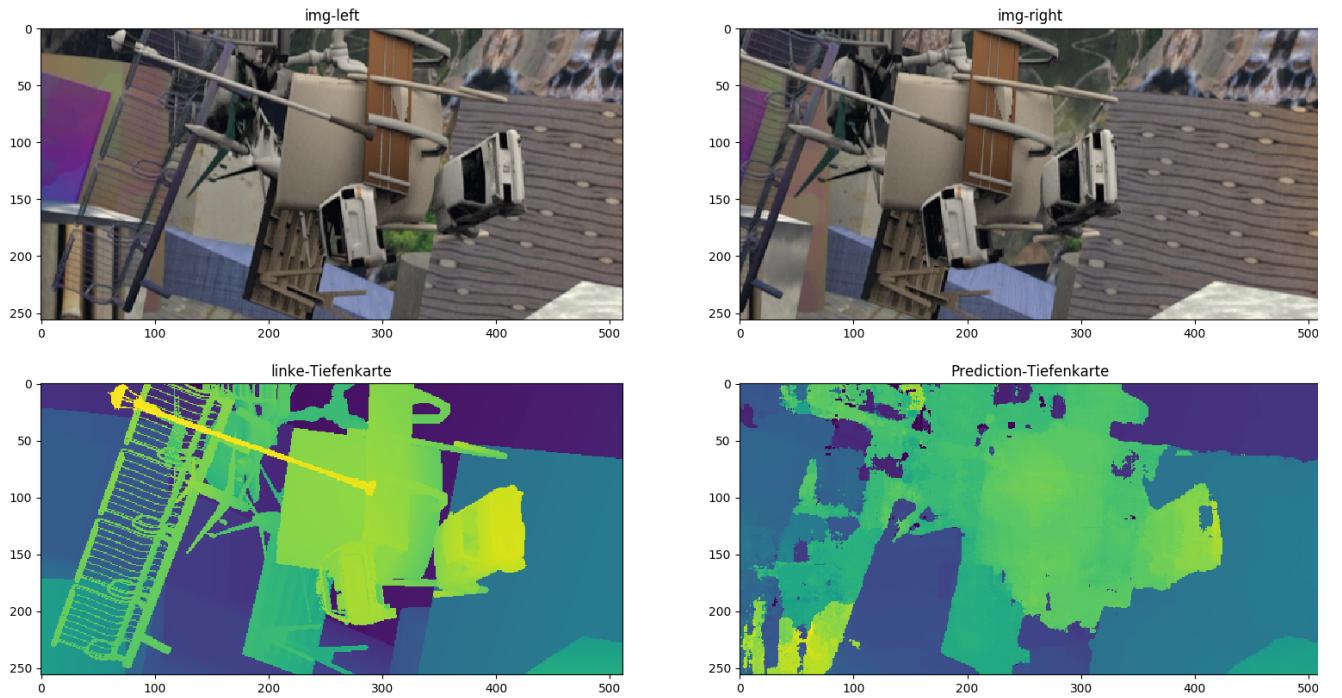


Abbildung 24: Erstes Beispiel der Disparitätsvorhersagen

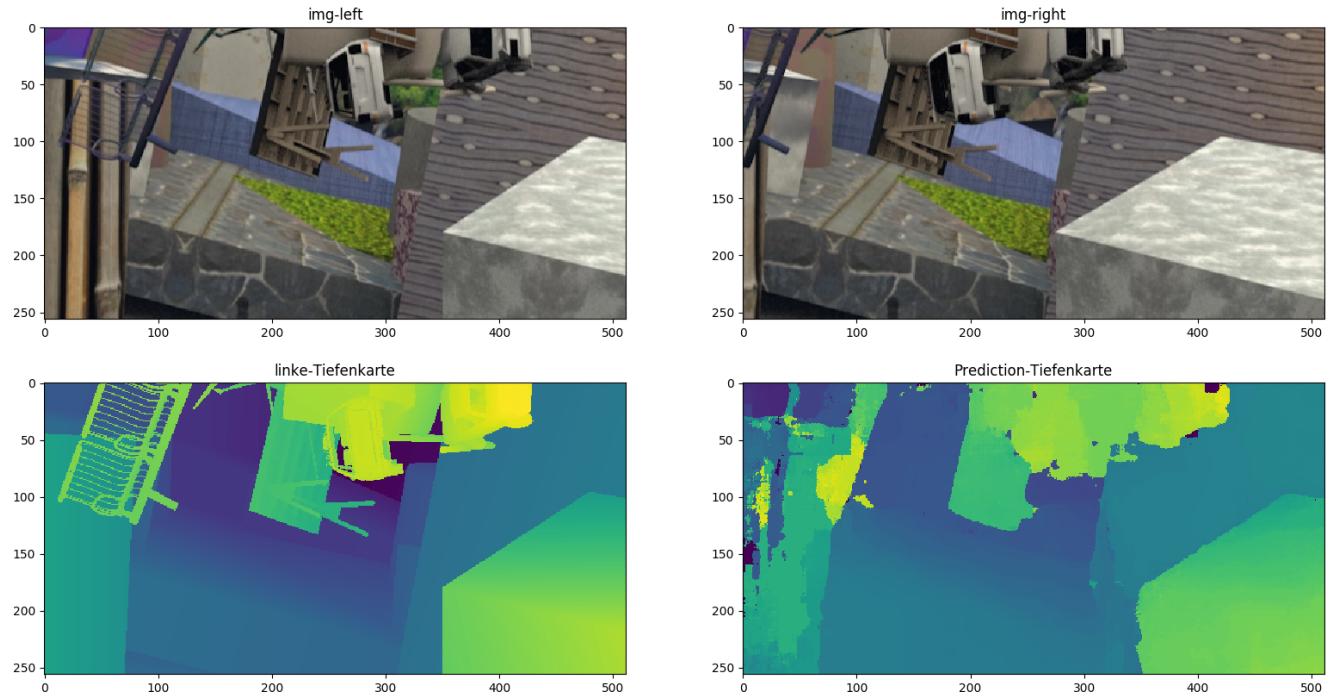


Abbildung 25: Zweites Beispiel der Disparitätsvorhersagen

Die vier Bilder in der obigen Abbildung, oben links und oben rechts, sind die Originalbilder des Eingabetrainingsnetzwerks. Das Bild unten links ist das Label (groundtruth). Das Bild in der rechten unteren Ecke ist die vorhergesagte Disparitätskarte. Es soll mit dem Label (groundtruth) verglichen werden. Nach dem Testen erreichte die Genauigkeit (Acc) schließlich 81,35 Prozent. Es berechnet eine Differenz, indem

aus der Disparitätskarte und dem Pixel die Beschriftung vorhergesagt wird: Wenn der Fehler innerhalb der drei Pixel liegt, ist dies das richtige Pixel, und wenn es größer als drei Pixel ist, ist es das falsche Pixel. Schließlich wird die Anzahl von richtigen Pixeln durch die Anzahl von falschen Pixeln geteilt. Dies ist die Definition der Genauigkeit in diesem Programm.

## 4.2 Der Vergleich zwischen Regressionsmethode und Klassifizierungsmethode

Es gibt zwei Möglichkeiten, um in der Arbeit zu trainieren, eine ist die Klassifizierungsmethode und die andere ist die Regressionsmethode. Da bereits viele Regressionsmethoden verwendet wurden, verwendet dieses Experiment eine Klassifizierungsmethode. Durch den Vergleich eigener Experimente und Papierdaten wurden die folgenden Unterschiede festgestellt.

Klassifizierung: Die Ausgabe der Klassifizierungsmethode basiert auf der Matrix der Softmax-Funktion, bei der es sich tatsächlich um einen Wahrscheinlichkeitswert handelt, bei dem die Wahrscheinlichkeit ein anderer Disparitätswert ist. Wenn Sie dann die spares\_cross\_entropen verwenden, kann das Label one-hot werden. Zum Beispiel, wenn das Label des Disparitätswerts 3 ist, soll es zu [0, 0, 0, 1, ....] transformiert werden, der Disparitätswert mit der höchsten Wahrscheinlichkeit von Softmax ermittelt und er in die Klasse des Labels eingeordnet werden.

