FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

FH Aachen

Fachbereich Maschinenbau und Mechatronik

Studiengang Maschinenbau

Vertiefungsrichtung Entwicklung und Konstruktion

Bachelorarbeit

Objekterkennung und Schienenerkennung in der Schienenfahrzeugtechnik

vorgelegt von **Vu, Tuan Dat**

Matrikel-Nr. **3114978**

Referent: Prof. Dr. Raphael Pfaff

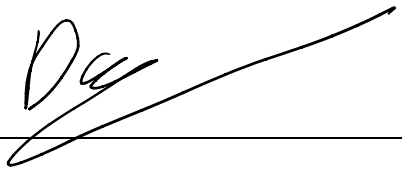
Korreferent: Matthias Blumenschein, M.Eng

Datum: 01.01.2022

Erklärung zur Urheberschaft

Ich versichere hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die im Quellenverzeichnis angegebenen Quellen benutzt habe. Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder noch nicht veröffentlichten Quellen entnommen sind, sind als solche kenntlich gemacht. Die Zeichnungen oder Abbildungen in dieser Arbeit sind von mir selbst erstellt worden oder mit einem entsprechenden Quellennachweis versehen. Diese Arbeit ist in gleicher oder ähnlicher Form noch bei keiner anderen Prüfungsbehörde eingereicht worden.

Aachen, den 22.12.2021



Vu, Tuan Dat

Danksagung

Bevor wir uns dem eigentlichen Thema dieser Bachelorarbeit zuwenden, möchte ich sagen, wie sehr ich mich freue, das Projekt abschließen zu können. Ich habe immer Spaß am Programmieren und programmierbezogene Themen in meinem Projekt bearbeiten zu können, ist so eine großartige Erfahrung.

Ich bedanke mich bei meinem Betreuer, Prof. Raphael Pfaff, für die Möglichkeit, dieses sehr interessante Projektthema zu bearbeiten. Und ich möchte ihm auch dafür danken, dass er mir die Ausrüstung für das Projekt zur Verfügung gestellt hat. Ohne die Hilfe meines anderen Betreuers Matthias Blumenschein wäre diese Reise nicht möglich gewesen. Er hat mich vom ersten bis zum letzten Tag unterstützt, wofür ich ihm ewig dankbar bin. Abschließend möchte ich mich bei einigen meiner Freunde in Deutschland bedanken, die mich bei meiner Recherche oder Arbeit von zu Hause aus beraten und begleitet haben.

Inhaltverzeichnis

Erklärung zur Urheberschaft **2**

Danksagung **3**

Inhaltverzeichnis **4**

1 Einführung **7**

1.1 Hintergrund 7

1.2 Motivation 8

1.3 Ziel 9

1.4 Gesamtkonzept 10

2 Grundlagen **11**

2.1 Objekterkennung 11
2.1.1 Allgemeine Merkmale der Objekterkennung 11
2.1.2 Modelle zur Objekterkennung in der Praxis 12

2.2 Jetson Nano 12

2.3 OpenCV 14

2.4 Python 16

2.5 Jupyter Notebook 16

2.6 Neuronale Netze 17
2.6.1 Aktivierungsfunktion 18
2.6.2 Verlustfunktion 21
2.6.3 Aktualisierung der Parameters im Lauf des Netzes 22

3 Methodik und Umsetzung **26**

Teil 1: Objekterkennung **26**

1 MobileNetV2 26
1.1 Faltungsformen 27
1.1.1 Konventionelle zweidimensionale Faltung 27
1.1.2 Tiefenweise trennbare Faltung (Depthwise Separable Convolution) 28
1.1.3 Vergleich von zwei Faltungsformen 30
1.1.3.1 Anzahl der zu berechneten Parameter 30
1.1.3.2 Anzahl der auszuführenden Operationen 31
1.2 Invertierter Restblock 31

2 SSD-Netz 33
2.1 Verlauf des SSDs 33
2.2 Architektur von den SSD-Netz auf Basis des MobileNetV2 34

3 Implementierung 36

Teil 2: Gleisbahnerkennung 40

1 Kamera kalibrieren 40
1.1 Der Grund des Kamerakalibrieren 40
1.2 Definition von Begriffen 41
1.3 Prozess 42

2 Gaußscher Unschärfe-Algorithmus und morphologische Transformationen 43

3 Canny Kantenerkennung 45

4 Prozess des Verzerrens der Bildperspektive 49

5 Histogram 51

6 Algorithmus 51
6.1 Grundidee 51
6.2 Parameter anpassen 52
6.3 Scanprozess 53
6.4 Verarbeitung erfasster Pixel, Kurvenanpassung 55

7 Endergebnis 57

4 Bewertung und Ausblick 59

Abkürzungsverzeichnis 60

Abbildungsverzeichnis 61

Tabellenverzeichnis 62

Literaturverzeichnis 63

Anhang A: Objekterkennnung 65

A.1 Docker ausführen 65

A.2 Prüfung der Voraussetzungen 65

A.3 Die Daten herunterladen 65

A.4 Training des Modells 65

A.5 Konvertierung des Modells in ONNX 65

Anhang B: Gleisbahnerkennung 66

B.1 Importieren die benötigten Bibliothek 66

B.2 Kamera kalibrieren 66

B.3 Gaußscher Unschärfe-Algorithmus und morphologische Transformationen 67

B.4 Perspective Warping 67

B.5 get_hist() -Funktion 68
B.6 Schiebefernster 68
B.7 Plotten 72

1 Einführung

1.1 Hintergrund

Im Zeitalter von Industrie 4.0 konzentriert die Menschheit sich auf die Entwicklung neuer Technologien mit dem Ziel, menschliche Eingriffe so gering wie möglich zu halten. Darüber hinaus ist auch die Anwendung dieser neuen Technologien auf bestehende Technologien sehr auffällig. Eine der großen Erfindungen der Menschheit ist die Eisenbahn und der Zug. Auch in der heutigen Zeit ist die Eisenbahnindustrie für den Personen- und Güterverkehr von klein wie in der Stadt bis hin zu groß wie auf dem Land noch immer äußerst notwendig. Mit der Entwicklung der Technologie ist für die Bahnindustrie eine Verbesserung der Sicherheit zu erwarten. Einige Beispiele für die Anwendung der Technologie in anderen Branchen der Mobilität sind selbstfahrende Autos und Drohnen.

Das Ziel dieser neuen Technologien ist nicht nur Steigerung der Sicherheit, sondern auch die Einsparung von Personal und Energie. Ein Beispiel für eine Übung, die in diesem Bereich benötigt wird, ist das Rückwärtsfahren eines Zuges beim Einfahren in einen Rangierbahnhof. Die traditionelle Methode ist, dass an beiden Enden des Zuges je eine Personen benötigt. Eine Person informiert den Lokführer am anderen Ende darüber, wo sich der Zug befindet, ob es sicher ist, sich zu bewegen.

Dadurch kann der Lokführer den Zug sicher umkehren. Das Einrichten einer Kamera ist hier möglich, jedoch nicht mit einer herkömmlichen Kamera. Der Grund ist, dass eine herkömmliche Kamera den Menschen nicht ersetzen kann, da dieser neben seiner Sicht auch viele andere Faktoren erkennen muss, wie etwa die Erkennung von Objekten in der Nähe des Zuges oder den Zustand der Gleise. Mit Hilfe von Software und Erkennungsalgorithmen sowie verfügbarer Hardware wie Kameras wurde das Problem jedoch gelöst. Obwohl noch viele Faktoren wie Leistung, Systemhaltbarkeit und Optimierung der Verarbeitungsgeschwindigkeit überwunden werden müssen, hat im Grunde die grundlegende Richtung den Nutzen von KI und Computer Vision bewiesen.

1.2 Motivation

Die Projektarbeit ist so konzipiert, dass sie mehrere Probleme der traditionellen analytischen Bildverarbeitung lösen wird. Erstens muss der Zug die Position der Gleise während der Fahrt bestimmen und das Objekt (in diesem Fall eine Person) erkennen, um eine Vorhersage treffen zu können und die optimale Lösung für die Situation zu wählen. Hier bietet sich der Einsatz von Computer Vision über Kameras an, da sie dem Zugführer einen besseren Überblick über das Geschehen und den Zustand des Fahrzeugs zu diesem Zeitpunkt verschaffen können. Mittels ausgewählter Algorithmen kann die Bewegung der Lokomotive sowie der verfolgten Objekte gespeichert, visualisiert und eventuell mit Hilfe von künstlicher Intelligenz eine Lösung für die Situation ermittelt oder vorgeschlagen werden. In dieser Arbeit wird eine Methode vorgestellt, die ausschließlich analytische Algorithmen verwendet, um die Schienen erkennen zu können.

Mit der Entwicklung der Technologie im Allgemeinen und der Informatik im Besonderen steht jedoch neue und modernere Werkzeuge zur Lösung dieses Problems zur Verfügung. Daher wird in dieser Arbeit eine andere Herangehensweise an das Problem der Objekterkennung vorgestellt, bei der ein faltbares neuronales Netz verwendet wird.

Das Thema der Fahrspurerkennung im Landverkehr hat in den letzten zwanzig Jahren viel Aufmerksamkeit erregt. Trotz des gleichen Ziels der verschiedenen Verkehrssysteme, nämlich der Erkennung und Verfolgung von Fahrbahnen oder Straßenlinien, gibt es immer noch einige Unterschiede und Schwierigkeiten bei der Herangehensweise und wird in Tabelle 1 aufgelistet.

Straße	Gleisbahn
Variable Fahrbahnbreite	Feste Fahrbahnbreite
Variable Seitenverschiebung	Keine Seitenverschiebung
Unterschiedliche Arten von Fahrbahnmarkierungen	Feste Fahrbahnmarkierungen
Das allgemeine Erscheinungsbild der Fahrbahn ist relativ homogen	Mehrere verschiedene (inhomogene) Fahrspuren
Fahrbahnmarkierungen sind so konzipiert, dass optimale Sichtbarkeit	Die Sichtbarkeit ist nicht garantiert
Fahrbahnmarkierungen haben kein Volumen und werfen daher keine Schatten auf den Boden	Gleise haben eine bestimmte Höhe und werfen daher werfen Schatten auf den Boden
Baustellen und Hindernisse können leicht verändern den Weg eines Fahrzeugs auf einer Straße	Weg des Fahrzeugs durch die Welt das Koordinatensystem ist festgelegt
Die Geschwindigkeit eines Fahrzeugs wird im Allgemeinen den Wetter- und Sichtverhältnissen angepasst	Automatische Züge fahren mit nahezu konstanter Geschwindigkeit unabhängig von den meisten Wetterbedingungen
Die horizontale Bewegung einer fahrzeugmontierten Kamera ist aufgrund der geringen Höhe des Fahrzeugs	Schwingen des Zuges verursacht erhebliche Änderung der Kameraposition entlang des der Strecke

Tabelle 1: Eigenschaften von Fahrbahnen und Gleisen [1]

1.3 Ziel

Ziel des Projekts ist es, einen analytischen Algorithmus zu programmieren, der die Aufgabe übernimmt, die Schiene per Kamera auf einem kleinen Gerät wie Jetson Nano zu erkennen. Eine weitere Aufgabe ist die Anwendung und Durchführung eines Algorithmus zur Objekterkennung, um Fußgänger und Landfahrzeuge in Sichtweite des Zuges zu erkennen. Da die Fußgängererkennung ein vielfach gelöstes Problem darstellt, konnte ein vortrainierter Algorithmus verwendet werden.

1.4 Gesamtkonzept

Der Ansatz dieses Projekts besteht darin, das Problem zunächst anhand von Beispielen von Standbildern, die aus tatsächlichen Videos geschnitten wurden, zu behandeln und zu entwickeln. Tatsächlich wird es bei installierten herkömmlichen Kameras eine Bildrate von etwa 30 fps (frame per second) oder 30 Bildern pro Sekunde geben. Die Erwartung des Projekts ist, dass die Skalierung des „Algorithmus" auf den Videomaßstab nach erfolgreicher Anwendung auf Standbilder viel einfacher zu handhaben sein wird.

Bei der Handhabung von Standbildobjekten werden zunächst Bildverarbeitungsalgorithmen und dann ein projektspezifischer Algorithmus angewendet. Die Bildverarbeitungsalgorithmen in der Vorverarbeitung sind allesamt Algorithmen, die in den Bibliotheken der Programmiersprache Python bereits zu finden sind. Der Grund, diese Algorithmen weder selbst zu entwickeln noch neu zu erfinden, ist folgender: Diese eingebauten Algorithmen sind alle für die schnellste Ausführung optimiert, die Variablen, die wir ändern können, reichen für dieses Projekt aus. Es ist also nicht notwendig, den Algorithmus selbst zu ändern oder zu entwickeln. Der Unterschied der Vorverarbeitung liegt hier in der Reihenfolge, in der die Algorithmen und den gewählten Parametern ausgeführt werden. Das Ziel der Vorverarbeitung ist es, ein gewünschtes Produkt mit minimalem Verlust herzustellen. Ein Beispiel für den Prozessablauf wird in Abbildung 1 verwiesen und erläutert.

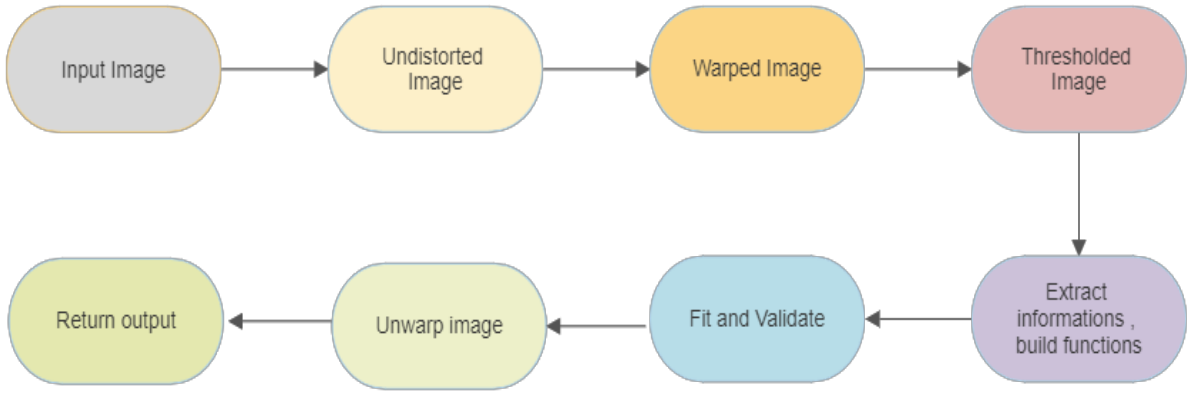


Abbildung 1: Ausführungsreihenfolge für die Gleisbahnerkennung

2 Grundlagen

In diesem Kapitel werden die grundlegenden Kenntnisse und Begriffe, die zum Verständnis der Arbeit notwendig sind, behandelt. Dazu gehören die Erklärung und das Verständnis einiger Begriffe sowie verschiedene Algorithmen, die in dieser Arbeit häufig verwendet werden.

2.1 Objekterkennung

2.1.1 Allgemeine Merkmale der Objekterkennung

Objekterkennung ist ein allgemeiner Begriff, der einige verwandte Computer-Vision-Aufgaben im Zusammenhang mit der Identifizierung von Objekten in digitalen Bildern beschreibt. Bildklassifizierung ist die Vorhersage der Klasse von Objekten in einem Bild. Objektplatzierung ist die Positionierung eines oder mehrerer Objekte in einem Bild und das Zeichnen eines Begrenzungsrahmens um diese Objekte. Die Objekterkennung kombiniert die beiden oben genannten Aufgaben, um ein oder mehrere Objekte in einem Bild zu finden. Diese drei grundlegenden Bildverarbeitungsaufgaben lassen sich anhand von Ein- und Ausgängen wie folgt unterscheiden:

- Bildklassifizierung: Sagt das Label von Objekten im Bild voraus. Dem Bild wird ein Label zugewiesen.
  - Eingabe: Bild mit Objekt (Beispiel: Foto)
  - Ausgabe: Klassenspezifikation (Beispiel: Eine oder mehrere dem Klassennamen zugewiesene Ganzzahlen).
- Objektplatzierung: Bestimmt das Vorhandensein von Objekten im Bild und verwendet Begrenzungsrahmen, um ihre Position anzugeben.
  - Eingabe: Ein Bild mit einem oder mehreren Objekten (z. B. ein Foto).
  - Ausgabe: Ein oder mehrere Begrenzungsrahmen, die durch Mittelpunkt-, Breiten- und Höhenkoordinaten definiert sind.
- Objekterkennung: Ermittelt das Vorhandensein von Objekten im Begrenzungsrahmen und die Beschriftung von Objekten im Bild.
  - Eingabe: Ein Bild mit einem oder mehreren Objekten (z. B. Foto.
  - Ausgabe: Ein oder mehrere Begrenzungsrahmen und die Beschriftung jedes Begrenzungsrahmens.

Ein weiterer Unterschied zwischen Bildklassifizierung und Objekterkennung besteht darin, dass das Bildklassifizierungsmodell eine Verlustfunktion hat, die nur auf dem Fehler zwischen dem vorhergesagten Label und dem tatsächlichen Label basiert, während die Objekterkennung den Fehler zwischen dem vorhergesagten Label und dem tatsächlichen Label bewertet.

2.1.2 Modelle zur Objekterkennung in der Praxis

In den letzten Jahren hat die starke Entwicklung der Computer Vision sowie die Zunahme der Leistungsfähigkeit der Computer-Hardware dazu beigetragen, viele neue Algorithmen und Ansätze für den Bereich der Objekterkennung zu entwickeln, von denen einige berühmte Algorithmen wie YOLO, ResNet, SSD-mobileNet-v1, SSD-mobileNet-v2 genannt werden können. Diese Algorithmen basieren auf Frameworks wie TensorFlow oder Pytorch und sind als neuronale Netze mit einer oder mehreren Schichten aufgebaut. Einige Definitionen und Konzepte von neuronalen Netzen werden in Teil 2 erläutert. Je nach dem tatsächlichen Verwendungszweck werden verschiedene algorithmische Architekturen ausgewählt und verwendet.

Mögliche Faktoren bei der Auswahl eines Algorithmus sind: Leistung auf einer Testmenge, Leistung auf einem Gerät, wie einfach ist es, den Algorithmus auf dem Gerät einzusetzen, Rechenkosten des Algorithmus. In dieser Bachelorarbeit wird der Algorithmus SSD-MobileNetv2 ausgewählt und verwendet. Der Grund für die Wahl von SSD-MobileNetV2 liegt in der Kombination einzigartiger Technik, die sich perfekt für den Einsatz auf kleinen Geräten wie dem Jetson Nano eignet. Diese Techniken und ihre Kombination werden in Kapitel 3 Teil 1 erläutert. SSD-MobileNet ist eine sehr bekannte Netzwerkarchitektur auf dem Gebiet der Echtzeit-Objekterkennung auf mobilen und integrierten Geräten. Sie kombiniert den SSD-300 Single-Shot MultiBox Detektor mit einem Mobilenet als Basisnetz.

2.2 Jetson Nano

NVIDIA Jetson ist eine Reihe fortschrittlicher eingebetteten Systeme, die von Entwicklern zum Erstellen innovativer KI-Produkte in verschiedenen Branchen verwendet werden. Da Jetson eine der weltweit führenden Hardwareplattformen für KI am Edge ist, ist es für Studenten und Technikbegeisterte gleichermaßen nützlich, durch eine Reihe kreativer Projekte praktische Erfahrungen mit KI zu sammeln. Die Plattform besteht

aus Hochleistungs-Edge-Computern mit kleinem Formfaktor, die als Module bezeichnet werden. Darüber hinaus enthält es auch das JetPack SDK für die Softwarebeschleunigung und ein komplettes Ökosystem, um den Entwicklungsprozess von benutzerdefinierten KI-Projekten zu beschleunigen. Die Hardware, die in dieser Arbeit zum Trainieren und Bereitstellen dieser Objekterkennungsaufgabe verwendet wird, heißt Jetson Nano 2GB Developer Kit. Der Chiphersteller NVIDIA konzentriert sich seit langem darauf, Geräte speziell für das Lernen und Arbeiten im Bereich des maschinellen Lernens zu entwickeln und herzustellen. Das Jetson Nano 2GB Developer Kit ist aufgrund seiner Kompaktheit und Rechenleistung ein ideales Gerät für praktische Projekte.

Um zu verdeutlichen, wie geeignet der Jetson Nano ist, werden auch einige Hauptmerkmale des Raspberry Pi aufgeführt. Die meisten der Hauptmerkmale der beiden berühmtesten Einplatinencomputer sind ähnlich, aber was den Jetson Nano zur besseren Wahl macht, liegt in seinen Grafikfähigkeiten, insbesondere in seinen grafischen Verarbeitungseinheiten (Graphic processing units oder GPU). GPUs gibt es schon lange, seit den frühen Spieleanwendungen in den 1970er Jahren. Später wurden GPUs verwendet, um geometrische Berechnungen beim maschinellen Lernen zu beschleunigen. Heutzutage werden GPUs im maschinellen Lernen aufgrund ihrer Parallelverarbeitungsfähigkeit geschätzt.

Laut Oracle sind GPUs eine sicherere Wahl für schnelles maschinelles Lernen [2].Dies liegt daran, dass das Data-Science-Modelltraining im Kern aus einfachen mathematischen Matrixberechnungen besteht. Wenn diese Berechnungen parallel ausgeführt werden können, kann daher die Geschwindigkeit dieser Berechnungen erheblich verbessert werden. Der NVIDIA Jetson Nano verfügt über eine 128-Kern-Maxwell-GPU mit 921MHz.

Im Vergleich verfügt der Jetson Nano über eine viel leistungsfähigere GPU als der Raspberry Pi 4. Dadurch eignet sich der Jetson Nano besser für KI- und ML-Anwendungen, was ein besonderer Vorteil sein kann, wenn er für den beabsichtigten Verwendungszweck angepasst wird.

	Jetson Nano 2GB	Raspberry Pi 4
GPU	128-core NVIDIA Maxwell™	Broadcom BCM2711
CPU	Quad-core ARM® A57 @ 1.43 GHz	Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz
Memory	2 GB 64-bit LPDDR4 25.6 GB/s	2GB, 4GB or 8GB LPDDR4-3200 SDRAM
Connectivity	Gigabit Ethernet, 802.11ac wireless	Gigabit Ethernet, 802.11ac wireless

Tabelle 2: Merkmale von Jetson Nano 2GB und Raspberry Pi 4 [3] [4]

2.3 OpenCV

Das Projekt OpenCV wurde 1999 von Gary Bradsky bei Intel ins Leben gerufen [5]. OpenCV steht für Open Source Computer Vision Library. OpenCV ist die führende Open-Source-Bibliothek für Computer Vision und maschinelles Lernen und bietet jetzt auch GPU-Beschleunigung für Echtzeitoperationen an. OpenCV wird unter der BSD-Lizenz veröffentlicht, ist also sowohl für Lernzwecke als auch für die kommerzielle Nutzung frei. Es ist mit C++-, C-, Python- und Java-Schnittstellen verfügbar und unterstützt Windows, Linux, Mac OS, iOS und Android. OpenCV wurde entwickelt, um die Effizienz von Berechnungen zu unterstützen und ist auf Echtzeitanwendungen spezialisiert. Wenn diese Bibliothek in optimalem C/C++ geschrieben ist, kann sie die Vorteile der Multicore-Verarbeitung nutzen.

OpenCV wird für eine Vielzahl von Anwendungen verwendet, wie zum Beispiel:

- Bilder von Straßenansichten
- Automatisierte Tests und Überwachung
- Roboter und selbstfahrende Autos
- Medizinische Bildanalyse, Suchen und Wiederherstellen von Fotos/Videos
- Interaktive Kunstinstallation

Je nach den Funktionen und Anwendungen von OpenCV kann diese Bibliothek in folgende Gruppen mit entsprechenden Funktionen und Modulen unterteilt werden:

- Bild/Video/I/O-Verarbeitung und Anzeige (core, imgproc, highgui)
- Erkennen von Objekten (objdetect, features2d, nonfree)
- Geometriebasiertes monokulares oder Stereo-Computersehen (calib3d, stitching, videostab)
- Computergestützte Fotografie (photo, video, superres)
- Maschinelles Lernen und Clustering (ml, flann)
- CUDA-Beschleunigung (gpu)

OpenCV hat eine modulare Struktur, was bedeutet, dass das Paket eine Reihe von statischen Bibliotheken oder dynamisch gelinkten Bibliotheken (shared libraries) enthält. Nachfolgend sind einige detaillierte Definitionen der gängigen Module aufgeführt:

- Kernfunktionalität (core) - kompaktes Modul zur Definition grundlegender Datenstrukturen, einschließlich dichter mehrdimensionaler Arrays und vieler grundlegender Funktionen, die von allen anderen Modulen verwendet werden.
- Bildverarbeitung (imgproc) - Bildverarbeitungsmodul einschließlich linearer und nichtlinearer Bildfilterung, Geometrietransformationen (Größe, Affin- und Warp-Perspektive, Neuzuordnung auf der Grundlage der globalen Tabelle), Farbraumkonvertierungen, Histogramme und mehr.
- Videoanalyse (Video) - Videoanalysemodul, das Algorithmen zur Bewegungsabschätzung, Hintergrundtrennung und Objektverfolgung umfasst.
- Kamerakalibrierung und 3D-Rekonstruktion (calib3d) - grundlegende mehrdimensionale Geometriealgorithmen, Einzel- und Stereokamerakalibrierung (Einzel- und Stereokamerakalibrierung), Objektposenschätzung, Korrespondenzalgorithmen, Stereokorrespondenzalgorithmen und 3D-Rekonstruktionselemente.
- 2D Features Framework (features2d) - erkennt hervorstechende Merkmale von Identifikatoren, Parameterabfragen und gegnerische Parameter.
- Objekterkennung (objdetect) - erkennt Objekte und simuliert vordefinierte Funktionen - vordefinierte Klassen (z.B. Gesichter, Augen, Tassen, Menschen, Autos, ...).
- High-Level-GUI (highgui) - eine einfach zu verwendende Schnittstelle zur Implementierung einfacher UI-Kommunikation.

- Video I/O (videoio) - eine einfach zu bedienende Schnittstelle für die Erfassung und Codierung von Video.
- GPU - Algorithmen zur GPU-Beschleunigung aus anderen OpenCV-Modulen.
- und einige andere unterstützende Module, wie FLANN und Google Test Wrapper, Python Bindung.

2.4 Python

Da KI-Projekte oft sehr komplex sind und viele Schritte und Implementierungszeit erfordern, ist eine der wichtigsten Anforderungen an eine Programmiersprache. Die verfügbare Quelle für unterstützende Tools und Bibliotheken, die in einem KI-Projekt verwendet wird, müssen flexibel, stabil und fähig sein. Wegen ihrer Flexibilität und ihrer Einfachheit bei der Anwendung ist Python eine sehr beliebte Programmiersprache in KI-Bereich. Es bietet viele Vorteile an, wie zum Beispiel

- Weniger Code - Einfachheit und Konsistenz
- Vorgefertigte Bibliotheken - Große Gemeinschaft und Quellcode/Bibliothekssystem
- Plattformunabhängig
- Unterstützung der Gemeinschaft

2.5 Jupyter Notebook

Jupyter ist ein kostenloses und quelloffenes Tool für Datenwissenschaft und Bildung, das es jedem leicht macht, gemeinsam Python zu lernen. Jupyter ist interaktiv und kann daher als Test- und Lehrumgebung verwendet werden.

Der Name Jupyter stammt aus einem Wortspiel „Ich bin Python, du bist R, wir sind Julia" [6], oder einer Kurzformel Jupyter = Julia + Python + R. Zuvor entstand Jupyter aus dem IPython-Projekt, bis es 2014 in Jupyter umbenannt wurde. Jupyter unterstützt viele Kernel für verschiedene Sprachen, über 40 Sprachen einschließlich Python. Die Namensänderung von IPython zu Jupyter ist auch der mehrsprachigen Unterstützung geschuldet.

Das Jupyter-Notebook ist ein Werkzeug, mit dem man sowohl Python-Code als auch komplexe Textelemente wie Bilder, Formeln, Videos, Ausdrücke in dieselbe Datei einfügen kann, um die Präsentation leichter verständlich zu machen, ähnlich wie eine Präsentationsdatei, in der Sie

aber auch interaktiven Code ausführen können. Diese Notizbuchdateien können mit allen geteilt werden und lassen sich schnell und genauso nachbearbeiten, wie der Autor dieser Arbeit sie erstellt hat.

2.6 Neuronale Netze

Neuronale Netze sind eine Kombination von Schichten von Perceptrons oder Multilayer-Perceptrons. Jedes neuronale Netz umfasst in der Regel 3 Arten von Schichten:

- Eingabeschicht (Eingabeschicht): Diese Schicht befindet sich auf der linken Seite des Netzes und stellt die Eingänge des Netzes dar.
- Ausgangsschicht (output layer): Diese Schicht befindet sich ganz rechts und stellt die Ausgänge des Netzes dar.
- Die versteckte Schicht (hidden layer): Diese Schicht befindet sich zwischen der Eingabe- und der Ausgabeschicht und stellt den logischen Inferenzprozess des Netzes dar.

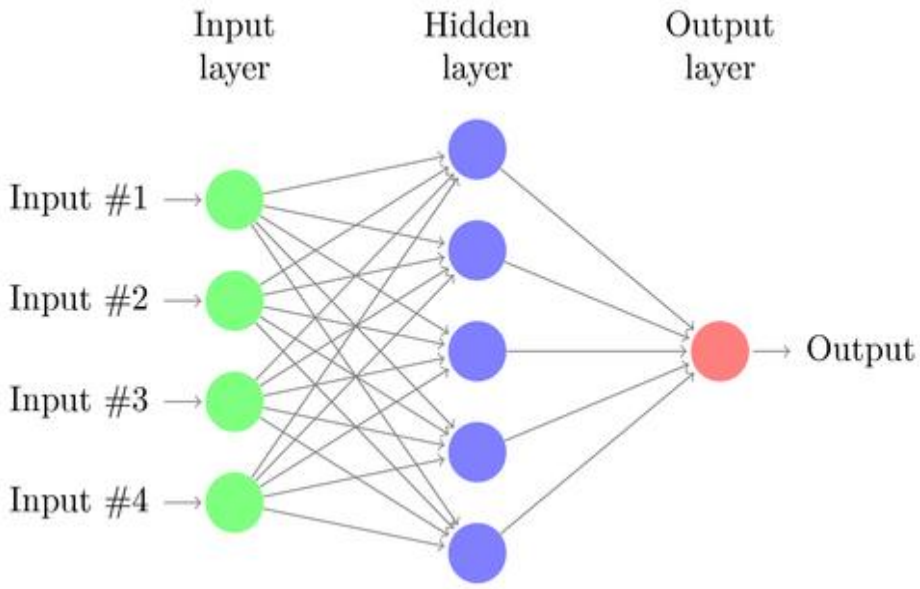


Abbildung 2: Darstellung von einem einfachen neuralen Netz [16]

Hinweis: Jedes neuronale Netz hat nur eine Eingabeschicht und eine Ausgabeschicht, aber es kann viele verborgene Schichten geben. Ein kreisförmiger Knoten in einer Schicht wird als Einheit oder Neuron bezeichnet. Einheiten in den Eingabeschichten, versteckten Schichten und Ausgabeschichten werden als Eingabeeinheiten, versteckte Einheiten bzw. Ausgabeeinheiten bezeichnet.

Jede Ausgabe einer Einheit (mit Ausnahme der Eingabeeinheiten) wird mit dieser Formel berechnet:

$$a_i^l = f(w_i^{(l)T} * a^{(l-1)} + b^{(l)})$$

wobei f(.) die Aktivierungsfunktion, $w_i^{(l)}$ und $b^{(l)}$ sind die Parameter von Einheit i von der Schicht (l).

2.6.1 Aktivierungsfunktion

In jeder Einheit gibt es eine Aktivierungsfunktion. Es stehen viele Aktivierungsfunktionen zur Auswahl, aber in der Praxis werden meist vier davon verwendet.

- Sigmoid Aktivierungsfunktion

Sigmoid ist eine Aktivierungsfunktion, bei der die Ausgabe sigma auf der Grundlage der Eingabe z nach der folgenden Formel berechnet wird:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Wenn die Eingabe groß ist, ergibt die Funktion eine Ausgabe nahe bei 1. Bei einer kleinen Eingabe (sehr negativ) wird die Funktion eine Ausgabe nahe 0 liefern. Diese Funktion wurde in der Vergangenheit häufig verwendet, da sie eine sehr schöne Ableitung hat. In den letzten Jahren wird diese Funktion nur noch selten verwendet, da es einen grundlegenden Nachteil hat. Ein bemerkenswerter Nachteil ist, dass die Steigung dieser Funktion sehr nahe bei 0 liegt, wenn die Eingabe einen großen absoluten Wert hat (sehr negativ oder sehr positiv). Das bedeutet, dass die Koeffizienten, die der betrachteten Einheit entsprechen, fast nicht aktualisiert werden.

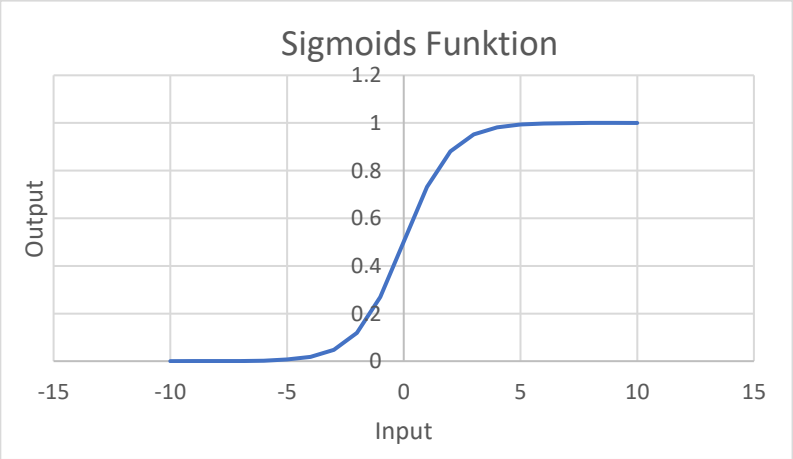


Abbildung 3: Darstellung von Sigmoids Funktion

- Hyperbolische Tangens-Aktivierungsfunktion

Die hyperbolische Tangens-Aktivierungsfunktion wird auch einfach als Tanh-Funktion (auch "tanh" und "TanH") bezeichnet.

Sie und die sigmoidale Aktivierungsfunktion sind sehr ähnlich und haben sogar die gleiche S-Form. Die Funktion nimmt einen beliebigen reellen Wert als Eingabe an und gibt Werte im Bereich von -1 bis 1 aus. Je größer die Eingabe (positiver), desto näher liegt der Ausgabewert bei 1.0, je kleiner die Eingabe (negativer), desto näher liegt die Ausgabe bei -1.0.

Die Tanh-Aktivierungsfunktion wird wie folgt berechnet:

$$tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

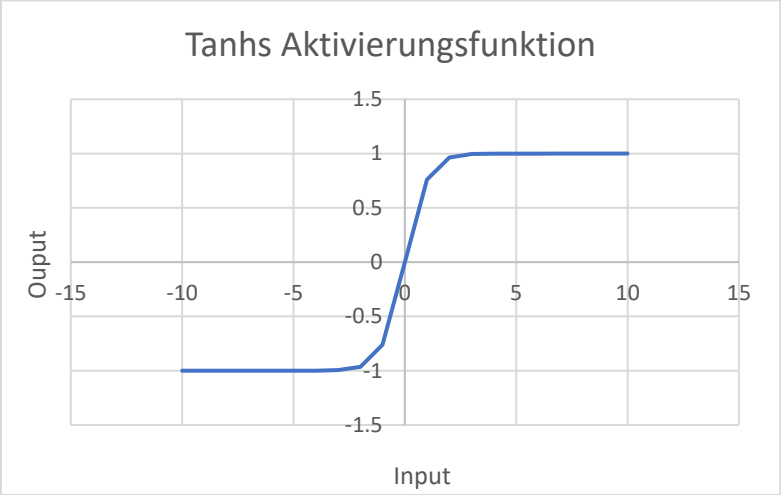


Abbildung 4: Darstellung der Tanhs Aktivierungsfunktion

- Rectified Linear Unit

ReLU oder Rectified Linear Unit ist eine sehr einfache Funktion und wird häufig wegen ihrer sehr schnellen Konvergenzgeschwindigkeit verwendet. Sie hat die folgende mathematische Formel $g(z) = max(0, z)$

Hinweis: Der Wert von z ist nie 0, sondern wird durch die Konvention ein Wert nahe 0 sein:

z = 10^{-e} mit e ist ein sehr große Wert.

Laut Krizhevsky *et al* macht ReLU das Training von Deep Networks nachweislich viel schneller [7]. Diese Beschleunigung ist darauf zurückzuführen, dass die ReLU fast augenblicklich berechnet wird und ihr

Gradient ebenfalls extrem schnell berechnet wird, mit einem Gradienten von 1, wenn die Eingabe größer als Null ist, und 0, wenn die Eingabe kleiner als Null ist.

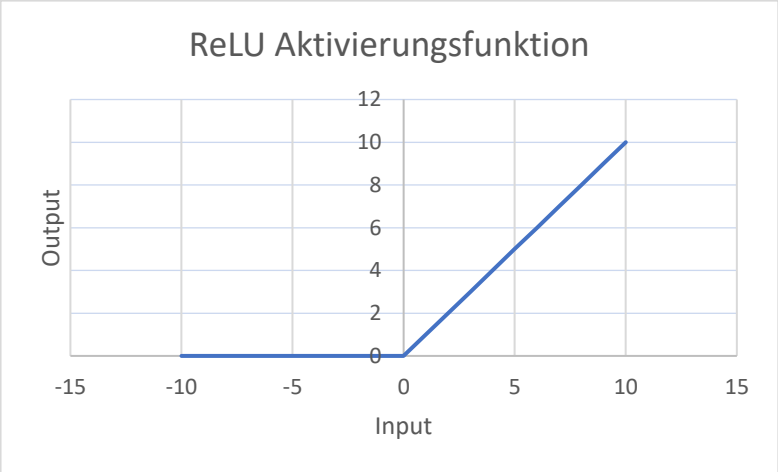


Abbildung 5: Darstellung von ReLUs Aktivierungsfunktion

- Leaky ReLU oder parametrische ReLU

Im modernen Deep Learning gibt es eine neue und sehr häufig verwendete Gleichung, nämlich Leaky ReLU. Im Grunde genommen ist Leaky ReLU eine Variante von ReLU. Der einzige Unterschied liegt in der Höhe des Wertes, der mit dem Wert von z verglichen wird. Die mathematische Formel von Leaky ReLU lautet $g(z) = max(a * z, z)$ wobei a eine normalerweise kleine Variable ist wie zum Beispiel 0.001.

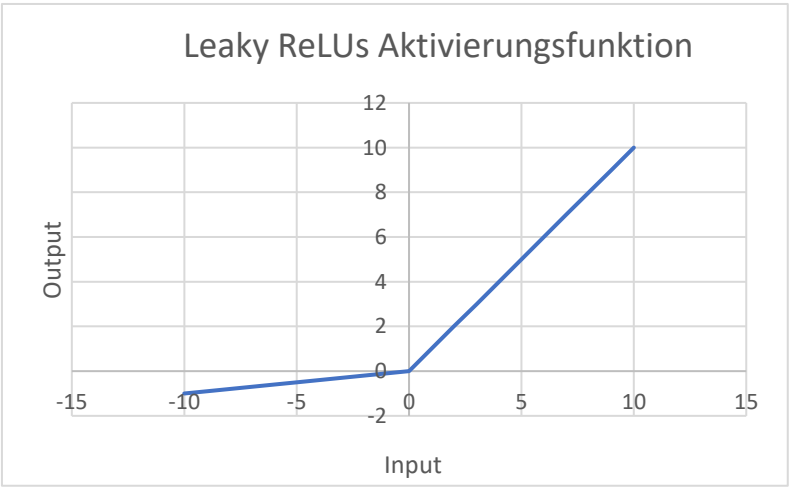


Abbildung 6: Darstellung von Leaky ReLUs Aktivierungsfunktion für a = 0.1

Obwohl es einige Studien gibt, die gezeigt haben, dass in einigen spezifischen Fällen die Verwendung von Leaky ReLU immer noch optimaler ist als ReLU, wie in der Studie von [8], wird es in der Tat vollständig auf der Grundlage der Merkmale sowie die unterschiedlichen Anforderungen der einzelnen Projekte ausgewählt.