Regression: Sie verwendet die Soft\_argmin-Funktion, um den Disparitätswert zu berechnen. Multiplizieren Sie die Wahrscheinlichkeit der Softmax-Ausgabe mit dem entsprechenden Disparitätswert und addieren Sie dann den Durchschnittswert als endgültige Ausgabe. Da der durch Multiplikation erhaltene Disparitätswert ein Float-Typ sein kann und die harte Klassifizierung nur einen Int-Typ(Positive ganze Zahl) erhalten kann, ist theoretisch der Effekt von Soft-argmin besser als die harte Klassifizierung. Durch den Vergleich der beiden Methoden können wir also wissen, dass die Pixelwert der Regressionsmethode genauer als die der Klassifizierungsmethode sind.

## 5 Zusammenfassung und Problem

Durch die Disparitätskarte erhalten wir den Abstand vom Objekt zum Kamerasystem. Ob die Pixelvorhersage der Disparitätskarte genau ist, ist daher das Wichtigste. GC-Net muss die folgenden Probleme lösen, wenn es in die reale Anwendung aufgenommen werden soll.

1. Die Vorhersagegenauigkeit beträgt etwa 82, 5 Prozent, was nicht den Effekt der genauen Vorhersage darstellt.
2. Die Echtzeit-Performance ist nicht stark, da wir uns in der Industrie mit Video beschäftigen müssen und jeden Frame abfangen müssen, was einen schnelleren Prozessor und einen schnelleren Sender erfordert
3. Da GC-Net zum beaufsichtigten Lernen gehört, sind für das Training viele Bilddaten erforderlich, und die Mobilität ist nicht stark. Zum Beispiel trainieren wir das Sceneflow-Dataset. Wenn andere Bildtypen vorhergesagt werden müssen, z. B. Kitti-Daten, müssen wir die Bildverarbeitung noch einmal machen.

In diesem Projekt haben wir nur Vorhersagen für die Disparitätskarte gemacht, und die Ergebnisse der Endgenauigkeit stimmten auch mit den in der Arbeit präsentierten Ergebnissen überein. Für unterschiedliche Toleranzbereiche wurden auch unterschiedliche exakte Werte angegeben. Die Standards aus dem Experiment im Papier wurden auch erfüllt. Durch Vergleichen der Kennzeichnungdisparitätskarte (Label) mit der vorhergesagten Disparitätskarte können wir intuitiv feststellen, dass, obwohl der numerische Unterschied klein ist, einige Formen auf dem Bild nicht besonders offensichtlich sind. Dies ist also noch ein Teil, der noch verbessert werden muss. Im Allgemeinen hat unser Einstufungseffekt die in der Arbeit geforderten Klassifikationsanforderungen erreicht.