2.6.2 Verlustfunktion

Die Verlustfunktion ist eine Funktion, mit der der Fehler zwischen dem vorhergesagten Wert und der verfügbaren richtigen Antwort gemessen wird. Die Verlustfunktion kann wie folgt dargestellt werden:

$$\mathcal{L}(\hat{y}, y) = | \hat{y} - y |$$

wobei $\hat{y}$ der vorhergesagte W und y die verfügbare richtige Antwort sind.

Häufiger wird jedoch die Größe verwendet, mit der sich die Genauigkeit des Modells messen lässt, nämlich die Kostenfunktion mit der folgenden Formel:

$$J(w^{[l]}, w^{[l-1]}, b^{[l]}, b^{[l-1]}) = \frac{1}{2m} * \sum_{i=1}^{m} (\hat{y}^{(i)} - y^{(i)})^2$$

wobei m die Anzahl der Objekte im Datensatz ist.

Man kann die Formel so umwandeln:

$$J\left(w^{[l]}, w^{[l-1]}, b^{[l]}, b^{[l-1]}\right) = \frac{1}{2m} * \sum_{i=1}^{m} \left(\hat{y}^{(i)} - y^{(i)}\right)^2 = \frac{1}{2m} * || X * w - y ||^2$$

wobei $w^{[l]}, w^{[l-1]}, b^{[l]}, b^{[l-1]}$ die Parameter von den Schichten (l) und (l-1) sind.

Der Prozess des maschinellen Lernens kann wie folgt vereinfacht werden: Optimierung der Kostenfunktion auf den Mindestwert, idealerweise Null. Erreicht eine Funktion einen lokalen Minimalwert, so ist die Ableitung dieser Funktion gleich Null. Mit anderen Worten: Der Computer wird in eine Schleife programmiert, in der die Fehlerquote mit der Anzahl der Lernvorgänge abnimmt. Das bedeutet auch, dass die Parameter nach jedem Zyklus repariert und aktualisiert werden.

2.6.3 Aktualisierung der Parameters im Lauf des Netzes

Betrachtet man die Funktion y = f(x) und das Ziel ist es, den Minimalpunkt der Funktion zu finden.

Ausgehend von einem beliebigen Punkt x muss x so angepasst werden, dass es zu dem Punkt tendiert, an dem die Funktion ihr Minimum erreicht. Dies kann durch sukzessive Wiederholung der Transformation erreicht werden

$$x := x - a * f'(x)$$

mit der aus der Pascal-Sprache entlehnten Notation := bedeutet, dass der Wert der rechten Seite der Variablen auf der linken Seite zugewiesen wird. In der obigen Transformation wird α als Lernrate bezeichnet. Je größer die Lernrate ist, desto größer ist jeder "Sprung" von x, und die Anzahl der Iterationen, die erforderlich sind, um das Minimum zu finden, nimmt ab. Wenn die Lernrate jedoch zu groß ist, ist es möglich, dass sich x nach jeder Iteration weiter vom Minimalpunkt entfernt und der Minimalpunkt nicht gefunden werden kann. Aus diesem Grund ist die Wahl der richtigen Lernrate sehr wichtig. Man kann viele verschiedene Lernratenwerte ausprobieren, um einen ausreichenden (aber nicht unbedingt den besten) Lernratenwert zu finden.

Der oben verwendete Algorithmus ist der Gradientenabstiegsalgorithmus für Funktionen mit einer Variablen. In einem neuronalen Netz mit Variablen als Gewichte und Verzerrungen wird der Algorithmus in den folgenden Schritten angewendet:

Beispiel Verlauf von einem einfachen neutralen 1-Schicht-Netz

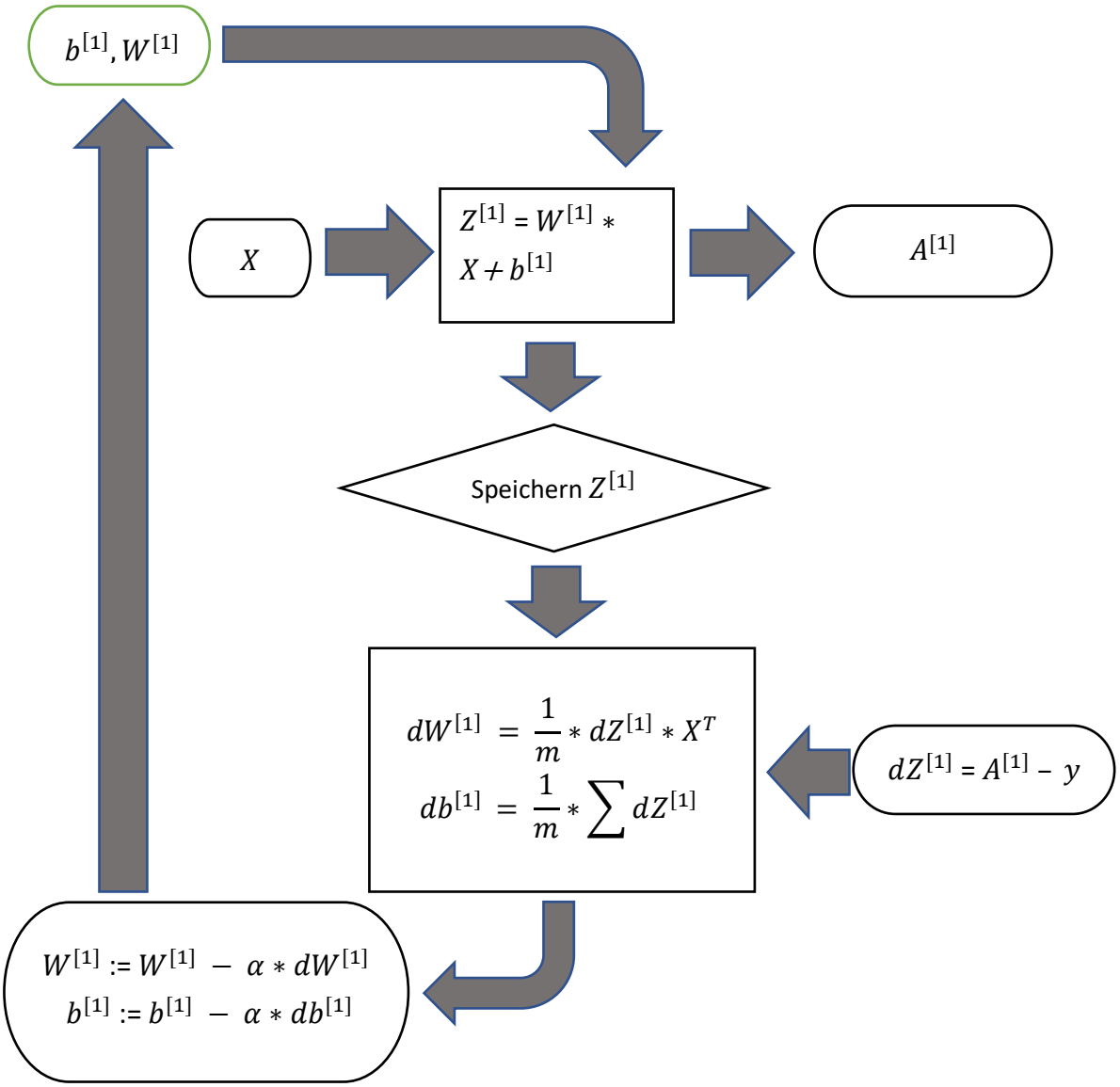


Abbildung 7: Verlauf von einem einfachen Netz

Beispiel Verlauf von neutralen Netze mit mehr als zwei verbogenen Schichten

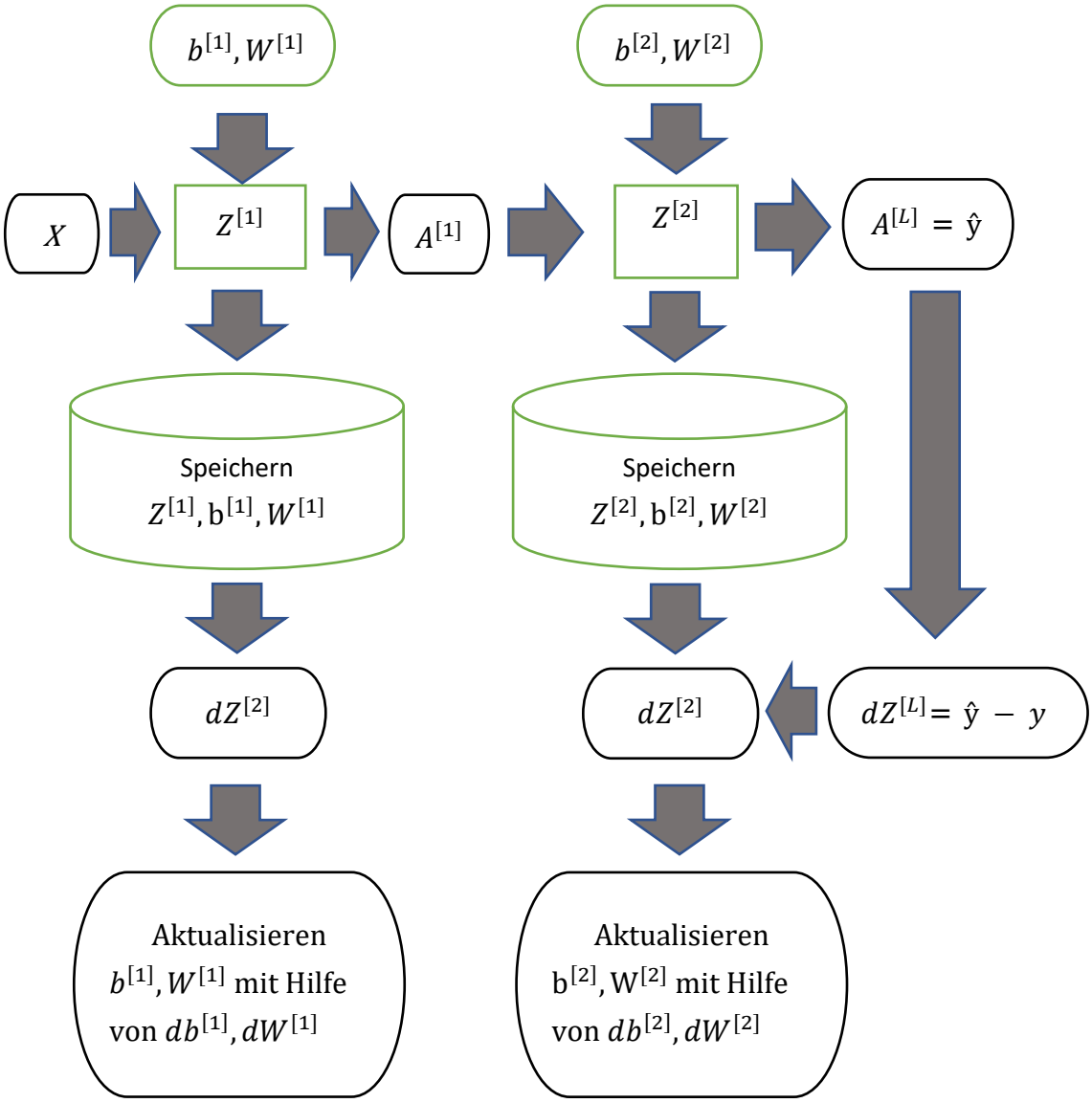


Abbildung 8: Verlauf von einem neutralen Netz

Wobei

L: Anzahl der Schichten

$b^{[l]}$, $W^{[l]}$: Gewichte und Verzerrungen für Berechnung von $Z^{[l]}$

$n^{[l]}$: Anzahl der Einheiten in Schicht (l)

Werte, die in Großbuchstaben geschrieben sind, sind alle Vektoren, da die Vektorisierung von Datenwerten die Berechnungen beschleunigt. Da die Werte vektorisiert sind, wird die Verwendung von Schleifen bei der Programmierung vermieden, was eine Menge Rechenressourcen spart.

$$A^{[l]} = g(Z^{[l]})$$
$$Z^{[l]} = W^{[l]} * A^{[l]} + b^{[l]}$$
$$dZ^{[l]} = dA^{[l]} * g^{[l]'}(Z^{[l]})$$
$$dW^{[l]} = \frac{1}{m} * dZ^{[l]} * A^{[l-1].T}$$
$$db^{[l]} = \frac{1}{m} * \sum dZ^{[l]} = dZ^{[l]}$$
$$dA^{[l-1]} = W^{[l].T} * dZ^{[l]}$$
$$W^{[l]} := W^{[l]} - \alpha * dW^{[l]}$$
$$b^{[l]} := b^{[l]} - \alpha * db^{[l]}$$

Notiz: Eigentlich wird $db^{[l]}$ so programmiert:

$$db^{[l]} = \frac{1}{m} * np.sum(dZ^{[l]}, axis = 1, keepdims = True)$$

Die richtige Dimension jeder Variablen wird in Tabelle 3 angezeigt, wobei m die Anzahl der Eingabeobjekte ist.

Variable	X	$Z^{[l]}$	$W^{[l]}\ und\ dW^{[l]}$	$db^{[l]}\ und\ b^{[l]}$
Dimension	$(n^{[0]}, m)$	$(n^{[l]}, m)$	$(n^{[l]}, n^{[l-1]})$	$(n^{[l]}, 1)$

Tabelle 3: Dimensionierung der Variablen in einem einfachen neuralen Netz

3 Methodik und Umsetzung

Teil 1: Objekterkennung

Der erste Schritt, um das Ziel dieser Arbeit zu erreichen, ist die Auswahl des richtigen Trainingsnetzwerks. Um das richtige Netzwerk auszuwählen, wird die Entscheidung durch die Wahl der Netzwerkarchitektur und den Vergleich mit anderen definiert, um die Vor- und Nachteile herauszufinden.

Die Fortschritte in der Computer Vision haben zu einer Vielzahl von Modellen mit verbesserter Genauigkeit geführt, die anhand des ImageNet-Datensatzes gemessen wurden. Allerdings können nicht alle von ihnen auf Geräten mit begrenzten Rechenressourcen verwendet werden. Um KI-Anwendungen für Geräte wie Mobilgeräte und IoT zu entwickeln, muss man die Ressourcen dieser Geräte kennen, um das richtige Modell für sie auszuwählen. Die bevorzugten Modelle sind in der Regel solche mit einer geringen Anzahl von Berechnungen und hoher Genauigkeit. Aufgrund der oben genannten spezifischen Faktoren wurde das SSD-MobileNetV2 ausgewählt und verwendet, um die Aufgabe der Erkennung von Menschen und Objekten zu bewältigen. Das SSD-MobileNetV2 ist eine Kombination aus zwei verschiedenen Arten von tiefen neuronalen Netzen. Während MobilNetV2 als Basisnetz betrachtet wird, ist SSD ein Erkennungsnetz. Das Basisnetz liefert hochrangige Merkmale für nachfolgende Klassifizierungs- oder Erkennungsaufgaben. Im nächsten Abschnitt wird die Struktur der einzelnen Elemente vorgestellt und erläutert, warum diese Netze ausgewählt wurden.

1 MobileNetV2

MobileNet ist ein tiefes neuronales Netzwerk, das 2017 veröffentlicht wurde [9]. Es ist die Arbeit einer Gruppe von Forschern bei Google. Danach wurde es weiterentwickelt und MobileNetV2 wurde im März 2019 offiziell von [10] veröffentlicht.

Während sich das MobileNetV1-Projekt auf tiefenweise trennbare Faltungstechniken konzentrierte, wird MobileNetV2, das auf MobileNetV1 basiert, durch zwei neue Erweiterungen "Inverted residual block" und "Linear bottlenecks" noch nützlicher.

1.1 Faltungsformen

Der Grund für die außergewöhnliche Leistung des MobileNetV1- und des MobileNetV2-Modells auf mobilen Geräten ist die tiefen-separierbare Faltung. Die tiefen-separierbare Faltung hat eine sehr gute Leistung auf mobilen Geräten, da sie im Vergleich zur herkömmlichen Faltung eine erhebliche Menge an Rechenleistung einspart.

1.1.1 Konventionelle zweidimensionale Faltung

Wie vorher schon bekannt sind, wird die zweidimensionale Faltung normalerweise über die gesamte Tiefe (Kanal) berechnet. Daher erhöht sich die Anzahl der Parameter des Modells in Abhängigkeit von der Tiefe der vorherigen Schicht erheblich.

Aus Gründen der Konsistenz sollen die 3D-Tensoren in der Reihenfolge der Dimensionen (Höhe, Breite, Kanal) geformt werden.

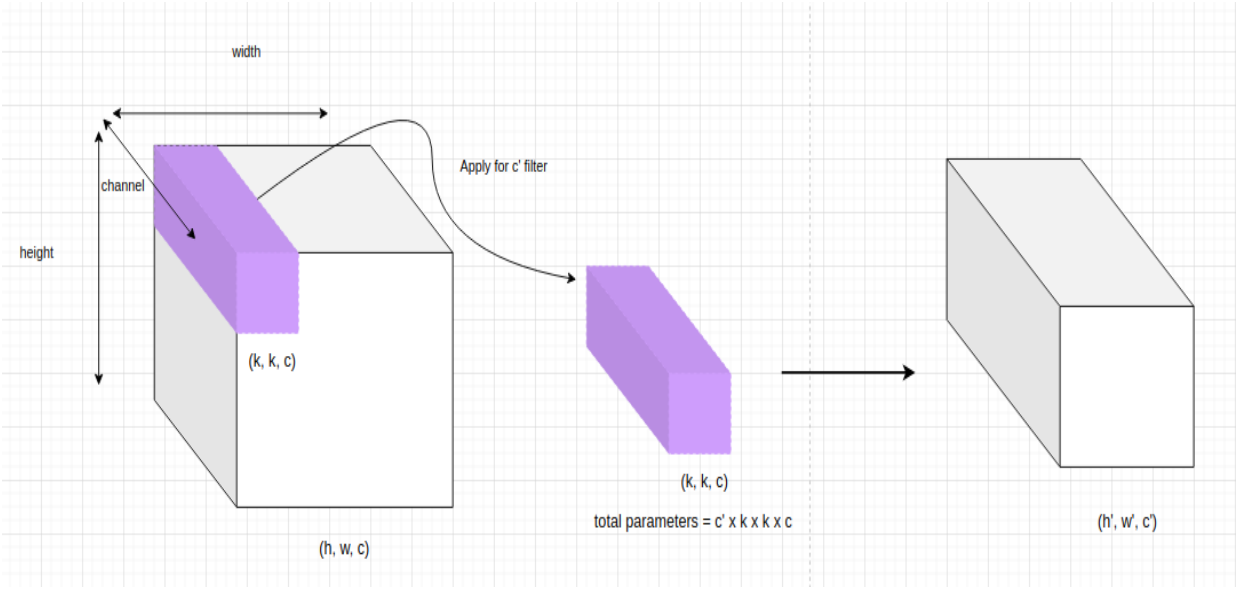


Abbildung 9: Darstellung der konventionelle Faltung [11]

Für eine Eingabe der Größe $h x w x c$ benötigt die Faltung normalerweise $k x k x c$ Parameter, um die Faltung über die gesamte Tiefe der Schichten durchzuführen. Jeder Filter erzeugt eine Ausgangsmatrix der Größe $h' x w' x 1$. Die Anwendung verschiedener Filter führt zu einer Ausgabe der Größe $h' x w' x c'$ (die Ausgabematrizen bei Anwendung der Faltung auf jeden Filter werden mit der Tiefe verkettet). Die Anzahl der für eine normale Faltung zu verwendenden Parameter ist dann: $c' x k x k x c$.

Die Anzahl der Parameter ist vernachlässigbar, wenn die Tiefe des Eingangskanals gering ist, in der Regel in den ersten Schichten von dem Netz. Wenn die Tiefe jedoch allmählich zu den letzten Schichten des neuronalen Faltungsnetzes hin zunimmt, wird die Anzahl der Parameter des Modells gering. Diese Zunahme der Parameter führt zu schwerfälligen Modellen, die viel Speicherplatz beanspruchen und die Berechnungsgeschwindigkeit beeinträchtigen. Alexnet und VGGNet sind typische Modelle mit einer sehr großen Anzahl von Parametern, da nur herkömmliche 2-dimensionale Faltungen angewendet werden.

1.1.2 Tiefenweise trennbare Faltung (Depthwise Separable Convolution)

Die Analyse der Struktur der herkömmlichen zweidimensionalen Faltung lässt den Schluss zu, dass die Tiefe eine der Hauptursachen für die Zunahme der Anzahl der Parameter des Modells ist. Die Lösung der Wahl wäre die tiefengetrennte Faltung. Die tiefengetrennte Faltung beseitigt die Abhängigkeit von der Tiefe und erzeugt dennoch eine Ausgangsform von gleicher Größe wie die normale Faltung. Konkret wird der Prozess in die folgenden Schritte unterteilt:

- Tiefenfaltung:

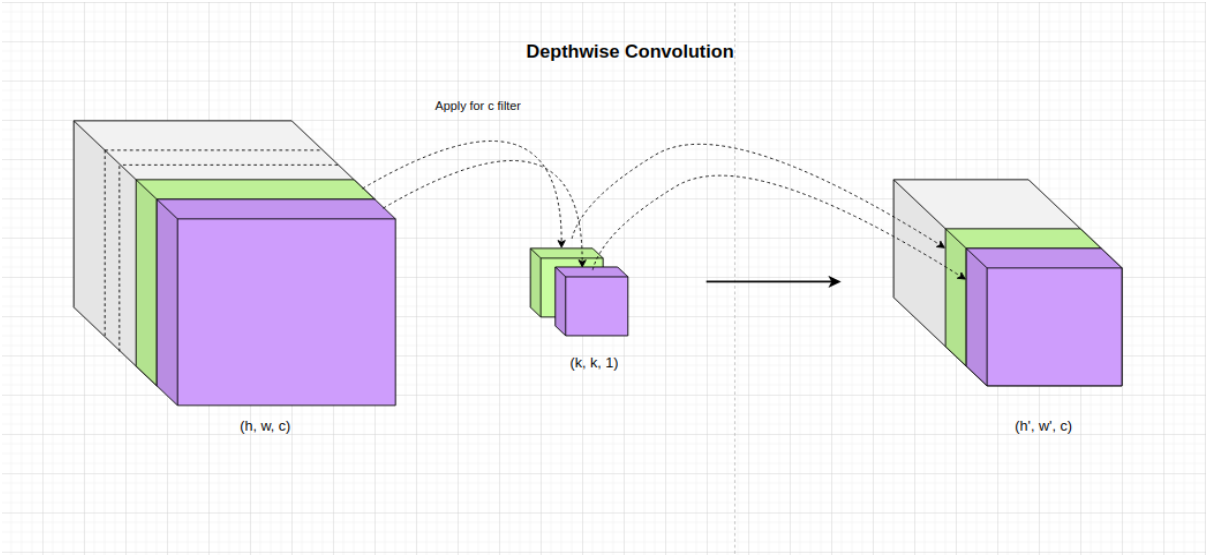


Abbildung 10: Darstellung der Tiefenfaltung (Depthwise Convolution) [11]

Die Tiefe eines dreidimensionalen Tensorblocks wird in Teile (d.h. Kanaldimensionen) aufgeteilt. Dann wird eine regelmäßige Faltung auf jedem der unterteilten Arrays durchgeführt werden. Auf jeden Kanal wird ein anderer Filter und keine gemeinsamen Parameter angewendet. Dieser einfache Vorgang hat zwei wesentliche Auswirkungen auf das Modell:

- Merkmalserkennung:
  Der Prozess der Merkmalserkennung wird durch jeden Filter getrennt. Wenn die Merkmale der Felder unterschiedlich sind, ist es notwendig, jeden Filter für jedes unterschiedliche Feld zu verwenden, um die Merkmale zu erkennen. Dies ist nützlich, weil man in der Praxis das Eingangsmerkmal als ein Bild mit drei RGB-Farbkanälen identifizieren müsste. Auf jeden Kanal des Bildes müssen verschiedene Filter angewendet werden, um jeden Kanal zu erkennen.
- Einsparung von Rechenressourcen:
  Dies ist der größte und notwendigste Effekt dieses Ansatzes. Der Rechenaufwand wird erheblich reduziert, da für die Erzeugung eines Pixels am Ausgang bei der regulären Faltung $kxkxc$ Operationen erforderlich sind, während die Tiefenfaltung nur kxk Operationen erfordert. Darüber hinaus wird auch die Anzahl der zu berechnenden Parameter reduziert. In der obigen Darstellung ist die Anzahl der zu berechnenden Parameter $cxkxkxc'$, während bei der nächsten Methode nur der Parameter $cxkxk$ berechnet werden muss. Das bedeutet, dass die Anzahl der zu berechnenden Parameter um das *c*'-fache reduziert wird.

Das Ergebnis nach der Faltung wird mit der Tiefe konkateniert. Die Ausgabe ist also ein dreidimensionaler Tensor der Größe *h*' *x w*' *x c .*

- Punktweise Faltung:

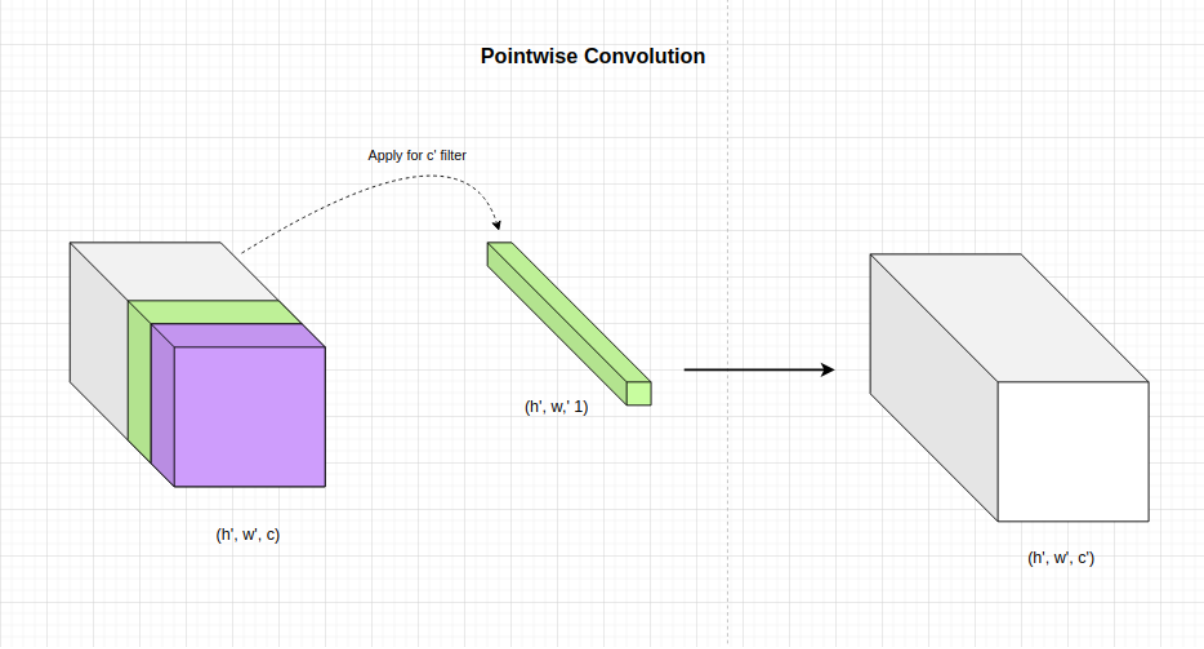


Abbildung 11: Darstellung der Punktweise Faltung [11]

In diesem Schritt wendet man eine punktuelle Faltung an, durch die die Ausgabe ihre ursprüngliche Tiefe erhält. Mit anderen Worten: man ändert die Tiefe der Ausgabe von c auf c'. Die Anzahl der zu diesem Zweck zu verwendenden Filter ist hier c'. Diese Filter haben jeweils eine Filtergröße von $1 x 1 x c$. Da die Filterabmessungen die gleiche Höhe und Breite haben, ändern sich die Werte dieser beiden Dimensionen nicht, nur die Tiefe.

Das Endergebnis ist eine Ausgabe der Größe $h' x w' x c'$. Die Anzahl der in diesem Fall anzuwendenden Parameter ist $c' x c$.

1.1.3 Vergleich von zwei Faltungsformen

1.1.3.1 Anzahl der zu berechneten Parameter

Angenommen, die Eingabe ist ein Tensor der Größe h x w x c und die Ausgabe ist ein Tensor der Größe h' x w' x c'. Die Anzahl der für jede Faltungsart zu berechnenden Parameter wird wie folgt berechnet:

- Konventionelle zweidimensionale Faltung:

$$Anzahl\ der\ Parameter\ =\ c'\ x\ k\ x\ k\ x\ c$$

- Tiefenweise trennbare Faltung:

$$Anzahl\ der\ Parameter\ =\ k\ x\ k\ x\ c\ +\ c'\ x\ c$$

Normalerweise ist c' viel größer als c, da die Tiefe in späteren Schichten des Netzes zunimmt. Das Verhältnis der Anzahl der Parameter zweier Faltungsstrukturen kann in folgender Form geschrieben werden:

$$\frac{c' \, x \, k \, x \, k \, x \, c}{k \, x \, k \, x \, c \; + \; c' \, x \, c} = \frac{c' \, x \, k \, x \, k}{k \, x \, k \; + \; c'} \approx \; k \, x \, k \;\; wenn \;\; k \, x \, k \; \ll \; c'$$

Aus der obigen Berechnung lässt sich ableiten, dass die Modellgröße durch die Anwendung der tiefenweisen Faltung um das $(k \, x \, k)$-fache reduziert werden kann - eine sehr bedeutende Zahl.

Aus diesem Grund wird das MobileNet-Modell sehr häufig auf mobilen Geräten verwendet und wegen seiner Geschwindigkeit und der geringen Rechenressourcen gegenüber Modellen wie Alexnet bevorzugt.

1.1.3.2 Anzahl der auszuführenden Operationen

Um gemeinsam eine Ausgangsform der Größe $h' \, x \, w' \, x \, c'$ zu erzeugen, sollte eine normale Faltung so viele Parameters berechnen:

$$(h' \, x \, w' \, x \, c') \, x \, (k \, x \, k \, x \, c)$$

Dabei ist $(h' \, x \, w' \, x \, c')$ die Anzahl der zu berechnenden Pixel und $(k \, x \, k \, x \, c)$ die Anzahl der Multiplikationen zur Erzeugung eines Pixels. Die tiefenweise trennbare Faltung hingegen muss nur der Reihe durchgeführt werden:

a. Tiefenfaltung:

$(h' \, x \, w' \, x \, c') \, x \, (k \, x \, k)$ Multiplikationen.

b. Punktfaltung:

$(h' \, x \, w' \, x \, c') \, x \, (h' \, x \, w')$ Multiplikationen.

1.2 Invertierter Restblock

Beim Einsatz mobiler Hardware ist es wichtig zu wissen, dass die Geschwindigkeit und die Verarbeitungsleistung der mobilen Geräte wichtige Faktoren sind. Intuitiv könnte dies zu dem Denkfehler führen, zwischen Genauigkeit und Leistung zu wählen und das andere zu "opfern", damit die anderen Faktoren ihren optimalen Wert erreichen können. Tatsächlich gibt es Methoden, die helfen können, Daten so zu verarbeiten, dass sowohl die Leistung als auch die Verarbeitungsgeschwindigkeit auf diesen mobilen Geräten optimiert

werden. Eine dieser Methoden ist der so genannte "invertierte Restblock" und wurde in [10] vorgestellt.

Der Autor geht davon aus, dass die Zwischenschichten in einem Block die nichtlineare Transformation durchführen, so dass sie „dicker" sein müssen, um mehr Transformationen oder mehr berechnete Parameter zu erzeugen, um ein optimales Ergebnis zu erzielen. Die Verknüpfungen zwischen Blöcken werden an den Eingabe- und Ausgabe-Engpässen und nicht an den Zwischenschichten durchgeführt. Daher müssen die Eingangs- und Ausgangsengpass-Schichten nur das Ergebnis aufzeichnen und nicht die nichtlineare Transformation durchführen. Um zu vermeiden, dass die Anzahl der Parameter zwischen den Blöcken für die Berechnung zu groß wird, wird zwischen den Blöcken auch tiefenweise trennbarer Faltung verwendet. Diese Residualarchitektur ist das Gegenteil der traditionellen Residualarchitekturen, da die traditionelle Residualarchitektur eine größere Anzahl von Kanälen am Eingang und Ausgang eines Blocks hat als die Zwischenschichten. Daher wird sie auch als invertierte Residualblockarchitektur bezeichnet.

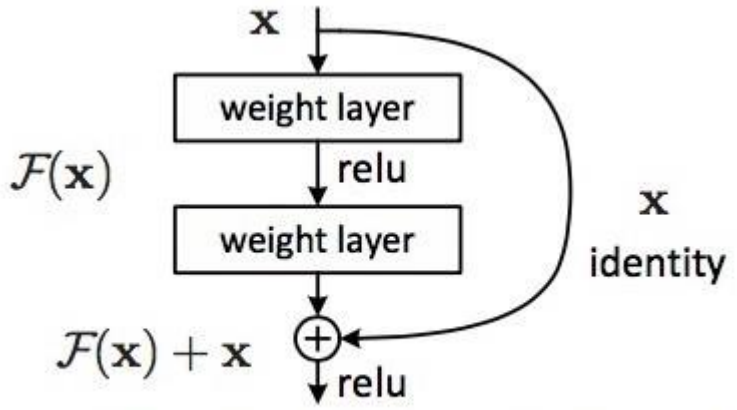


Figure 2. Residual learning: a building block.

Abbildung 12: Grafische Darstellung eines Restblocks [11]

Die Blöcke der vorherigen Schicht werden direkt in die nächste Schicht eingefügt. Betrachtet man die vorherige Schicht als x , so ist die endgültige Ausgabe ein Restblock mit dem Wert $x + F(x)$, nachdem er die zweiseitige Faltung durchgelaufen hat, um das Ergebnis $F(x)$ zu erhalten.

2 SSD-Netz

2.1 Verlauf des SSDs

Wie bei den meisten anderen Objekterkennungsarchitekturen besteht die Eingabe für das SSD aus den Bounding-Box-Koordinaten des Objekts (auch bekannt als Bounding-Box-Offsets) und der Bezeichnung des in der Bounding-Box enthaltenen Objekts. Das SSD-Modell ist so schnell, weil es ein einziges neuronales Netz verwendet. Sein Ansatz basiert auf der Objekterkennung in Feature-Maps (einer 3D-Ausgabeform eines tiefen CNN nach Entfernung der letzten vollständig verbundenen Schichten) in verschiedenen Auflösungen. Das Modell erstellt ein Gitter aus Quadraten auf den Feature-Maps, die als Gitterzellen bezeichnet werden. Jedes Quadrat wird als Zelle bezeichnet, und von der Mitte jeder Zelle aus wird eine Reihe von Standardboxen definiert, um den Rahmen vorherzusagen, in dem sich die Objekte befinden könnten. Zum Zeitpunkt der Vorhersage liefert das neuronale Netz zwei Werte: die Wahrscheinlichkeitsverteilung der Bezeichnung des in der Bounding Box enthaltenen Objekts und eine Koordinate, die als Offset der Bounding Box bezeichnet wird. Der Trainingsprozess ist auch ein Prozess der Verfeinerung der Wahrscheinlichkeitsverteilung der Beschriftung und des Begrenzungsrahmens, um die Eingabewerte des Modells (einschließlich der Beschriftungen und der Offset des Begrenzungsrahmens) an die tatsächliche Situation anzupassen.

Darüber hinaus wird das Netz durch viele Merkmalskarten mit unterschiedlichen Auflösungen kombiniert, um Objekte unterschiedlicher Größe und Form zu erkennen. Im Gegensatz zum schnellen R-CNN-Modell überspringen die SSDs den Maskierungsschritt des Region Proposal Network, um Objektregionen vorzuschlagen. Stattdessen werden die Objekterkennung und die Objektklassifizierung im selben Netzwerk durchgeführt. Der Name des Modells selbst - Single Shot MultiBox Detector - besagt auch, dass das Modell mehrere Box-Frames mit unterschiedlichen Maßstäben verwendet, um Objektregionen zu identifizieren und Objekte zu klassifizieren, wodurch der Schritt der Erstellung des Region Proposal Network minimiert wird. Im Vergleich zum schnellen R-CNN soll die Verarbeitungsgeschwindigkeit um ein Vielfaches erhöht werden, wobei die Verarbeitungsgenauigkeit beibehalten wird.

Method	mAP	FPS	batch size	# Boxes	Input resolution
Faster R-CNN (VGG16)	73.2	7	1	~ 6000	~ 1000 × 600
Fast YOLO	52.7	155	1	98	448 × 448
YOLO (VGG16)	66.4	21	1	98	448 × 448
SSD300	74.3	46	1	8732	300 × 300
SSD512	76.8	19	1	24564	512 × 512
SSD300	74.3	59	8	8732	300 × 300
SSD512	76.8	22	8	24564	512 × 512

Abbildung 13: Vergleichstabelle der Verarbeitungsgeschwindigkeit und der Genauigkeit von Modellobjekt-Erkennungsklassen [12]

Kurz gesagt, das SSD-Modell wird eine Kombination aus 2 Schritten sein:

- Extrahieren von Merkmalskarten aus dem CNN-Netzwerk.
- Anwendung von Faltungsfiltern (oder Kernel-Filtern) zur Erkennung von Objekten auf Merkmalskarten mit unterschiedlichen Auflösungen (Revolution).

2.2 Architektur von den SSD-Netz auf Basis des MobileNetV2

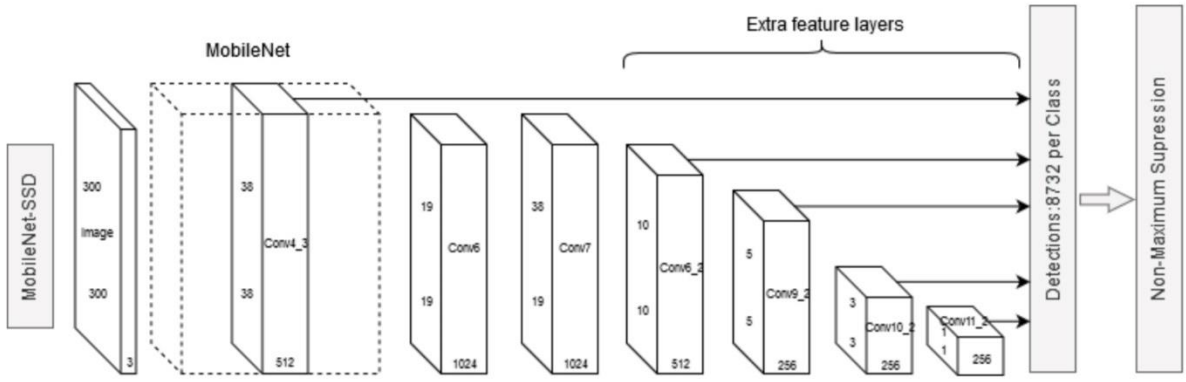


Abbildung 13: Architektur des SSD-MobilenetV2 [13]

SSDs basieren auf einer Vorwärtspropagation einer Standardarchitektur, um einen 3-dimensionalen Merkmalskarten-Ausgabeblock zu erzeugen. Die MobilenetV2-Architektur wurde in dem SSD-MobilenetV2-Modell verwendet und wird als Basisnetz bezeichnet. Es hilft, wie beschrieben, die Merkmale von Objekten und ihre Koordinaten als Vorschlag für die Definition der Gitterzellen zu extrahieren. Dann wird die Architektur hinter dem Basisnetz hinzugefügt, um die Objekterkennung durchzuführen, wie im Abschnitt Extra Feature Layers des Diagramms gezeigt. Diese Schichten werden einfach wie folgt erklärt:

- Eingabeschichten: Nimmt als Eingabe Bilder der Größe (Breite x Höhe x Kanäle) = *300 x 300 x 3* für die SSD300-Architektur
- Conv5_3 Schicht: Dies ist das Basisnetz, das die Architektur von MobilenetV2 verwendet, aber einige vollständig verbundene Schichten am Ende entfernt. Die Ausgabe dieser Schicht ist die Conv4_3-Schicht und ist eine Merkmalskarte der Größe *38 x 38 x 512*.
- Conv4_3-Schicht: Conv4_3 kann als Feature Map mit den Abmessungen *38 x 38 x 512* betrachtet werden. Auf diese Merkmalskarte werden zwei Haupttransformationen angewendet:
  - Anwendung einer Faltungsschicht wie bei einem normalen CNN, um die Ausgabe der nächsten Schicht zu erhalten. Konkret hat die Faltungsschicht einen *3 x 3 x 1024* Faltungs-Kernel, die resultierende Conv6-Ausgabe hat eine Größe von *19 x 19 x 1024*.
  - Auch in diesem Schritt wird ein Klassifikator angewandt (ebenfalls auf der Grundlage des *3 x 3* Faltungsfilters), um Objekte auf der Merkmalskarte zu identifizieren. Dies ist ein komplizierter Prozess, da er die Objekterkennung (durch die Erkennung von Begrenzungsrahmen) und die Objektklassifizierung sicherstellen muss. Zunächst wird die Merkmalskarte der Größe *38 x 38 x 512* in eine Gitterzelle der Größe *38 x 38* aufgeteilt (die Tiefe wird ignoriert, da die Faltung auf der ganzen Welt durchgeführt wird), dann werden in jeder Zelle der Gitterzelle 4 Standard-Begrenzungsrahmen mit unterschiedlichen Seitenverhältnissen erstellt, wobei jeder Standard-Begrenzungsrahmen die folgenden Parameter finden muss: Wahrscheinlichkeitsverteilung der Beschriftung ist ein Vektor mit *(n_classes + 1)* Dimension. Auch 4 weitere Parameter werden als Offsets benötigt, um die Bounding-Box des Objekts in den Rahmen zu bestimmen. Multipliziert mit der Anzahl der Zellen von Conv4_3, um c zu erhalten, ist die Ausgabemenge ein Tensor der Größe *38 x 38 x 4 x (n_classes+5)*, falls der Hintergrund auch ein Label ist, hat der Tensor die Größe *38 x 38 x 4 x (n_classes+4)*. Und die Anzahl der erzeugten Bounding Boxes ist *38 x 38 x 4*.