Für die zukünftige Entwicklung ist, wie zu Beginn der Arbeit erwähnt, die binokulare Stereo Vision ein sehr wichtiger Teil der Computer Vision, und sie ist auch ein Teil der Forschungsinstitute und Unternehmen, die hart an der Entwicklung künstlicher Intelligenz arbeiten. Insbesondere das autonome Fahren war schon immer die Forschungs- und Entwicklungsrichtung verschiedener Automobilunternehmen und hat große Vorteile gegenüber der Laserentfernung. Mit der Entwicklung der binokularen Stereo Vision wird er definitiv eine wichtigere Rolle in unserem Leben spielen. Die Entwicklung der künstlichen Intelligenz hat noch einen langen Weg. Binokulares Stereo Vision muss ein sehr wichtiger Teil davon sein.

## 6 Anhang

```
1."Hauptprogramm.py"
from Aufrufe import creat_model
import matplotlib.pyplot as plt
from Aufrufe1 import *
from Daten import *
batch_size = 1
dataset = input_from_dataset(buffer=1024)
dataset_test = input_from_dataset(filename="/daten/b/sceneflow_disp_train.tfrecord",
buffer=128)
dataset_test = dataset_test.batch(batch_size)
dataset_test = dataset_test.filter(lambda x, y: tf.less(tf.reduce_max(y), 192))
dataset = dataset.filter(lambda x, y: tf.less(tf.reduce_max(y), 192))
dataset = dataset.repeat()
dataset = dataset.batch(batch_size).prefetch(32)
interator=dataset.make_one_shot_iterator()
next_batch=interator.get_next()
next_batch_test = dataset_test.repeat().make_one_shot_iterator().get_next()
img_left_test = next_batch_test[0][:,0]
img_right_test = next_batch_test[0][:,1]
disp_left_test = next_batch_test[1][:,0]
images=next_batch[0]
disp=next_batch[1]
img_left=images[:,0]
img_right=images[:,1]
disp_left=disp[:,0]
disp_right=disp[:,1]
print(disp)
print('left_shape',img_left)
print('left_disp',disp_left)
limit gpu-mdmory consumption
config = tf.ConfigProto()
config.gpu_options.allow_growth = True
tf.keras.backend.set_session(tf.Session(config=config))
print(disp_left.shape)
shape_tuple=images[0][1]
Model=creat_model(shape_tuple)
print(Model.output_shape)
optimizer=tf.keras.optimizers.RMSprop(lr=0.0001)
Model.compile(optimizer=optimizer,loss=def_loss,metrics=[disp_accuracy])
callback = tf.keras.callbacks.TensorBoard(log_dir=".TB/")
Model.fit([img_left, img_right], disp_left, epochs=30, steps_per_epoch=5000,
callbacks=[callback], validation_steps=int(4000/batch_size), validation_data=([img_left_test, img_right_test], disp_left_test),
verbose=2 )
Model.save('Gc-Net3.h5')
```

```

2.Gc-net.py( Aufruf)
from tensorflow import keras as K
from Aufrufe1 import *
import numpy as np
BatchNormalization = K.layers.BatchNormalization
Conv2D=K.layers.Conv2D
Conv3D=K.layers.Conv3D
Conv3Dtransposed=K.layers.Conv3DTranspose
Aktivation = K.layers.Activation
Conv2Dtransposed=tf.keras.layers.Conv2DTranspose
def creat_model(shape_tuple):
    img_left= K.Input(shape=shape_tuple.shape)
    img_right= K.Input(shape=shape_tuple.shape)
    layer1=Conv2D(32, kernel_size=(5, 5),
    padding='same',
    strides=2,
    activation='relu',
    )
    layer2=Conv2D(32,kernel_size=(3,3),
    padding='same',
    activation='relu')
    layer3=Conv2D(32,kernel_size=(3,3),
    padding='same',
    activation='relu')
    layer4=Conv2D(32,kernel_size=(3,3),
    padding='same',
    activation='relu')
    layer5=Conv2D(32,kernel_size=(3,3),
    padding='same',
    activation='relu')
    layer6=Conv2D(32,kernel_size=(3,3),
    padding='same',
    activation='relu')
    layer7=Conv2D(32,kernel_size=(3,3),
    padding='same',
    activation='relu')
    layer8=Conv2D(32,kernel_size=(3,3),
    padding='same',
    activation='relu')
    layer9=Conv2D(32,kernel_size=(3,3),
    padding='same',
    activation='relu')
    layer10=Conv2D(32,kernel_size=(3,3),
    padding='same',
    activation='relu')
    layer11=Conv2D(32,kernel_size=(3,3),
    padding='same',
    activation='relu')
    layer12=Conv2D(32,kernel_size=(3,3),
    padding='same',