- Der Prozess der Anwendung des Klassifikators auf die Merkmalskarte ist ähnlich wie bei den Schichten Conv7, Conv8_2, Conv_9, Conv10_2, Conv11_2. Die Form der folgenden Schichten hängt von dem in der vorherigen Schicht angewandten

Faltungsprozess, der Größe des Kernel-Filters (wie im obigen Diagramm dargestellt, ist die Kernel-Größe immer 3 x 3) und der Schrittweite der Faltungsschicht ab. Für jede Zelle auf der Merkmalskarte wird eine Anzahl von 4 oder 6 Standard-Bounding Boxes definiert. Daher ist die Anzahl der Standardboxen, die in den nachfolgenden Schichten erzeugt werden, wie folgt:

- Conv7: $19 \times 19 \times 6 = 2166$ Kisten (6 Kisten/Zelle)
- Conv8_2: $10 \times 10 \times 6 = 600$ Kisten (6 Kisten/Zelle)
- Conv9_2: $5 \times 5 \times 6 = 150$ Kästchen (6 Kästchen/Zelle)
- Conv10_2: $3 \times 3 \times 4 = 36$ Kästchen (4 Kästchen/Zelle)
- Conv11_2: $1 \times 1 \times 4 = 4$ Kästchen (4 Kästchen/Zelle). Insgesamt ist die Anzahl der Ausgabe am Ende:

$$5776 + 166 + 600 + 150 + 36 + 4 = 8732$$

[14]

3 Implementierung

Für die Erkennung von Objekten wie Fußgängern, Autos oder anderen Landfahrzeugen, die sich während der Zugfahrt zu nahe an den Schienen befinden könnten, wird das SSD-MobileNetV2 mit diesen drei neuen Klassen neu trainiert. Die Bilder, die für das Training verwendet werden, stammen von Open Images.

Es wird empfohlen, zu Beginn des Trainings auf die Größe der Daten zu achten. Eine zu große Datenmenge kann dazu führen, dass das Training sehr langsam ist. Um also sicherzustellen, dass die Daten der drei Klassen nicht zu groß sind, sollte man zunächst die Statistik der Daten überprüfen. Auch wenn die Datenmenge reduziert wird, um das Training zu beschleunigen, sollten die Daten die gleiche Verteilung wie die Gesamtdaten haben. Wie in den folgenden Abbildungen zu sehen ist, bleibt die gleiche Verteilung auch nach der Reduzierung der Bildmenge erhalten.

```
-------------------------------------
 'train' set statistics
-------------------------------------
  Image count:  321486
  Bounding box count:  1239575
  Bounding box distribution:
    Person:  922047/1239575 = 0.74
    Car:  241120/1239575 = 0.19
    Land vehicle:  76408/1239575 = 0.06
```

```
-------------------------------------
 'validation' set statistics
-------------------------------------
  Image count:  10720
  Bounding box count:  24298
  Bounding box distribution:
    Person:  12371/24298 = 0.51
    Car:  9285/24298 = 0.38
    Land vehicle:  2642/24298 = 0.11


-------------------------------------
 'test' set statistics
-------------------------------------
  Image count:  32255
  Bounding box count:  74852
  Bounding box distribution:
    Person:  37997/74852 = 0.51
    Car:  28471/74852 = 0.38
    Land vehicle:  8384/74852 = 0.11


-------------------------------------
 Overall statistics
-------------------------------------
  Image count:  364461
  Bounding box count:  1338725
```

Die Statistik der für das Training verwendeten Daten ist:

```
STATISTIC OF 10000 IMAGES:
-------------------------------------
 'train' set statistics
-------------------------------------
  Image count:  8820
  Bounding box count:  34763
  Bounding box distribution:
    Person:  26078/34763 = 0.75
    Car:  6552/34763 = 0.19
    Land vehicle:  2133/34763 = 0.06
```

```
-------------------------------------
 'validation' set statistics
-------------------------------------
  Image count:  294
  Bounding box count:  668
  Bounding box distribution:
    Person:  335/668 = 0.50
    Car:  255/668 = 0.38
    Land vehicle:  78/668 = 0.12


-------------------------------------
 'test' set statistics
-------------------------------------
  Image count:  885
  Bounding box count:  2050
  Bounding box distribution:
    Person:  1034/2050 = 0.50
    Car:  758/2050 = 0.37
    Land vehicle:  258/2050 = 0.13


-------------------------------------
 Overall statistics
-------------------------------------
  Image count:  9999
   Bounding box count:  37481
```

Das Training wurde für 48 Epochen mit einer Stapelgröße von 4 festgelegt. Die Trainingszeit betrug etwa 36 Minuten und 18 Sekunden pro Epoche. Das Ergebnis des Trainings ist in der Abbildung 14 dargestellt.

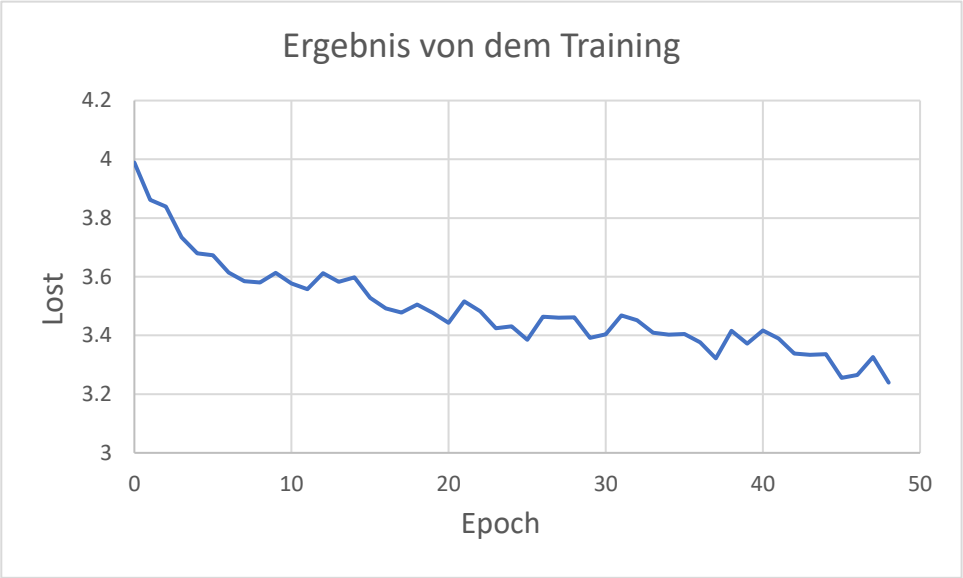


Abbildung 14: Ergebnis von dem Training

Das trainierte Modell wurde dann an echten Videos getestet, um die Zuverlässigkeit der Leistung in Echtzeit zu überprüfen. Die Testergebnisse zeigten, dass das Modell bei diesen Videos eine gute Leistung erbrachte und einen Spitzenwert von maximal 25 Bildern pro Sekunde aufnahm, was eine akzeptable Rate für kleine Geräte wie den Jetson Nano ist.

Teil 2: Gleisbahnerkennung

1 Kamera kalibrieren

1.1 Der Grund des Kamerakalibrieren

Die geometrische Kamerakalibrierung, auch als Kameraresektion bezeichnet, schätzt die Parameter eines Objektivs und des Bildsensors einer Bild- oder Videokamera. Mit diesen Parametern kann man Objektivverzerrungen korrigieren, die Größe eines Objekts in Welteinheiten messen oder den Standort der Kamera in der Szene bestimmen. Diese Aufgaben werden in Anwendungen wie der Bildverarbeitung zum Erkennen und Messen von Objekten verwendet. Sie werden auch in der Robotik, für Navigationssysteme und bei der Rekonstruktion von 3D-Szenen verwenden.

Zu den Kameraparametern gehören intrinsische und extrinsische Parameter sowie Verzerrungskoeffizienten. Zur Schätzung der Kameraparameter werden die 3D-Weltpunkte und die entsprechenden 2D-Bildpunkte benötigt. Man kann diese Entsprechungen erhalten, indem man mehrere Bilder eines Kalibrierungsmusters verwendet, zum Beispiel ein Schachbrett. Nachdem die Kamera kalibriert geworden ist, könnte man die Genauigkeit seiner Kamera schätzen sowie ausrechnen zum Beispiel durch:

- Aufzeichnen der relativen Positionen der Kamera und des Kalibrierungsmusters
- Rechnen der Reprojektionsfehler
- Rechnen der Parameterschätzungsfehler

Die intrinsischen und extrinsischen Parameter der Kamera werden durch eine Matrix beschrieben. Zu den intrinsischen Parametern einer Kamera gehören Brennweite, Bildsensorformat und Hauptpunkt. Die extrinsischen Parameter bestehen aus einer Rotation R und einer Translation t. Der Ursprung des Koordinatensystems der Kamera liegt in ihrem optischen Zentrum, und ihre x- und y-Achsen definieren die Bildebene.

Lochkameras führen zu erheblichen Bildverzerrungen. Zwei Hauptarten der Verzerrung sind die radiale und die tangentiale Verzerrung. Die radiale Verzeichnung lässt gerade Linien gekrümmt erscheinen. Die radiale Verzerrung nimmt zu, je weiter die Punkte von der Bildmitte entfernt sind. In der folgenden Abbildung sind zum Beispiel zwei Kanten

eines Schachbretts mit roten Linien markiert. Man kann hier jedoch sehen, dass der Rand des Schachbretts keine gerade Linie ist und nicht mit der roten Linie übereinstimmt. Alle erwarteten geraden Linien sind gekrümmt.

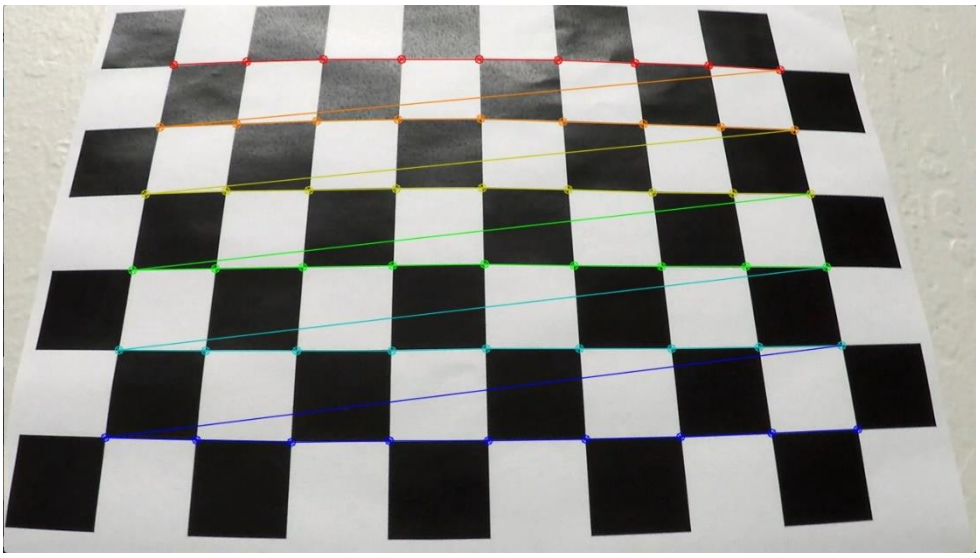


Abbildung 15: Darstellung des Schachbretts für Projektanwendung

1.2 Definition von Begriffen

- Radiale Bildverzerrung : Eine radiale Verzerrung tritt auf, wenn sich Lichtstrahlen in der Nähe der Kanten einer Linse stärker biegen als in ihrem optischen Zentrum. Je kleiner das Objektiv ist, desto größer ist die Verzerrung.

$$x_{verzerrt} = x * (1 + k_1 * r^2 + k_2 * r^4 + k_3 * r^6)$$

$$y_{verzerrt} = y * (1 + k_1 * r^2 + k_2 * r^4 + k_3 * r^6)$$

wobei

x, y: unverzerrte Pixelpositionen

k_n: Radiale Verzeichnungskoeffizienten des Objektivs

$$r^2 = x^2 + y^2$$

Die Variablen x und y sind normalisierte Bildkoordinaten. Normalisierte Bildkoordinaten werden aus Pixelkoordinaten berechnet, indem sie in das optische Zentrum verschoben und

durch die Brennweite in Pixel dividiert werden. Somit sind x und y dimensionslos.

- Tangentiale Verzerrung: Eine tangentiale Verzerrung tritt auf, wenn das Objektiv und die Bildebene nicht parallel sind. Die tangentialen Verzerrungskoeffizienten modellieren diese Art der Verzerrung.

$$x_{verzerrt} = x + [2 * p_1 * x * y + p_2 * (r^2 + 2 * x^2)]$$

$$y_{verzerrt} = y + [p_1 * (r^2 + 2 * y^2) + 2 * p_2 * x * y]$$

1.3 Prozess

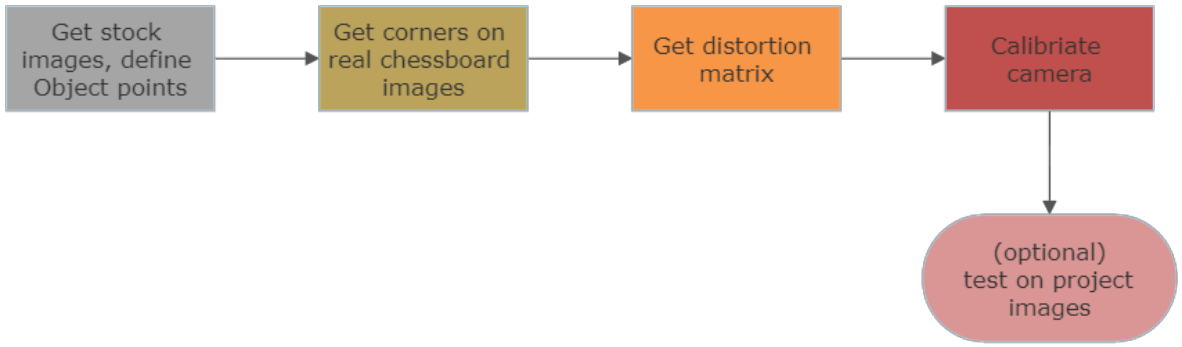


Abbildung 16: Prozessablaufbeschreibung der Kamerakalibrierung

Um die radiale und tangentiale Verzerrungen zu finden, werden die fünf Parameter betrachtet, durch die die Verzerrungen beschrieben werden und zwar:

(k1 k2 p1 p2 k3)

Man nennt diese Parameter Verzerrungskoeffizienten. Um diese Parameter zu finden, muss man einige Beispielbilder eines genau definierten Musters (z. B. eines Schachbretts) bereitstellen. Man kann bestimmte Punkte finden, deren relative Position man bereits kennt (zum Beispiel die Ecken eines Quadrats im Schachbrett). Die Koordinaten dieser Punkte im realen Raum und im Bild sind bereits bekannt, so dass die Verzerrungskoeffizienten gesucht werden können. Um bessere Ergebnisse zu erzielen, werden mindestens zehn Testmuster benötigt.

Um Muster im Schachbrett zu finden, kann man die Funktion `cv.findChessboardCorners()` verwenden. Es ist notwendig, die Form des Musters zu bestimmen, zum Beispiel 8x8-Raster, 5x5-Raster, und so weiter. In diesem Beispiel wurde das 9x6-Raster verwendet. (Normalerweise hat ein Schachbrett 8x8 Quadrate und 7x7 Innenecken).

Es gibt die "echten" Ecken zurück, wenn ein Muster erhalten wird. Diese Eckpunkte sind in einer bestimmten Reihenfolge angeordnet (von links nach rechts, von oben nach unten). Nachdem die Objekt- und Bildscheitelpunkte definiert sind, kann nun die Kalibrierung beginnen. Wie im Anhang wurde die Funktion `cv.calibrateCamera()` verwendet, die die Kameramatrix, Verzerrungskoeffizienten, Rotations- und Translationsvektoren usw. zurückgibt. Für diesen Schritt im Projekt wird die Funktion `cv.undistort()` aufgerufen.

2 Gaußscher Unschärfe-Algorithmus und morphologische Transformationen

Bei der Bildvorverarbeitung ist die Unschärfe ein äußerst wichtiger Schritt. Wenn die Eingabedaten als Bild vorliegen, besteht die Aufgabe des Ingenieurs darin, zunächst zu versuchen, alle Störfaktoren zu entfernen, die dieses Bild beeinträchtigen. Dieses Störungen können die Leistung des Systems stark beeinträchtigen und sind besonders in Bereichen mit hohen Präzisionsanforderungen wie der Medizin- oder Feinmechanik wichtig. Daher ist es wichtig, die Auswirkungen der Filtertechniken und die Situationen zu verstehen, in denen sie eingesetzt werden sollten.

Das allgemeine Prinzip der Filterverfahren besteht darin, die Bildmatrix mit einer Filtermatrix (Kernel) zu multiplizieren. Filtermatrix (Kernel) kann auch als Faltungsfenster (bei Faltungsmultiplikation), Filterfenster, Maske und so weiter bezeichnet werden. In diesem Arbeit wird der Begriff Filtermatrix (Kernel) verwendet. Das Multiplizieren des Bildes mit der Filtermatrix ist so, als würde man die Filtermatrix zeilenweise über das Bild schieben und mit jedem Bereich des Bildes multiplizieren, wobei die Ergebnisse zum zentralen Pixelergebnis addiert werden. Die Eingangsmatrix I wird mit der Filtermatrix multipliziert, um die Ausgangsmatrix O zu bilden.

Tatsächlich ist ersichtlich, dass es zwei Bildfilter gibt, Korrelation und Faltung. Während der Korrelation wird die Filtermatrix verschoben und mit jedem Bereich des Bildes multipliziert. Bei der Faltung wird die Filtermatrix jedoch um 180 Grad (sowohl horizontal als auch vertikal) gedreht, bevor die Multiplikation durchgeführt wird. Diese beiden Operationen sind äquivalent, wenn die Filtermatrix symmetrisch ist. Für jeden Filter gibt es verschiedene Filtermatrizen (Kernel), es gibt keine spezielle Regel zur Bestimmung von M. Die Größe der Matrix M ist meistens eine ungerade Zahl. Beispiel: 3x3, 5x5. Beim Multiplizieren der entsprechenden Elemente (zwischen Pixeln, Nachbarpunkten -

Komponenten im Kernel) fehlen für die Elemente am Rand einige Pixel, derzeit gibt es viele Lösungen wie das Weglassen, Einfügen einer (mehreren) Zeile oder Spalte mit die nächste Null oder gleich dem Wert, oder wird eine Spiegelsymmetrie am Rand des Bildes erstellt. Der Gauß-Filter wird als der nützlichste Filter angesehen, der implementiert wird, indem das Eingabebild mit einer Gauß-Filtermatrix gefaltet und dann zusammengefügt wird, um das Ausgabebild zu bilden. Die allgemeine Idee ist, dass der Wert jedes Pixels mehr von den benachbarten Pixeln als von den entfernten Pixeln abhängt. Das Gewicht der Abhängigkeit wird als Gaußsche Funktion verwendet (auch in der Normalverteilung verwendet). Unten ist eine Darstellung der Gaußschen Filtermatrix:

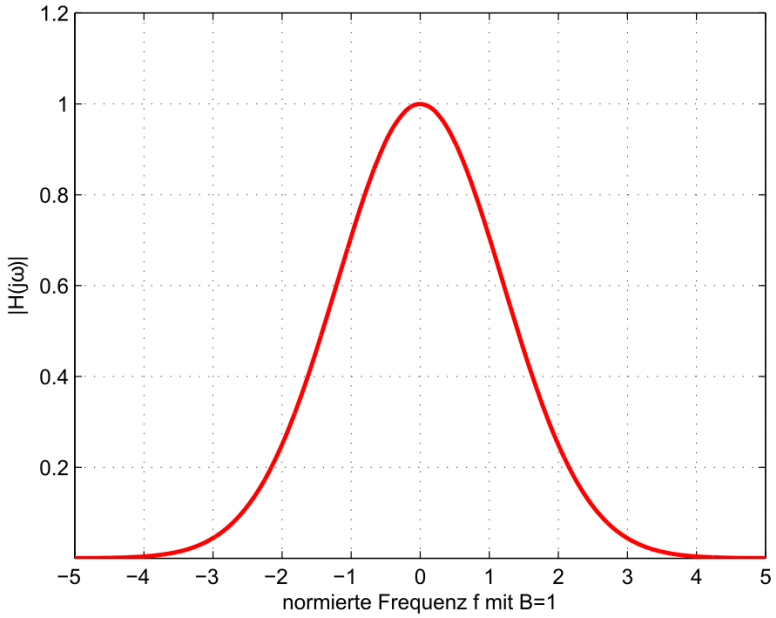


Abbildung 17: Darstellung der Gaußschen Matrix [17]

Angenommen, dass das Bild eindimensional ist. Der Pixel in der Mitte hat das größte Gewicht. Pixel, die weiter von der Mitte entfernt sind, werden mit zunehmendem Abstand von der Mitte abnehmend gewichtet. Je näher der Punkt am Mittelpunkt liegt, desto mehr trägt er zum Mittelpunktwert bei. Hinweis: Tatsächlich basiert die Bildfilterung auf einer 2-D (horizontal und vertikal) Gauß-Funktion. Die Normalverteilung kann wie folgt ausgedrückt werden:

$$G_0(x, y) = Ae^{\frac{-(x-\mu_x)^2}{2\sigma_x^2} + \frac{-(y-\mu_y)^2}{2\sigma_y^2}}$$

wobei μ der Mittelwert (Spitze) ist und σ^2 ist die Varianz der Variablen. Der Parameter bestimmt die Wirkung des Gauss-Filters auf das Bild. Die Größe der Filtermatrix (Kernel) muss groß genug gewählt werden.

Aktueller Code für die Gauß-Filterung mit Python-OpenCV: In OpenCV wird die folgende Funktion zum Filtern von Gauss verwendet: `cv.GaussianBlur(src, ksize, sigmaX, dst, sigmaY, borderType)`. Die benötigten Parameter für die Funktion sind:

Parameter	Bedeutung
src	Eingabebild
dst	Ausgabebild
ksize	Gaußsche Kernelgröße. [Höhe Breite]. Höhe und Breite sollten ungerade sein und können unterschiedliche Werte haben. Wenn ksize auf [0 0] gesetzt ist, wird ksize aus Sigma-Werten berechnet
sigmaX	Kernel-Standardabweichung entlang der X-Achse (horizontale Richtung).
sigmaY	Kernel-Standardabweichung entlang der Y-Achse (vertikale Richtung).
borderType	Gibt Bildgrenzen an, während der Kernel auf Bildgrenzen angewendet wird. Mögliche Werte sind: `cv.BORDER_CONSTANT` `cv.BORDER_REPLICATE` `cv.BORDER_REFLECT` `cv.BORDER_WRAP` `cv.BORDER_REFLECT_101` `cv.BORDER_TRANSPARENT` `cv.BORDER_REFLECT101` `cv.BORDER_DEFAULT` `cv.BORDER_ISOLATED`

Tabelle 4: Erklärung von Funktion `cv.GaussianBlur()`

3 Canny Kantenerkennung

In Bildern gibt es oft Komponenten wie: glatte Bereiche, Ecken/Kanten und Rauschen. Kanten in einem Bild haben wichtige Merkmale, die normalerweise zum Objekt im Bild gehören. Daher ist der Canny-Algorithmus für die Kantenerkennung in Bildern einer der beliebtesten sowie bekanntesten Algorithmen in der Bildverarbeitung. Der Canny-Kantenerkennungsalgorithmus besteht aus vier Hauptschritten:

- Rauschunterdrückung: Bild verwischen, Rauschen mit Gauß-Filter der Größe 5x5 reduzieren. Die Größe 5x5 eignet sich in der Regel gut für den Canny-Algorithmus. Es ist entsprechend auch möglich, die Größe des Unschärfefilters ändern zu können.

- Berechnung von Gradient und Gradientenrichtung: Hier werden Sobel-X und Sobel-Y Filter (3x3) verwendet, um das abgeleitete Bild Gx und Gy zu berechnen. Danach können das Gradientenbild und der Winkel des Gradienten nach der folgenden Formel weiter berechnet werden. Das abgeleitete Bild Gx und Gy ist eine Matrix (z. B. 640x640), das Ergebnis der Berechnung des abgeleiteten Bildkantengradienten ist ebenfalls eine Matrix (640x640), wobei jeder Pixel dieser Matrix die Größe der Lichtwerttransformation an der entsprechenden Position im Originalbild darstellt. Auch die Winkelmatrix hat die gleiche Größe (640x640), wobei jeder Pixel der Winkelmatrix den Winkel darstellt, der die Richtung der Kante an der entsprechenden Position beschreibt. Ein einfaches Beispiel: Wenn der Gradientenwinkel 0 Grad beträgt, dann ist die Kante auf dem Bild eine vertikale Linie (das heißt 90 Grad zur horizontalen Achse oder senkrecht zur Gradientenrichtung). Bei der Berechnung liegt der Wert der Gradientenrichtung im Bereich [-180, 180] Grad. Der genaue Wert dieses Winkels wird nicht beibehalten, aber diese Werte werden gesammelt und in vier Bereiche sortiert, die vier Richtungen darstellen: horizontale Richtung (0 Grad), rechte diagonale Richtung (45 Grad), vertikale Richtung (90 Grad) und linke diagonale Richtung (135 Grad).

$$Edge\_Gradient(G) = \sqrt{G_x^2 + G_y^2}$$

$$Angle\ (\theta) = tan^{-1}(\frac{G_y}{G_x})$$

- Nicht-Maximum-Unterdrückung (NMS oder Non max suppression): Entfernt Pixel an Positionen, die nicht dem globalen Maximum entsprechen. Bei diesem Schritt wird ein 3x3-Filter verwendet, der nacheinander durch die Pixeln des Gradientenbildes läuft. Bei der Filterung wird geprüft, ob die Gradientengröße des zentralen Pixels das Maximum (lokales Maximum) im Vergleich zu den Gradienten in den umliegenden Pixeln ist. Wenn es wirklich der Maximalwert ist, wird dieser Pixel beibehalten. Und wenn der Pixel dort kein

benachbartes Maximum hat, wird seine Gradientengröße auf Null gesetzt. Nur der zentrale Pixel wird mit zwei benachbarten Pixeln in Gradientenrichtung verglichen. Wenn die Gradientenrichtung z. B. 0 Grad beträgt, wird der mittlere Pixel mit seinen linken und rechten Nachbarpixeln verglichen. In einem anderen Fall, wenn die Gradientenrichtung 45 Grad beträgt, wird es mit zwei benachbarten Pixeln verglichen, der oberen rechten Ecke und der unteren linken Ecke des mittleren Pixels. Das Gleiche gilt für die beiden anderen Fälle mit Gradientenrichtung. Eine Illustration der Filterschwelle wird in Abbildung 18 dargestellt.

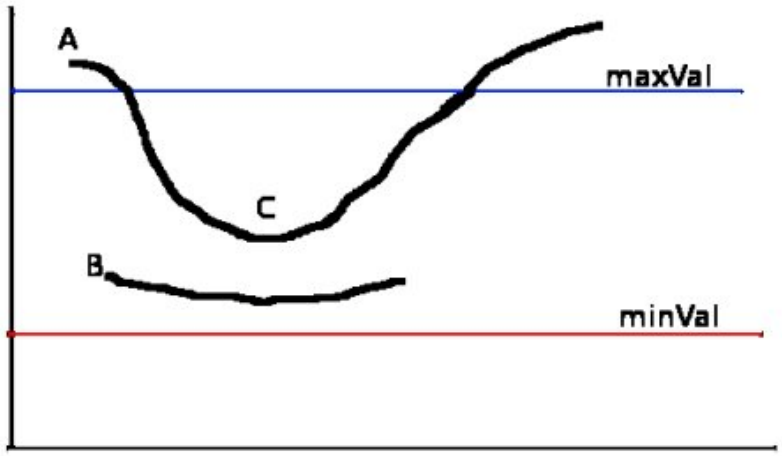


Abbildung 18: Non- Max-Supression
Filterschwelle [17]

Aktueller Code für die Gauss-Filterung mit Python-OpenCV: In OpenCV wird die folgende Funktion verwendet:

```
cv.Canny(image, edges, threshold_1, threshold_2, apertureSize,
L2gradient).
```

Die benötigten Parameter für die Funktion sind:

Parameter	Bedeutung
Image	Eingabebild (Bedingung: das Bild muss ein Graustufenbild sein)
Edges	Ausgabebild
Threshold_1	Die erste Schwelle für den Hystereseprozess Minval
Threshold_2	Zweite Schwelle für Hysterese MaxVal
apertureSize	Die Kerngröße des Sobel-Operators(der Standardwert ist 3 aber am meistens benutzt man den Wert 5)
L2gradient	Es gibt die Gleichung für die Suche nach Gradientengrößen an.

Tabelle 5: Erklärung von Funktion `cv.Canny()`

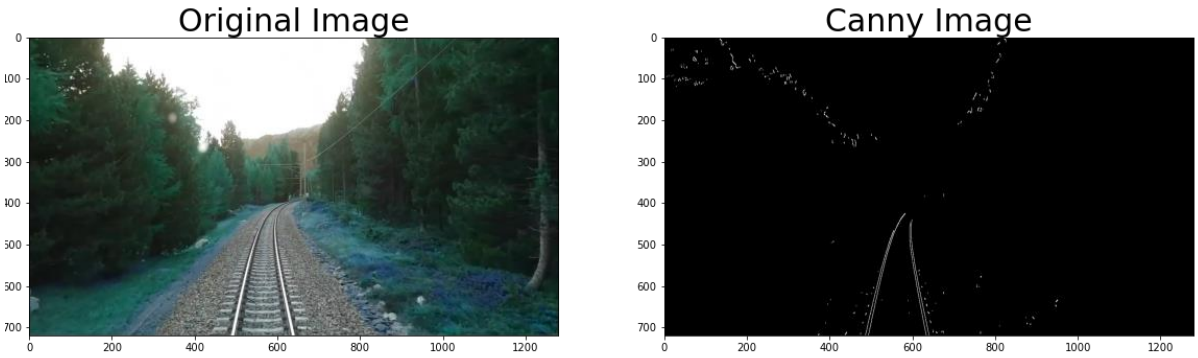


Abbildung 19: Ein Beispiel der Anwendung von Cannys Methode

4 Prozess des Verzerrens der Bildperspektive

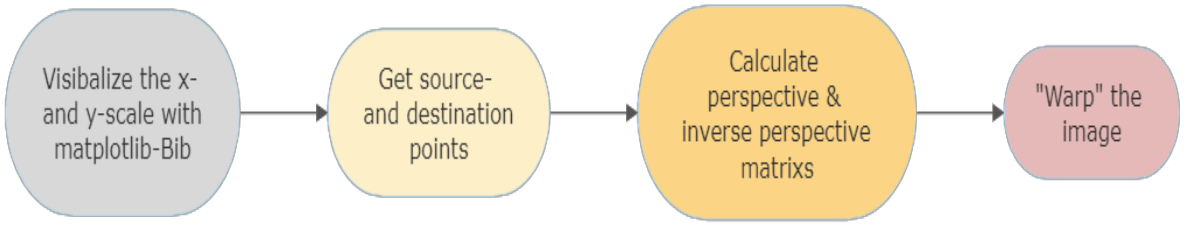


Abbildung 20: Prozessablauf des Verzerrens der Bildperspektive

Die Methode des Image Perspective Warping ist eine weit verbreitete Methode im Bereich autonomer Fahrzeuge sowie der Objekterkennung. Eine einfache Definition dieser Methode besteht darin, dass das Bild so bearbeitet wird, als ob es aus einer anderen Perspektive betrachtet würde, nämlich vom Himmel zum Boden. Daher hat diese Methode einen anderen Namen, der Bird Eyes View ist. Hier ist ein Beispiel für die Anwendung dieser Methode.

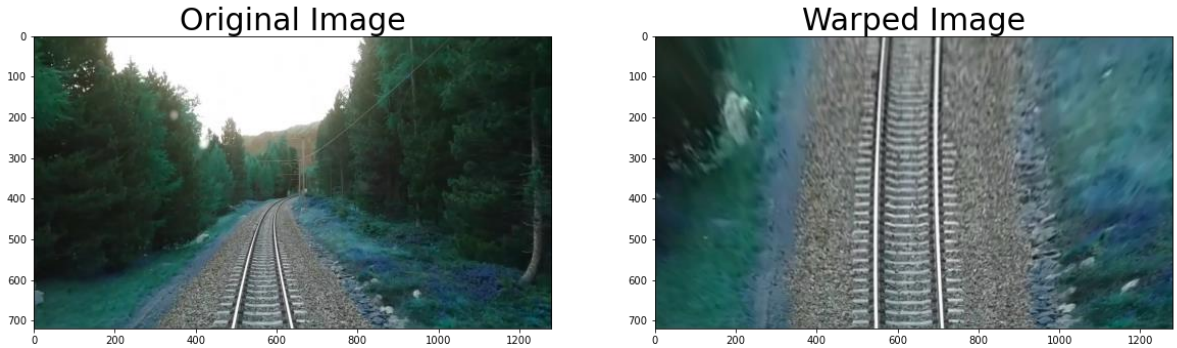


Abbildung 21: Beispiel der Verwendung von „Image Warping Process"

Wie am Anfangskapitel definiert, ist die Perspektivtransformation mit der Änderung des Blickwinkels verbunden. Bei dieser Art der Transformation werden Parallelität, Länge und Winkel nicht beibehalten. Sie bewahren aber die Kollinearität und Inzidenz. Das bedeutet, dass die Geraden auch nach der Transformation gerade bleiben.

Im Allgemeinen kann die perspektivische Transformation ausgedrückt werden als

$$\begin{bmatrix} t_i * x' \\ t_i * y' \\ t_i \end{bmatrix} = \begin{bmatrix} a_1 & a_2 & b_1 \\ a_3 & a_4 & b_2 \\ c_1 & c_2 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Wobei

$\begin{bmatrix} t_i * x' \\ t_i * y' \\ t_i \end{bmatrix}$: Skalierungsfaktor

$\begin{bmatrix} a_1 & a_2 & b_1 \\ a_3 & a_4 & b_2 \\ c_1 & c_2 & 1 \end{bmatrix}$: Transformationsmatrix

(x, y): Eingabepunkte

(x', y'): Ausgabepunkte

$\begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix}$: Definiert Transformationen wie Rotation, Skalierung

$\begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$: Definiert den translatorischen Vektor

$[c_1 \quad c_2]$: Der Projektionsvektor

sind.

Die hier verwendete Funktion ist `cv2.getPerspectiveTransform(src, dst[,solvMethod]`.

Parameter	Bedeutung
src	Koordinaten von Viereckscheitelpunkten im Quellbild. Hier ist es ([(0.3,0.65),(0.6,0.65),(0,1),(0.9,1)])
dst	Koordinaten der entsprechenden Viereckscheitelpunkte im Zielbild. Hier ist es ([(0,0), (1, 0), (0,1), (1,1)])

Tabelle 6: Erklärung von Funktion `cv2.getPerspectiveTransform()`

Nachdem die Funktion angewendet wurde, sind hier einige Ergebnisse aus dem Projekt.

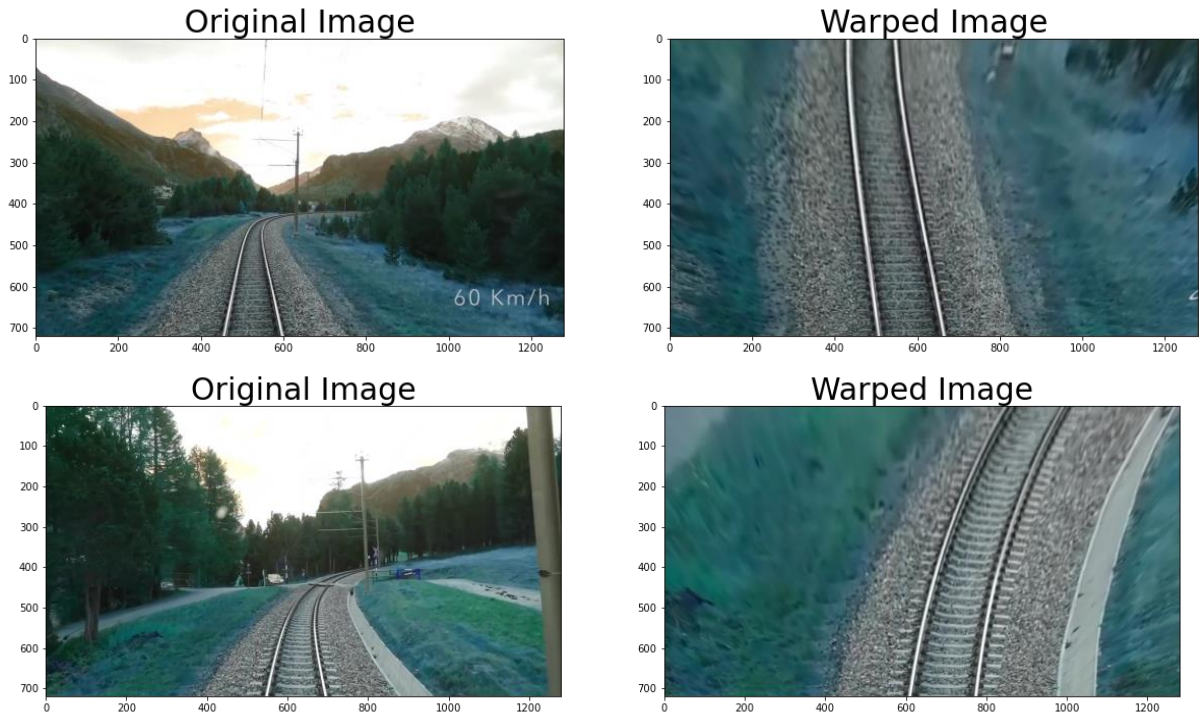


Abbildung 22: Beispielergebnisse von der Anwendung der Funktion `cv2.getPerspectiveTransform()`

5 Histogram

Das Ergebnis dieser Gleichung ist ein Histogramm des Bildes in Bezug auf die X-Achse. Jeder Teil des Histogramms zeigt, wie viele weiße Pixel sich in jeder Spalte des Bildes befinden. Dann werden die höchsten Spitzenwerte (Maximum oder Peak) auf jeder Seite des Bildes angegeben, einer für jede Pfadlinie.

6 Algorithmus

In diesem Abschnitt wird diskutiert, wie der Algorithmus aufgebaut und modifiziert wurde. Der Ansatz des Teils gliedert sich in 4 verschiedene Schlüsselschritte und wird detailliert beschrieben.