```

```

activation='relu')
layer13=Conv2D(32,kernel_size=(3,3),
padding='same',
activation='relu')
layer14=Conv2D(32,kernel_size=(3,3),
padding='same',
activation='relu')
layer15=Conv2D(32,kernel_size=(3,3),
padding='same',
activation='relu')
layer16=Conv2D(32,kernel_size=(3,3),
padding='same',
activation='relu')
layer17=Conv2D(32,kernel_size=(3,3),
padding='same',
activation='relu')
layer18=Conv2D(32,kernel_size=(3,3),
padding='same',
)
with tf.name_scope("img_left"):
x_l1=layer1(img_left)
x_l1 = BatchNormalization()(x_l1)
x_l2=layer2(x_l1)
x_l2=BatchNormalization()(x_l2)
x_l3 = layer3(x_l2)
x_l3 = BatchNormalization()(x_l3)
x_l3=tf.keras.layers.add([x_l1,x_l3])
x_l4 = layer4(x_l3)
x_l4 = BatchNormalization()(x_l4)
x_l5 = layer5(x_l4)
x_l5 = BatchNormalization()(x_l5)
x_l5 = tf.keras.layers.add([x_l3, x_l5])
x_l6 = layer6(x_l5)
x_l6 = BatchNormalization()(x_l6)
x_l7 = layer7(x_l6)
x_l7 = BatchNormalization()(x_l7)
x_l7 = tf.keras.layers.add([x_l5, x_l7])
x_l8 = layer8(x_l7)
x_l8 = BatchNormalization()(x_l8)
x_l9 = layer9(x_l8)
x_l9 = BatchNormalization()(x_l9)
x_l9=tf.keras.layers.add([x_l9,x_l7])
x_l10 = layer10(x_l9)
x_l10 = BatchNormalization()(x_l10)
x_l11 = layer11(x_l10)
x_l11 = BatchNormalization()(x_l11)
x_l11 = tf.keras.layers.add([x_l9, x_l11])
x_l12 = layer12(x_l11)
x_l12 = BatchNormalization()(x_l12)
x_l13 = layer13(x_l12)

```

```

x_l13 = BatchNormalization()(x_l13)
x_l13 = tf.keras.layers.add([x_l11, x_l13])
x_l14 = layer14(x_l13)
x_l14 = BatchNormalization()(x_l14)
x_l15 = layer15(x_l14)
x_l15 = BatchNormalization()(x_l15)
x_l15 = tf.keras.layers.add([x_l13, x_l15])
x_l16 = layer16(x_l15)
x_l16 = BatchNormalization()(x_l16)
x_l17 = layer17(x_l16)
x_l17 = BatchNormalization()(x_l17)
x_l17 = tf.keras.layers.add([x_l17, x_l15])
x_l18 = layer18(x_l17)

```

```

with tf.name_scope("img_right"):
    x_r1 = layer1(img_right)
    x_r1 = BatchNormalization()(x_r1)
    x_r2 = layer2(x_r1)
    x_r2 = BatchNormalization()(x_r2)
    x_r3 = layer3(x_r2)
    x_r3 = BatchNormalization()(x_r3)
    x_r3 = tf.keras.layers.add([x_r1, x_r3])
    x_r4 = layer4(x_r3)
    x_r4 = BatchNormalization()(x_r4)
    x_r5 = layer5(x_r4)
    x_r5 = BatchNormalization()(x_r5)
    x_r5 = tf.keras.layers.add([x_r5, x_r3])
    x_r6 = layer6(x_r5)
    x_r6 = BatchNormalization()(x_r6)
    x_r7 = layer7(x_r6)
    x_r7 = BatchNormalization()(x_r7)
    x_r7 = tf.keras.layers.add([x_r7, x_r5])
    x_r8 = layer8(x_r7)
    x_r8 = BatchNormalization()(x_r8)
    x_r9 = layer9(x_r8)
    x_r9 = BatchNormalization()(x_r9)
    x_r9 = tf.keras.layers.add([x_r9, x_r7])
    x_r10 = layer10(x_r9)
    x_r10 = BatchNormalization()(x_r10)
    x_r11 = layer11(x_r10)
    x_r11 = BatchNormalization()(x_r11)
    x_r11 = tf.keras.layers.add([x_r11, x_r9])
    x_r12 = layer12(x_r11)
    x_r12 = BatchNormalization()(x_r12)
    x_r13 = layer13(x_r12)
    x_r13 = BatchNormalization()(x_r13)
    x_r13 = tf.keras.layers.add([x_r11, x_r13])
    x_r14 = layer14(x_r13)
    x_r14 = BatchNormalization()(x_r14)