6.1 Grundidee

Die Idee, diese notwendige Gleichung zu erstellen, beginnt mit dem Bau von Schiebefenstern. Denn obwohl die perspektivische Transformation angewendet wurde und die aktuelle Ansicht von oben ist, kann die Strecke immer noch eine gebogene oder gerade Form haben. Angenommen, jede Kurve in einer zweidimensionalen Ebene hat eine Gleichung, die diese Kurve erfüllt. Diese Kurvenpunkte müssen irgendwie identifiziert werden, damit dann eine Gleichung $f(x, y)$, die die Koordinaten

dieser Kurve erfüllt, angewendet und gefunden werden kann. Das ist die Idee des Problems aus mathematischer Sicht. Aufgrund der Vorverarbeitung wird in diesem Schritt ein binäres Bild der Originalbilder eingegeben und die Vogelperspektive angewendet. Mit anderen Worten, es gibt nur zwei Arten von Pixeln in einem Bild: Schwarz und Weiß mit Pixelwerten von 0 beziehungsweise 1. Nur die weißen Pixel werden interessant sein, da nur diese Pixel Informationen enthalten, die identifiziert werden müssen. Die Informationen über die weißen Pixel lassen sich in zwei Arten unterteilen: Informationen über die Position der Pixel in der Spur und Informationen über das Rauschen. Da die Rauschfilterung durch die Vorverarbeitung optimiert wurde, können die verbleibenden Rauschsignale nicht mehr gefiltert werden, sondern müssen irgendwie noch mit „nützlichen" Pixeln aber mit geringem Gewicht kombiniert werden, um die Ausgabe nicht viel zu verändern. Wie oben erwähnt, wird eine einfache, aber nützliche Technik namens "Sliding Window" im System angewendet. In der Informatik ist ein Schiebefenster ein rechteckiger Bereich fester Breite und Höhe, der über ein Bild „gleitet".

Schiebefensterdetektoren finden Objekte in 4 sehr einfachen Schritten und zwar sind:

(1.) Inspizieren jedes Fensters,

(2.) Merkmale im Fenster extrahieren,

(3.) Schiebefernster klassifizieren und akzeptieren, wenn die Punktzahl über dem Schwellenwert liegt,

(4.) Alle Störungen aufräumen (als Nachbearbeitung bezeichnet) [15]

6.2 Parameter anpassen

Die in diesem Abschnitt erwähnten Parameter gehören zu Schiebefenstern. Anders als bei der Anwendung von Schiebefenstern in der erweiterten Objekterkennung sowie der künstlichen Intelligenz können die hier erstellten Schiebefenster nicht in der Größe verändert werden, das heißt es müssen Informationen über die Größe des Schiebefensters gespeichert werden und werden nicht verändert während des Erkennungsscans.

Da die zu durchsuchende Spur die Höhe des Bildes überspannt, wird das Schiebefenster durch die Höhe des Bildes verteilt. In diesem Projekt gibt

es circa 20 Schiebefenster geteilt durch die Höhe bei Eingabe als $(1280 \, x \, 720)$ Bild. Als nächstes müssen die Variablen minpix und margin definiert werden.

Die Variable margin beschreibt die horizontale Länge des Bildes eines gescannten Bereichs oder anders ausgedrückt die Breite des Schiebefensters. Im Algorithmus hier ist der Rand auf etwa 50 Pixel eingestellt. Die Variable minpix beschreibt die Mindestanzahl von Pixeln, die in einem Schiebefenster erforderlich ist, um den Test zu bestehen. Dies dient dazu, die Bearbeitungszeit zu verkürzen. Wenn jedes Schiebefenster gescannt wird und den Algorithmus auf jedes Schiebefenster angewendet wird, wird es viel länger dauern, als nur die notwendigen Schiebefenster anzuwenden.

6.3 Scanprozess

Angenommen, die Spur ist in zwei ungefähr symmetrische Teile über die beiden Bildhälften unterteilt. Beide bis Hälften des Bildes werden parallel gescannt. Die hier verwendete Methode ist die Schiebefenstermethode. Wenn diese Schiebefenster jedoch gleichzeitig scannen, werden viele Hardwareressourcen wie Speicher oder Strom verbraucht. Dies ist ein wichtiger Faktor, der berücksichtigt werden muss, wenn die Arbeitsumgebung aus kleinen Geräten wie dem Jetson Nano besteht. Außerdem ist es sehr schwierig, sie zu kombinieren oder die Informationen zu filtern, um die Kurvengleichung zu berechnen, nachdem genügend Daten von all diesen Gleitfenstern gesammelt wurden. Daher wurde in diesem Projekt eine einfachere, geeignetere Methode verwendet. Begonnen mit den ersten beiden Schiebefenstern, deren Positionen $[x1, y1]$ im Bereich $[0 : 640, 720 : 694]$ und $[x2, y2]$ im Bereich $[640 : 1280, 720 : 694]$ liegen. Diese beiden Schiebefenster befinden sich am "unten im Bild". Im nächsten Schritt wird das Histogramm angewendet, um den Peak der bewegten Region jedes Bildes zu finden (hier liegt die Koordinaten der bewegten Region im Bereich $[0 : 640, 720 : 694]$ und $[640 : 1280, 720 : 694]$).

Mit der Gleichung `get_hist()` wird der maximale Punkt der beiden Seiten des Bildes gefunden und ihre Koordinaten werden gespeichert. Die maximalen Werte, die jeder Seite des Histogramms entsprechen, werden gefunden und dann werden diese Werte als die anfänglichen x-Werte für die ersten beiden Gleitfenster verwendet. Zusammengefasst lauten die definierten Koordinaten der ersten beiden Schiebefenster wie folgt:Die Größe jeder Box wurde bestimmt als $(H,B) = (36, 50)$

- Mittelpunktskoordinaten jeder Box: Die y-Achsen-Koordinaten der Box werden berechnet als:

$$(720 - (n - 0.5) * window\_height)$$

mit n ist der Anzahl der Schiebefernster und window_height ist die Höhe eines Boxes und zwar gleich $(720 / n)$. Die x-Achsen-Koordinaten dieser beiden Schiebefenster werden durch die `get_hist()`-Gleichung bestimmt: Dies sind die Koordinaten der Maximum-Punkte auf jeder Seite des Bildes. Diese Maximalpunkte stellen die Punkte mit der höchsten Pixeldichte dar und das sind die Schienen.

Nachdem nun die Koordinaten dieser beiden Schiebefenster bestimmt wurden, besteht der nächste Schritt darin, die Pixel in jedem Fenster abzutasten. Die Koordinaten der in jedem Fenster gefundenen Pixel werden als Liste mit den Listennamen `good_left_index` oder `good_right_index` im Speicher abgelegt. Als nächstes müssen die nächsten beiden Fenster identifiziert werden. Die Koordinaten der y-Achse dieser beiden Fenster werden auf der Grundlage von Formel (1) wie folgt berechnet:

$$720 - (19-0.5) * 720 / 20 = 54$$

Die Koordinaten der x-Achse werden auf der Grundlage der Informationen aus dem darunter liegenden Schiebefenster berechnet. Unter der Annahme, dass das darunter liegende Fenster eine minpix-Anzahl von Pixeln enthält (hier ist zum Beispiel 20), wird die x-Koordinate des nächsten Fensters als Durchschnitt der x-Koordinaten des aktuellen Fensters bestimmt.

Zusammenfassend wird die Scanmethode durch die folgenden Schritte bestimmt:

1. Zunächst werden zwei Schiebefenster am unteren Rand des Bildes definiert.
2. Die nächsten beiden Schiebefenster werden basierend auf den Informationen, die von den beiden darunter liegenden Schiebefenstern gesammelt wurden, identifiziert.
3. So fortfahrend werden alle zu durchsuchenden Boxen bestimmt.

6.4 Verarbeitung erfasster Pixel, Kurvenanpassung

Alle Schiebefenster sowie die Position und die Anzahl der von Null verschiedenen Pixel, die in jedem dieser Schiebefenster vorhanden sind, sind nun bestimmt. Um jedoch eine Gleichung für die Kurve einer Eisenbahnlinie aufzustellen, muss ein Zwischenschritt gemacht werden. Das heißt, es müssen die Listen `left_lane_index` und `right_lane_index` verkettet werden und dann die Pixelpositionen der linken und rechten Zeile extrahiert werden. Der Grund für diesen "redundanten" Vorgang hat mit der Datenverarbeitung in der Software-Sprache Python zu tun. In der Liste enthalten `left_lane_index` und `right_lane_index` die Elemente, die die Koordinaten der von Null verschiedenen Pixel in jedem Schiebefenster sind, zum Beispiel:

$$a = ([[1, 2], [3, 4]], [[5, 6]])$$

oder eine grafische Darstellung wie in Abbildung 23:

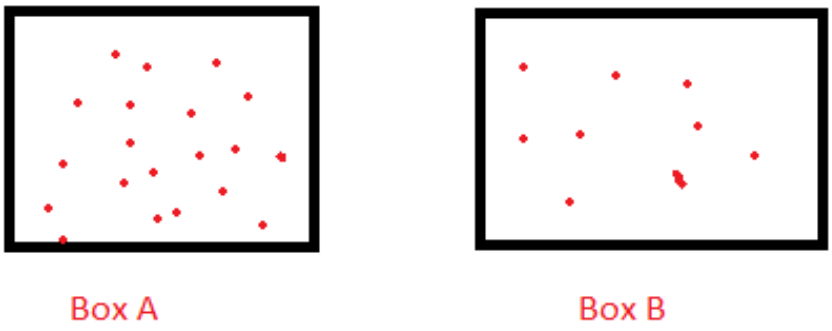


Abbildung 23: Darstellung der Pixels in einem Schiebefernster

Da die Anzahl der Pixel ungleich Null in jedem Gleitfenster unterschiedlich ist, ist auch die Anzahl der in jedem Element enthaltenen Koordinatenvektoren unterschiedlich und es wird schwierig sein, Informationen über diese Punkte zu extrahieren. Durch die Verwendung der Funktion `np.concatenate()`, die in der numpy-Bibliothek verfügbar ist, wird das Problem gelöst.

Als nächstes werden die von Null verschiedenen Pixel im gesamten Bild sortiert und in die Liste `nonzero[]` gespeichert. Es ist leicht zu erkennen, dass die Liste `left_lane_index` und `right_lane_index` eine Teilmenge von `nonzero[]` sind, da zusätzlich außer die von Null verschiedenen Pixel,

die in den Gleitfenstern enthalten sind, andere von Null verschiedenen Pixel außerhalb vorhanden sind. Schließlich werden die Listen `left_lane_index` und `right_lane_index` mit `nonzero[]` verglichen, um die endgültigen Pixelpositionen der linken und rechten Zeile abzuleiten.

Als nächstes wird aus der NumPy-Bibliothek die Polyfit-Funktion aufgerufen. Die Funktion `np.polyfit()` benötigt folgende Parameter: Koordinaten von x und y und Grad des Polynoms. Die Koordinaten von x und y werden aus `left_x`, `left_y`, `right_x` und `right_y` genommen und der Grad des Polynoms wird zu zwei gewählt. Mit Polynom der 2.Ordnung hat die Kurvengleichung die Form

$$a * x^2 + b * x + c$$

wobei a, b, c die Parameter sind (a = 0 wenn die Kurve eine Gerade ist)

Wenn man sich die vorherigen Schritte ansieht, ist leicht zu erkennen, dass es zwar eine Menge notwendiger Punkte gibt, diese Punkte jedoch immer noch diskrete Punkte sind. Das heißt, theoretisch sollte die Ausgabe die Gleichung einer einzelnen Kurve sein, die einem eindeutigen Satz von Punkten entspricht, die die Gleichung dieser Kurve erfüllen. In Abbildung 24 wird das Problem beschrieben:

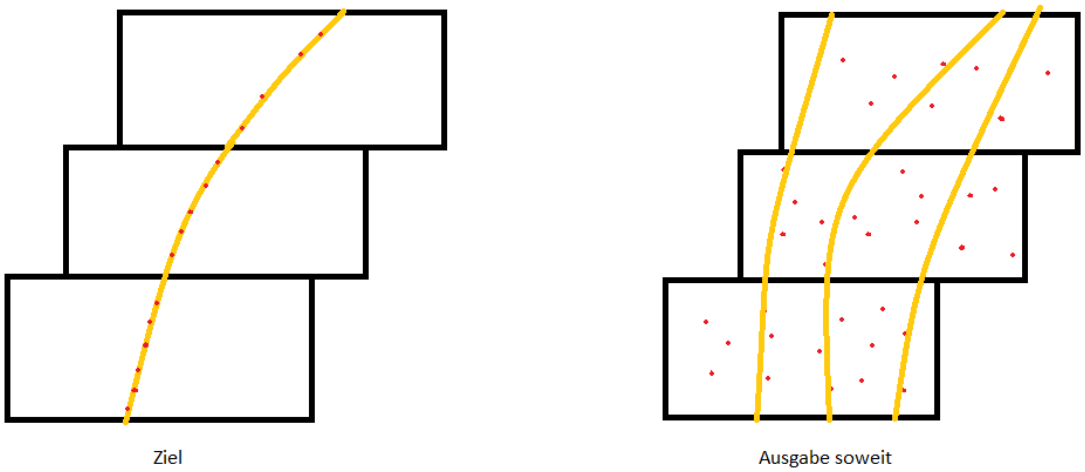


Abbildung 24: Darstellung der Zielkurven und Ergebniskurven

Es gibt zwei Möglichkeiten, diese Aufgabe zu bearbeiten, nämlich die Verarbeitung der gesammelten Punkte oder die Verarbeitung der Kurven. Wenn die erste Methode gewählt wurde, dann müssen diese Punkte von Anfang verarbeitet werden, so dass bei der Ausführung der Funktion `np.polyfit()` eine einzige Kurve erhalten wird. Da der

Informationsverlust nach jedem Schritt des Prozesses zunimmt, kann dies dazu führen, dass die falsche Kurve gewählt wird oder nicht genügend Informationen vorhanden sind, um die Kurve zu bestimmen. Eine rein mathematische Interpretation wird in den nachfolgenden Dokumenten beschrieben.

Die zweite Methode wird wie folgt angewendet: Die letzten zehn Elemente jeder Liste der Variablen a, b und c werden gemittelt. Die Berücksichtigung der letzten zehn Werte jeder Variablen spart Hardware-Ressourcen, die in Berechnungen verwendet werden, und ist sehr wichtig, wenn die Hardware-Ressourcen in diesem Projekt relativ begrenzt sind, wie beim Jetson Nano.

Der letzte Schritt wäre, x- und y-Werte zum Plotten zu generieren. Der gesamte Code wird in Anhang B beigefügt.

7 Endergebnis

Hier ist das Ergebnis der Anwendung des Algorithmus auf mehrere verschiedene Beispiele