```

```

x_r15 = layer15(x_r14)
x_r15 = BatchNormalization()(x_r15)
x_r15 = tf.keras.layers.add([x_r13, x_r15])
x_r16 = layer16(x_r15)
x_r16 = BatchNormalization()(x_r16)
x_r17 = layer17(x_r16)
x_r17 = BatchNormalization()(x_r17)
x_r17 = tf.keras.layers.add([x_r17, x_r15])
x_r18 = layer18(x_r17)

left_cost=tf.keras.layers.Lambda(expand)(x_l18)
right_cost=tf.keras.layers.Lambda(expand)(x_r18)
cost_volum=tf.keras.layers.concatenate([left_cost,right_cost],axis=4)
total=K.layers.Lambda(cost5)(cost_volum)

x19 = Conv3D(32, kernel_size=(3, 3,3),
padding='same',
activation='relu'
)(total)
x19 = BatchNormalization()(x19)
x20 = Conv3D(32, kernel_size=(3, 3,3), padding='same',
activation='relu')(x19)
x20 = BatchNormalization()(x20)
x21 = Conv3D(64, kernel_size=(3, 3,3), padding='same',
strides=2,
activation='relu')(total)
x21 = BatchNormalization()(x21)
x22 = Conv3D(64, kernel_size=(3, 3,3),
padding='same',
activation='relu'
)(x21)
x22 = BatchNormalization()(x22)
x23 = Conv3D(64, kernel_size=(3, 3,3),
padding='same',
activation='relu'
)(x22)
x23 = BatchNormalization()(x23)
x24 = Conv3D(64, kernel_size=(3, 3,3), padding='same',
strides=2,
activation='relu')(x21)
x24 = BatchNormalization()(x24)
x25 = Conv3D(64, kernel_size=(3,3,3),
padding='same',
activation='relu'
)(x24)
x25 = BatchNormalization()(x25)
x26 = Conv3D(64, kernel_size=(3, 3,3),
padding='same',
activation='relu'
)(x25)

```

```

x26 = BatchNormalization()(x26)
x27 = Conv3D(64, kernel_size=(3,3,3), padding='same',
strides=2,
activation='relu')(x24)
x27 = BatchNormalization()(x27)
x28 = Conv3D(64, kernel_size=(3,3,3),
padding='same',
activation='relu'
)(x27)
x28 = BatchNormalization()(x28)
x29 = Conv3D(64, kernel_size=(3,3,3),
padding='same',
activation='relu'
)(x28)
x29 = BatchNormalization()(x29)
x30 = Conv3D(128, kernel_size=(3,3,3),
padding='same',
strides=2,
activation='relu'
)(x27)
x30 = BatchNormalization()(x30)
x31 = Conv3D(128, kernel_size=(3,3,3), padding='same',
activation='relu')(x30)
x31 = BatchNormalization()(x31)
x32 = Conv3D(128, kernel_size=(3, 3,3),
padding='same',
activation='relu'
)(x31)
x32 = BatchNormalization()(x32)
x33 = Conv3Dtransposed(64, kernel_size=(3,3,3),
padding='same',
strides=2,
activation='relu'
)(x32)
x33 = BatchNormalization()(x33)
x33 = tf.keras.layers.add([x29, x33])
x34 = Conv3Dtransposed(64, kernel_size=(3,3,3),
activation='relu',
strides=2,
padding='same'
)(x33)
x34 = BatchNormalization()(x34)
x34 = tf.keras.layers.add([x26, x34])
x35 = Conv3Dtransposed(64, kernel_size=(3,3 ,3), strides=2,
activation='relu',
padding='same')(x34)
x35 = BatchNormalization()(x35)
x35 = tf.keras.layers.add([x35, x23])
x36 = Conv3Dtransposed(32, kernel_size=(3,3,3), strides=2,
activation='relu',

```

```

padding='same')(x35)
x36 = BatchNormalization()(x36)
x36 = tf.keras.layers.add([x20, x36])
x37 = Conv3DTranspose(1, kernel_size=(3,3,3),
strides=2,
padding='same')(x36)
P=tf.keras.layers.Lambda(softmax)(x37)
return K.Model(inputs=[img_left, img_right], outputs=P)

```

### 3.Aufruf1.py (Alle-Funktion-Programm)

```

import tensorflow as tf
def expand(x):
c=tf.expand_dims(x, axis=4)
return c

def cost5(x):
x_l=x[:, :, :, :, 0]
x_r=x[:, :, :, :, 1]
cost_vol = []
cost_vol1 = []
pad = [[0, 0], [0, 0], [0, 96], [0, 0]]
cost = tf.pad(x_r, paddings=pad)
for i in range(0, 96):
slice = tf.slice(cost, [0, 0, i, 0], [-1, -1, 256, -1])
cost_vol.append(slice)
cost_vol1.append(x_l)
cost_left = tf.stack(cost_vol1, axis=4)
cost_right = tf.stack(cost_vol, axis=4)
total1= tf.concat([cost_left,cost_right],axis=3)
total1= tf.transpose(total1,[0,1,2,4,3])
return total1

def softmax(x):
x=tf.squeeze(x, axis=4)
x=tf.negative(x)
return x

def disp_accuracy(gt, pred):
pred= tf.nn.softmax(pred, axis=3) pred = tf.cast(tf.argmax(pred, 3), tf.float32)
gt = gt[:, :, :, 0]
tf.assert_equal(tf.rank(pred), tf.rank(gt))
valid = tf.ones_like(gt, dtype=tf.int32) tf.cast(tf.cond(tf.equal(tf.rank(gt), 3), lambda:
gt[:, :, 2], lambda: tf.ones_like(gt)), tf.int32)
num_pixel = tf.reduce_sum(valid)
num_correct = tf.reduce_sum(tf.cast(tf.less_equal(tf.abs(pred - gt), 3), tf.int32) * valid)
return (num_correct / num_pixel) * 100

```

```

def def_loss(gt,pred):
    gt=gt[:, :, :, 0]
    gt=tf.cast(gt,dtype=tf.int32)
    loss=tf.reduce_mean(tf.nn.sparse_softmax_cross_entropy_with_logits(labels=gt,logits=pred))
    return loss

def soft_argmin(val_model):
    prob = tf.negative(tf.squeeze(val_model, axis=-1))
    with tf.name_scope('Softmax'):
        cost_vol_norm = tf.nn.softmax(prob, 1)
        d_max_p = cost_vol_norm.shape[1].value * res
        d_values = np.arange(0, 192, step=1, dtype=np.float32).reshape([1, 1, 1, -1])
        mult = tf.multiply(x=d_values, y=cost_vol_norm)
        s_argmin = tf.reduce_sum(input_tensor=mult, axis=-1)
    return s_argmin

```

```

4.Prediction.py
from Daten import *
import matplotlib.pyplot as plt
import tensorflow as tf
import numpy as np
from Aufrufe import creat_model
from Aufrufe1 import *
batch_size = 1
dataset = input_from_dataset(buffer=1024)
dataset_test = input_from_dataset(filename='/daten/b/sceneflow_disp_test.tfrecord',
buffer=128)
dataset_test = dataset_test.batch(batch_size)
dataset = dataset.repeat()
dataset = dataset.batch(batch_size)
iterator=dataset.make_one_shot_iterator()
next_batch=iterator.get_next()
next_batch_test = dataset_test.repeat().make_one_shot_iterator().get_next()
img_left_test = next_batch_test[0][:,0]
img_right_test = next_batch_test[0][:,1]
disp_left_test = next_batch_test[1][:,0]
images=next_batch[0]
disp=next_batch[1]
img_left=images[:,0]
img_right=images[:,1]
disp_left=disp[:,0]
disp_right=disp[:,1]
print(disp)
print('left_shape',img_left)
print('left_disp',disp_left)
config = tf.ConfigProto()
config.gpu_options.allow_growth = True
with tf.Session(config=config) as sess:
    for i in range(1):
        tensor1,tensor2=sess.run([next_batch[0],next_batch[1]])

```

```

l=tensor1[:,0]
l=l[0,:]
r=tensor1[:,1]
r=r[0,:]
b=tensor2[:,0]
b=b[0,:]
labe1=np.reshape(b,(256,512))
a=tensor2[:,1]
a=a[0,:]
labe2=np.reshape(a,(256,512))
shape_tuple=images[0][1]
Model=creat_model(shape_tuple)
Model.load_weights('Gc-Net3.h5')
pre=Model.predict_on_batch([img_left, img_right])
b=np.max(pre, axis=3)
b=b[0,:,:]
plt.figure()
plt.subplot(2,2,1)
plt.imshow(l)
plt.subplot(2,2,2)
plt.imshow(r)

plt.subplot(2,2,3)
plt.imshow(labe1)

plt.subplot(2,2,4)
plt.imshow(b)

plt.show()

```

```

5.Daten.py(Bildvorverarbeitung)
import numpy as np
import tensorflow as tf
import cv2
def decode_sceneflow(serialized_example, kitty=False):
    with tf.name_scope("decode"):
        features = tf.parse_single_example(
            serialized_example,
            *Defaults are not specified since both keys are required.
            features=
            "height": tf.FixedLenFeature([], tf.int64),
            "width": tf.FixedLenFeature([], tf.int64),
            "depth": tf.FixedLenFeature([], tf.int64),
            "image_raw_left": tf.FixedLenFeature([], tf.string),
            "image_raw_right": tf.FixedLenFeature([], tf.string),
            "label_left": tf.FixedLenFeature([], tf.string),
            "label_right": tf.FixedLenFeature([], tf.string)
        )
        height = tf.cast(features["height"], tf.int32)

```

```

width = tf.cast(features["width"], tf.int32)
depth = tf.cast(features["depth"], tf.int32)
* length mnist.IMAGE_PIXELS) to a uint8 tensor with shape
* [mnist.IMAGE_PIXELS].
*image_left = tf.decode_raw(features[image_raw_left], tf.uint8)
*image_right = tf.decode_raw(features[image_raw_right], tf.uint8)
disp_left = tf.decode_raw(features[label_left], tf.float32)
disp_right = tf.decode_raw(features[label_right], tf.float32)
image_left = tf.cast(tf.reshape(image_left, [540, 960, 3]), dtype=tf.float32)
image_right = tf.cast(tf.reshape(image_right, [540, 960, 3]), dtype=tf.float32)
disp_left = tf.reshape(disp_left, [540, 960, 1])
disp_right = tf.reshape(disp_right, [540, 960, 1])
images = tf.stack([image_left, image_right])
labels = tf.stack([disp_left, disp_right])
return images, labels

def decode_kitty(serialized_example):
with tf.name_scope("decode"):
features = tf.parse_single_example(
serialized_example,
features=
{
"height": tf.FixedLenFeature([], tf.int64),
"width": tf.FixedLenFeature([], tf.int64),
"depth": tf.FixedLenFeature([], tf.int64),
'image_raw_left': tf.FixedLenFeature([], tf.string),
'image_raw_right': tf.FixedLenFeature([], tf.string),
'label_left': tf.FixedLenFeature([], tf.string),
})
height = tf.cast(features["height"], tf.int32)
width = tf.cast(features["width"], tf.int32)
depth = tf.cast(features["depth"], tf.int32)
image_left = tf.decode_raw(features[image_raw_left], tf.uint8)
image_right = tf.decode_raw(features[image_raw_right], tf.uint8)
disp_left = tf.decode_raw(features[label_left], tf.float32)
image_left = tf.cast(tf.reshape(image_left, [388, 1240, 3]), dtype=tf.float32)
image_right = tf.cast(tf.reshape(image_right, [388, 1240, 3]), dtype=tf.float32)
disp_left = tf.reshape(disp_left, [388, 1240, 1])
images = tf.stack([image_left, image_right])
labels = tf.stack([disp_left])
return images, labels, props

def augment(image, label):
with tf.name_scope("augment"):
crop_height = 256
crop_width = 512
seed = np.random.randint(low=1, high=32000, dtype=np.int16)
image_cropped = tf.random_crop(image, size=[2, crop_height, crop_width, 3], seed=seed)
label_cropped = tf.random_crop(label, size=[2, crop_height, crop_width, 1], seed=seed)
return image_cropped, label_cropped

```

```

def norm_img(img, label):
    with tf.name_scope(norm_img):
        img = tf.cast(img, dtype=tf.float32)
        r_min = -1
        r_max = 1
        dim = "HWC"
        if dim == NHWC":
            touple = [1, 2, 3]
        else:
            touple = [0, 1, 2]
        *img = tf.cast(img, tf.float32)
        min_val_left = tf.reduce_min(img[0], axis=touple, keepdims=True)
        min_val_right = tf.reduce_min(img[1], axis=touple, keepdims=True)
        max_val_left = tf.reduce_max(img[0], axis=touple, keepdims=True)
        max_val_right = tf.reduce_max(img[1], axis=touple, keepdims=True)
        *If min not 0 => subtract each image with it's minimum value to get range [0, x]
        img_l = img[0] - min_val_left
        img_r = img[1] - min_val_right
        *min_val = np.squeeze(min_val)
        max_val_left_u = max_val_left - min_val_left
        max_val_right_u = max_val_right - min_val_right
        * Norm
        img_l = tf.multiply(img_l, 1. / (max_val_left_u / np.sum(np.abs([r_min, r_max]))))
        img_l_n = img_l + r_min
        img_r = tf.multiply(img_r, 1. / (max_val_right_u / np.sum(np.abs([r_min, r_max]))))
        img_r_n = img_r + r_min
        imge = tf.stack([img_l_n, img_r_n])
        return imge, label
def input_from_dataset(filename="/daten/b/sceneflow_disp_train.tfrecord", kitty=False,
batch=1, buffer=1024):
    with tf.name_scope('input'):
        dataset = tf.data.TFRecordDataset(filename)
        * map takes a python function and applies it to every sample
        if kitty:
            dataset = dataset.map(decode_kitty, num_parallel_calls=4)
        else:
            dataset = dataset.map(decode_sceneflow, num_parallel_calls=4)
            dataset = dataset.shuffle(buffer)
            dataset = dataset.map(augment, num_parallel_calls=4)
            dataset = dataset.map(norm_img, num_parallel_calls=4)
        return dataset

```

## Literatur

- [1] Jure Žbontar, Yann LeCun, "Computing the Stereo Matching Cost with a Convolutional Neural Network", 10.1109/CVPR.2015.7298767 June 2015
- [2] Alex Kendall Hayk Martirosyan Saumitro Dasgupta Peter Henry, 'End-to-End Learning of Geometry and Context for Deep Stereo Regression', 1703.04309vl 13 Mar 2017
- [3] Jure Žbontar, Yann LeCun , 'Stereo Matching by Training a Convolutional Neural Network to Compare Image Patches' JMLR 17(65):1-32, 2016
- [4] [https://blog.csdn.net/nature\\_XD/article/details/69365812](https://blog.csdn.net/nature_XD/article/details/69365812) 01.02.2019
- [5] Alex Kendall Hayk Martirosyan Saumitro Dasgupta Peter Henry. "End-to-End Learning of Geometry and Context for Deep Stereo Regression", 1703.04309vl 13 Mar 2017
- [6] <https://www.sandscape.com/593.html> 02.02.2019
- [7] <https://www.pressebox.de/pressemitteilung/the-imaging-source-europe-gmbh/Der-schnelle-Einstieg-in-3D-Bildverarbeitung-IC-3D/boxid/873830>
- [8] <https://blog.csdn.net/happytofly/article/details/80124644> 03.02.2019
- [9] <http://imgtec.eetrend.com/d6-imgtec/blog/2018-12/19274.html> 04.02.2019
- [10] <https://www.zhihu.com/question/264726552> 05.02.2019
- [11] <http://www voidcn com/article/p-yxlmsirk-bpw.html>
- [12] [https://blog.csdn.net/fb\\_help/article/details/83339092](https://blog.csdn.net/fb_help/article/details/83339092) 05.02.2019
- [13] <https://blog.csdn.net/u010025211/article/details/53212331> 06.02.2019
- [14] Daniel Scharstein, Tatsunori Taniai, Sudipta N. Sinha, "Semi-Global Stereo Matching with Surface Orientation Priors", 1712.00818 [cs.CV] 3 Dec 2017
- [15] <http://www.cnblogs.com/hrlnw/p/4746170.html> 06.02.2019
- [16] Daniel Hernandez-Juarez, Alejandro Chacón, Antonio Espinosa, David Vázquez, Juan Carlos Moure, Antonio Manuel López, "Embedded real-time stereo estimation via Semi-Global Matching on the GPU", 10.1016/j.procs.2016.05.305

- [17] Heiko Hirschmuller, "Stereo Processing by Semiglobal Matching and Mutual Information", 10.1109/TPAMI.2007.1166 IEEE, 2, Feb. 2008
- [18] <http://www.cnblogs.com/hrlnw/p/4746170.html> 06.02.2019
- [19] <https://blog.csdn.net/wsj998689aa/article/details/49464017> 08.02.2019
- [20] <http://lunokhod.org/?p=1356> 08.02.2019
- [21] <https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148> 09.02.2019
- [22] <https://skymind.ai/wiki/convolutional-network> 09.02.2019
- [23] <https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148> 09.02.2019
- [24] Alex Kendall Hayk Martirosyan Saumitro Dasgupta Peter Henry, "End-to-End Learning of Geometry and Context for Deep Stereo Regression", 1703.04309vl 13 Mar 2017
- [25] <https://zh.wikipedia.org/wiki/TensorFlow> 09.02.2019
- [26] Mit Hilfe Python-code von Sceneflow-Dataset ausnehmen
- [27] <https://www.tensorflow.org/> 09.02.2019
- [28] <https://medium.com/@WuStangDan/step-by-step-tensorflow-object-detection-api-tutorial-part-1-selecting-a-model-a02b6aabe39e> 10.02.2019
- [29] Alex Kendall Hayk Martirosyan Saumitro Dasgupta Peter Henry, "End-to-End Learning of Geometry and Context for Deep Stereo Regression", 1703.04309vl 13 Mar 2017
- [30] <https://bouthilx.wordpress.com/2013/04/21/a-soft-argmax/> 09.02.2019
- [31] Deqing Sun, Xiaodong Yang, Ming-Yu Liu, Jan Kautz, 'Pwc-net CNNs for Optical Flow Using Pyramid, Warping, and Cost Volume' 1709.0213 7 Sep, 2017
- [32] Netzwerk-Flussdiagramm, das durch Aufrufen von Tensorboard erhalten wird
- [33] Alex Kendall Hayk Martirosyan Saumitro Dasgupta Peter Henry, "End-to-End Learning of Geometry and Context for Deep Stereo Regression", 1703.04309vl 13 Mar 2017

- [34] [https://www.tensorflow.org/api\\_docs/python/tf/metrics/mean\\_absolute\\_error](https://www.tensorflow.org/api_docs/python/tf/metrics/mean_absolute_error) 12 02 2019
- [35] <https://machinelearningmastery.com/learning-rate-for-deep-learning-neural-networks/> 13 02 2019
- [36] [https://www.tensorflow.org/api\\_docs/python/tf/train/exponential\\_decay](https://www.tensorflow.org/api_docs/python/tf/train/exponential_decay)
- [37] Pavel Surmenok, "Estimating an Optimal Learning Rate For a Deep Neural Network" ,Nov 12, 2017
- [38] <https://juejin.im/post/5b21dccfe51d450679257cd2> 28 02 2019
- [39] Alex Kendall Hayk Martirosyan Saumitro Dasgupta Peter Henry, "End-to-End Learning of Geometry and Context for Deep Stereo Regression", 1703.04309vl 13 Mar 2017
- [40] N. Mayer and E. Ilg and P. Häusser and P. Fischer and D. Cremers and A. Dosovitskiy and T. Brox, "A Large Dataset to Train Convolutional Networks for Disparity, Optical Flow, and Scene Flow Estimation",arXiv:1512.02134, 2016
- [41] "<http://lmb.informatik.uni-freiburg.de/Publications/2016/MIFDB16>" 04 03 2019
- [42] <https://blog.csdn.net/happytofly/article/details/80124644> 05 03 2019
- [43] <https://www.sim.informatik.tu-darmstadt.de/publ/da/2005-Klemp.pdf> 04 04 2019
- [44] <https://towardsdatascience.com/a-look-at-gradient-descent-and-rmsprop-optimizers-f77d483ef08b> 05 04 2019
- [45] [https://blog.csdn.net/tyhj\\_sf/article/details/79932893](https://blog.csdn.net/tyhj_sf/article/details/79932893) 05 04 2019

## Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel verfasst habe.

Zutreffendes bitte ankreuzen:

Mit der hochschulinternen Veröffentlichung der Arbeit bin ich:

einverstanden       nicht einverstanden

Ich bin damit einverstanden, dass

- der Titel meiner Arbeit mit meinem Name und denen der Betreuer in Bibliothekskatalog (OPAC) veröffentlicht wird und
- die Arbeit als PDF hochschulintern eingesehen werden kann.

Dafür erhält die Hochschule ein einfaches, nicht übertragbares Nutzungsrecht ausschließlich für den Zweck der Veröffentlichung in der Bibliothek. Das Recht der Veröffentlichung oder Verwertung durch den Verfasser oder die Verfasserin auf andere Weise, z.B. über einen Verlag, bleibt davon unberührt. Die Hochschule ist nicht verpflichtet, die Arbeit zu veröffentlichen. Das Einverständnis kann jederzeit schriftlich (per Brief!) widerrufen werden. Die Bibliothek wird die Arbeit dann unverzüglich aus dem OPAC oder der Auslage entfernen. Die Veröffentlichung hängt außerdem von der später erteilten Zustimmung des Erstbetreuers ab.

---

Ort, Datum

---

Unterschrift