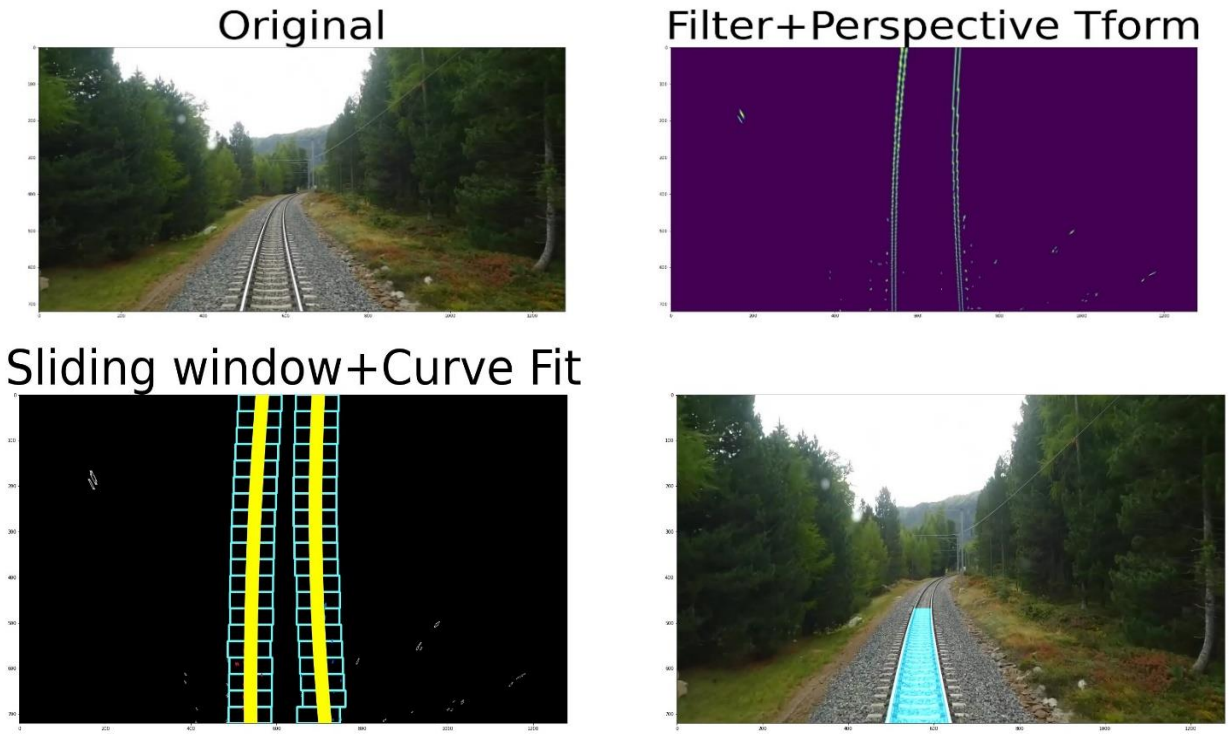


Abbildung 25: Ergebnis von dem Algorithmus

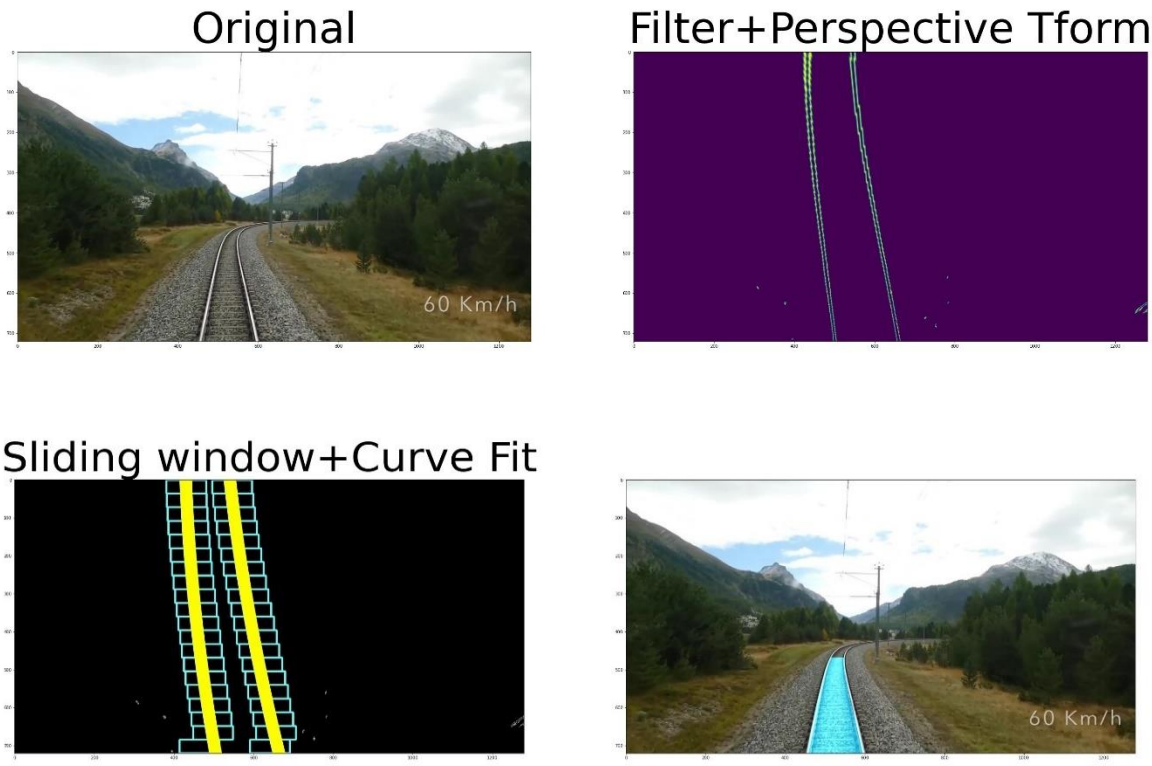


Abbildung 26: Anderes Ergebnis

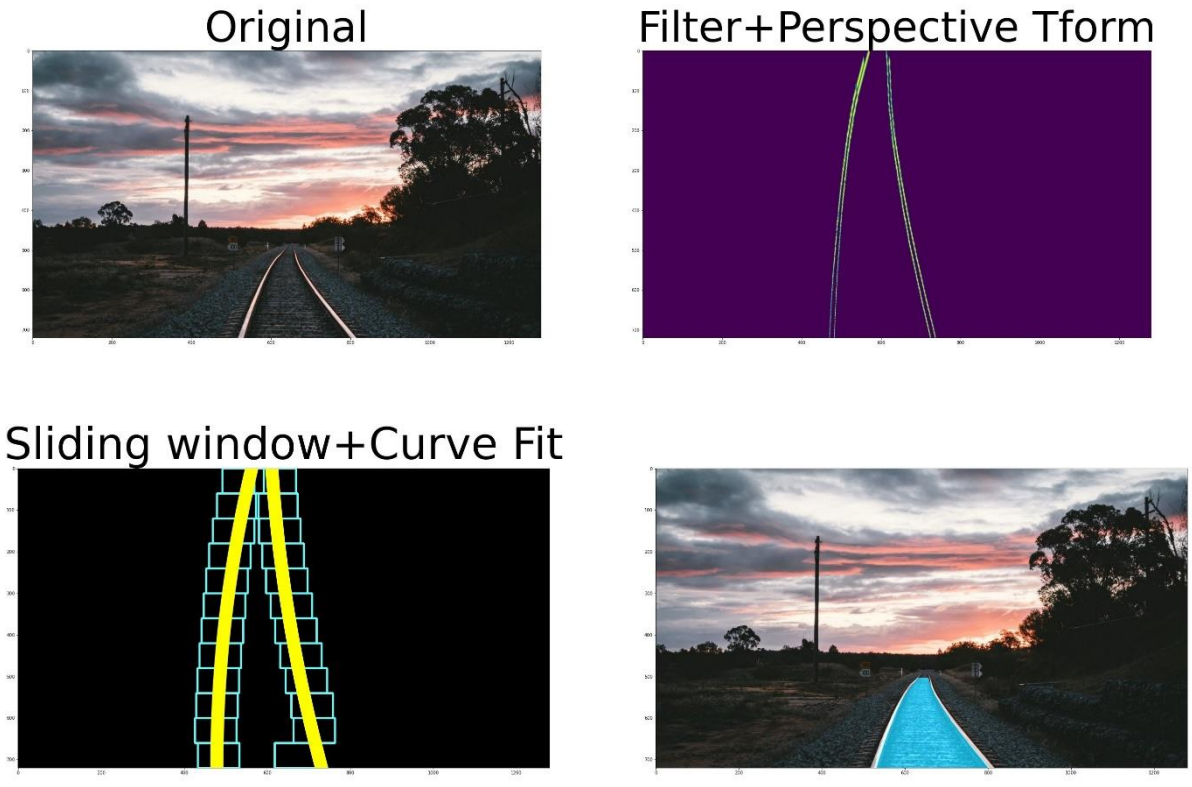


Abbildung 27: Anderes Ergebnis

4 Bewertung und Ausblick

Objekterkennung: Das Modell zeigte gute Leistungen bei der Erkennung von Objekten in Bildern und aufgezeichneten Videos. Allerdings gibt es noch Unterschiede in der Leistung in der realen Situation. Einige davon könnten die Zuverlässigkeit des Systems unter verschiedenen Wetterbedingungen oder die Ausdauer des Systems auch über einen langen Zeitraum hinweg sein. Das Potenzial der Entwicklung auf der Grundlage des Modells ist groß und kann in vielen verwandten Bereichen genutzt werden. Eines davon ist, dass es mit dem Gleiserkennungsalgorithmus kombiniert werden kann, um Teile des Lokomotivalarmsystems aufzubauen.

Erkennung von Spuren: Obwohl die Tests mit Standbildern eine gute Leistung zeigten, schien das gesamte Programm bei Testvideos noch Probleme zu haben. Mögliche Ursachen sind zu lange Programmlaufzeit, der Eingabewert der Transformationsmatrix und andere Parameter sind fest oder die Geschwindigkeit der Hardware.

Es gibt jedoch noch viel Potenzial und Funktionen, die diesem Programm hinzugefügt werden können, zum Beispiel: Schätzen des Kurvenradius während der Fahrt eines Zuges. Dabei werden Bremszeit, Bremskraft und sicherer Bremsweg berechnet. In Kombination mit Objekterkennungssoftware und -hardware wie Infrarotkameras oder Laserkameras können zudem Personen oder Fahrzeuge im gefährlichen Abstand des Zuges identifiziert und Sicherheitswarnungen ausgeben werden.

Abkürzungsverzeichnis

fps	frame per second
GPU	Graphic Processing Units
GUI	Graphical user interface
TanH oder tanh	Hyperbolische Tangensfunktion
IoT	Internet of Things
KI	Künstliche Intelligenz
ReLU	Rectified Linear Unit
SSD	Single Shot Detector
YOLO	You Only Look Once

Abbildungsverzeichnis

Abbildung 1: Ausführungsreihenfolge für die Gleisbahnerkennung 10
Abbildung 2: Darstellung von einem einfachen neuralen Netz [16] 17
Abbildung 3: Darstellung von Sigmoids Funktion 18
Abbildung 4: Darstellung der Tanhs Aktivierungsfunktion 19
Abbildung 5: Darstellung von ReLUs Aktivierungsfunktion 20
Abbildung 6: Darstellung von Leaky ReLUs Aktivierungsfunktion für a = 0.1 20
Abbildung 7: Verlauf von einem einfachen Netz 23
Abbildung 8: Verlauf von einem neutralen Netz 24
Abbildung 9: Darstellung der konventionelle Faltung [11] 27
Abbildung 10: Darstellung der Tiefenfaltung (Depthwise Convolution) [11] 28
Abbildung 11: Darstellung der Punktweise Faltung [11] 30
Abbildung 12: Grafische Darstellung eines Restblocks [11] 32
Abbildung 13: Architektur des SSD-MobilenetV2 [13] 34
Abbildung 14: Ergebnis von dem Training 39
Abbildung 15: Darstellung des Schachbretts für Projektanwendung 41
Abbildung 16: Prozessablaufbeschreibung der Kamerakalibrierung 42
Abbildung 17: Darstellung der Gaußschen Matrix [17] 44
Abbildung 18: Non- Max-Supression Filterschwelle [17] 47
Abbildung 19: Ein Beispiel der Anwendung von Cannys Methode 48
Abbildung 20: Prozessablauf des Verzerrens der Bildperspektive 49
Abbildung 21: Beispiel der Verwendung von „Image Warping Process“ 49
Abbildung 22: Beispielergebnisse von der Anwendung der Funktion `cv2.getPerspectiveTransform()` 51
Abbildung 23: Darstellung der Pixels in einem Schiebefernster 55
Abbildung 24: Darstellung der Zielkurven und Ergebniskurven 56
Abbildung 25: Ergebnis von dem Algorithmus 57
Abbildung 26: Anderes Ergebnis 58
Abbildung 27: Anderes Ergebnis 58

Tabellenverzeichnis

Tabelle 1: Eigenschaften von Fahrbahnen und Gleisen [1] 9
Tabelle 2: Merkmale von Jetson Nano 2GB und Raspberry Pi 4 [3] [4] 14
Tabelle 3: Dimensionierung der Variablen in einem einfachen neuralen Netz 25
Tabelle 4: Erklärung von Funktion `cv.GaussianBlur()` 45
Tabelle 5: Erklärung von Funktion `cv.Canny()` 48
Tabelle 6: Erklärung von Funktion `cv2.getPerspectiveTransform()` 50

Literaturverzeichnis

[1] M. Gschwandtner, W. Pree und A. Uhl, „Track detection for autonomous trains," *International Symposium on Visual Computing,* pp. 19-28, November 2010. Springer, Berlin, Heidelberg, 2010..

[2] G. Baltazar, „CPU vs GPU in Machine Learning," . Abgerufen am 20.12.2021 <https://blogs.oracle.com/ai-and-datascience/post/cpu-vs-gpu-in-machine-learning>

[3] NVIDIA, „NVIDIA Developer," [Online]. Abgerufen am 22.12.2021 <https://developer.nvidia.com/embedded/jetson-nano-2gb-developer-kit>

[4] „Raspberry Pi," [Online]. Abgerufen am 22.12.2021 <https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/>

[5] G. Bradski und A. Kaehler, „OpenCV," *Dr. Dobb's journal of software tools 3 ,* 2000.

[6] M. Bussonnier, „I Python, You R, We Julia". Abgerufen am 22.12.2021 <https://blog.jupyter.org/i-python-you-r-we-julia-baf064ca1fb6>

[7] A. Krizhevsky, I. Sutskever und G. E. Hinton, „ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in neural information processing systems 25 (2012)*, pp. 1097-1105.

[8] A. L. Maas, A. Y. Hannun und A. Y. Ng, „Rectifier nonlinearities improve neural network acoustic models.," *Proc. icml. Vol. 30. No. 1*. 2013..

[9] A. G.Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto und H. Adam, „MobileNets: Efficient Convolutional Neural Network for Mobile Visison Applications," *arXiv preprint arXiv:1704.04861* , 2017.

[10] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov und L.-C. Chen, „MobileNetV2: Inverted and Linear Bottlenecks," *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018.

[11] K. D. Pham, „MobileNet Model," [Online]. Abgerufen am 23.12.2021 <https://phamdinhkhanh.github.io/2020/09/19/MobileNet.html>

[12] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu und A. C.Berg, „SSD: Single Shot MultiBox Detector," *European conference on computer vision.* Springer, Cham, 2016.

[13] D. Franklin, „Re-training SSD Mobilenet". Abgerufen am 24.12.2021 <https://github.com/dusty-nv/jetson-inference/blob/master/docs/pytorch-ssd.md>

[14] P. Khanh DInh, „Model SSD trong Object Detection". Abgerufen am 22.12.2021 <https://phamdinhkhanh.github.io/2019/10/05/SSDModelObjectDetection.html>

[15] S. Fidler, September 9 2019, „SCC420 Intro to Image understanding“ ,University of Toronto Vorlesung 17 Seite 7

[16] R. Whitwam, „Google Neural Network Can Predict Your Health Status From Your Retina“. Aufgerufen am 22.12.2021

[17] Chris828, „Wikipedia,“ [Online]. Abgerufen am 25.21.2021

[18] „theailearner,“ [Online]. Abgerufen am 25.12.2021 <https://theailearner.com/tag/cv2-getperspectivetransform/>

Anhang A: Objekterkennnung

A.1 Docker ausführen

Als erstes muss der Docker ausgeführt werden, um eine Umgebung zu erstellen.

```
$ cd jetson-inference/
$ docker/run.sh
```

A.2 Prüfung der Voraussetzungen

```
$ cd jetson-inference/python/training/detection/ssd

$ wget
https://nvidia.box.com/shared/static/djf5w54rjvpqocsiztzaandq1
m3avr7c.pth -O models/mobilenet-v1-ssd-mp-0_675.pth

$ pip3 install -v -r requirements.txt
```

A.3 Die Daten herunterladen

Überprüfung der Größe der Daten

```
$ python3 open_images_downloader.py --stats-only --class-names
"Person,Car,Land vehicle" --data=data/detection-objects
```

Herunterladen der Daten mit einer Größe von 10000 Bildern

```
$ python3 open_images_downloader.py --max-images=10000 --
class-names "Person,Car,Land vehicle"
```

A.4 Training des Modells

```
$ python3 train_ssd.py --data=data/detetion-objects --model-
dir=models/detection-objects --batch-size=4 --epochs=48
```

A.5 Konvertierung des Modells in ONNX

```
$ python3 onnx_export.py --model-dir=models/detection-objects
```

Anhang B: Gleisbahnerkennung

B.1 Importieren die benötigten Bibliothek

```
### Alle Kommentare werden in Blau geschrieben.
import pickle
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import glob
import cv2
import scipy
import sklearn
import numpy
from sklearn.metrics import mean_squared_error
%matplotlib inline
```

B.2 Kamera kalibrieren

```
### Alle Kommentare werden in Blau geschrieben.

x_cor = 9
y_cor = 6
# Objektpunkte vorbereiten, wie (0,0,0), (1,0,0), (2,0,0)
....,(6,5,0)
objp = np.zeros((y_cor*x_cor,3), np.float32)
objp[:,:2] = np.mgrid[0:x_cor, 0:y_cor].T.reshape(-1,2)

# Arrays zum Speichern von Objektpunkten und Bildpunkten aus
allen Bildern.
objpoints = [] # 3d-Punkte im realen Weltraum
imgpoints = [] # 2d Punkte in der Bildebene
images = glob.glob('camera_cal/calibration*.jpg')
# Erstellen einer Liste von Pfaden zu Kalibrierungsbildern
# Die Liste durchgehen und nach Schachbrettecken suchen
corners_not_found = []
#Kalibrierungsbilder, bei denen opencv keine Ecken finden
konnte
for idx, fname in enumerate(images):
    img = cv2.imread(fname)
    # Umwandlung in Graustufen
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    # Finde die Schachbrettecken
```

```
    ret, corners = cv2.findChessboardCorners(gray,
                    (x_cor,y_cor), None)
    # Wenn gefunden, Objektpunkte und Bildpunkte hinzufügen
    if ret == True:
        objpoints.append(objp)
        imgpoints.append(corners)
        cv2.drawChessboardCorners(img, (x_cor,y_cor), corners,
        ret)
    else:
        corners_not_found.append(fname)

# Test undistortion on an image
img = cv2.imread('camera_cal/calibration1.jpg')
img_size = (img.shape[1], img.shape[0])

# Kamerakalibrierung anhand von Objektpunkten und Bildpunkten
durchführen
ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints,
                               imgpoints, img_size,None,None)

def undistort(img):
    return cv2.undistort(img, mtx, dist, None, mtx)
```

B.3 Gaußscher Unschärfe-Algorithmus und morphologische Transformationen

```
### Alle Kommentare werden in Blau geschrieben.

def pipeline(img):
    img = undistort(img)
    img = np.copy(img)
    img = cv2.GaussianBlur(img,(5,5),0)
    edges = cv2.Canny(img,250,300)
    edges = cv2.morphologyEx(edges,cv2.MORPH_OPEN,(5,5))
    edges = cv2.dilate(edges,(5,5),1)
    return edges
```

B.4 Perspective Warping

```
### Alle Kommentare werden in Blau geschrieben.

def perspective_warp(img,
                     dst_size=(1280,720),
```

```
                    src =
          np.float32([(0.3,0.65),(0.6,0.65),(0,1),(0.9,1)]),
                    dst =
          np.float32([(0,0), (1, 0), (0,1), (1,1)])):
    img_size = np.float32([(img.shape[1],img.shape[0])])
    img_size = np.float32([(img.shape[1],img.shape[0])])
    src = src* img_size
    dst = dst * np.float32(dst_size)
    M = cv2.getPerspectiveTransform(src, dst)
    # Warp the image using OpenCV warpPerspective()
    warped = cv2.warpPerspective(img, M, dst_size)
    return warped

def inv_perspective_warp(img,
                    dst_size=(1280,720),
                    dst =
        np.float32([(0.3,0.65),(0.6,0.65),(0,1),(0.9,1)]),
                    src =
        np.float32([(0,0), (1, 0), (0,1), (1,1)])):
    img_size = np.float32([(img.shape[1],img.shape[0])])
    src = src* img_size
    dst = dst * np.float32(dst_size)
    M = cv2.getPerspectiveTransform(src, dst)
    warped = cv2.warpPerspective(img, M, dst_size)
    return warped
```

B.5 get_hist() -Funktion

```
### Alle Kommentare werden in Blau geschrieben.

def get_hist(img):
    hist = np.sum(img[img.shape[0]//2:, :], axis=0)
    return hist
```

B.6 Schiebefernster

```
### Alle Kommentare werden in Blau geschrieben.

left_a, left_b, left_c = [],[],[]
right_a, right_b, right_c = [],[],[]

def sliding_window(img, nwindows=20, margin=50, minpix = 20,
                   draw_windows=True):
```

```
    global left_a, left_b, left_c,right_a, right_b, right_c
    left_fit_= np.empty(3)
    right_fit_ = np.empty(3)
    out_img = np.dstack((img, img, img))*255

    histogram = get_hist(img)

    # Ermittlung der Spitzenwerte der linken und rechten
      Hälfte
    midpoint = int(histogram.shape[0]//2)
    leftx_start = np.argmax(histogram[:midpoint])
    rightx_start = np.argmax(histogram[midpoint:]) + midpoint

    # Höhe der Fenster einstellen
    window_height = np.int(img.shape[0]/nwindows)

    # Identifizieren Sie die x- und y-Positionen aller Pixel
      im Bild, die nicht Null sind.
    nonzero = img.nonzero()
    nonzero_y = np.array(nonzero[0])
    nonzero_x = np.array(nonzero[1])

    # Die aktuellen Positionen werden für jedes Fenster
      aktualisiert.
    leftx_current = leftx_start
    rightx_current = rightx_start

    # Leere Listen für die Pixelindizes der linken und rechten
    Fahrspur erstellen.
    left_lane_index = []
    right_lane_index = []

    # Schritt für Schritt durch die Fenster
    for window in range(nwindows):
        # Identifizieren der Fenstergrenzen in x und y (und
          rechts und links)
        window_y_low = img.shape[0] - (window+1)*window_height
        window_y_high = img.shape[0] - window*window_height
        window_xleft_low = leftx_current - margin
        window_xleft_high = leftx_current + margin
        window_xright_low = rightx_current - margin
        window_xright_high = rightx_current + margin
        # Zeichnen Sie die Fenster auf das
          Visualisierungsbild.
```

```
        if draw_windows == True:
            cv2.rectangle(out_img,
            (window_xleft_low,window_y_low),
            (window_xleft_high,window_y_high),
            (100,255,255), 3)

            cv2.rectangle(out_img,
            (window_xright_low,window_y_low),
            (window_xright_high,window_y_high),
            (100,255,255), 3)

         # Identifizierung der Non-Zeros-Pixels in x und y
           innerhalb des Fensters
        good_left_index = ((nonzero_y >= window_y_low)&
                           (nonzero_y  < window_y_high) &
                           (nonzero_x >= window_xleft_low) &
                    (nonzero_x <window_xleft_high)).nonzero()[0]

        good_right_index = ((nonzero_y >= window_y_low) &
                            (nonzero_y < window_y_high) &
                            (nonzero_x >= window_xright_low) &
                   (nonzero_x < window_xright_high)).nonzero()[0]

        # Anhängen dieser Indizes an die Listen
        left_lane_index.append(good_left_index)
        right_lane_index.append(good_right_index)

        # Wenn man > minpix Pixel gefunden hat, zentriert man
          das nächste Fenster auf deren mittlere Position.
        if len(good_left_index) > minpix:
            leftx_current =
            np.int(np.mean(nonzero_x[good_left_index]))
        if len(good_right_index) > minpix:
            rightx_current =
            np.int(np.mean(nonzero_x[good_right_index]))

    # Verketten die Arrays der Indizes
    left_lane_index = np.concatenate(left_lane_index)
    right_lane_index = np.concatenate(right_lane_index)

    # Extract left and right line pixel positions
    left_x = nonzero_x[left_lane_index]
    left_y = nonzero_y[left_lane_index]
    right_x = nonzero_x[right_lane_index]
```

```
    right_y = nonzero_y[right_lane_index]

    # Anpassung eines Polynoms zweiter Ordnung an jedes
      Kurve
    left_fit = np.polyfit(left_y, left_x, 2)
    right_fit = np.polyfit(right_y, right_x, 2)

    left_a.append(left_fit[0])
    left_b.append(left_fit[1])
    left_c.append(left_fit[2])

    right_a.append(right_fit[0])
    right_b.append(right_fit[1])
    right_c.append(right_fit[2])

    left_fit_[0] = np.mean(left_a[-10:])
    left_fit_[1] = np.mean(left_b[-10:])
    left_fit_[2] = np.mean(left_c[-10:])

    right_fit_[0] = np.mean(right_a[-10:])
    right_fit_[1] = np.mean(right_b[-10:])
    right_fit_[2] = np.mean(right_c[-10:])


    # x- und y-Werte für die grafische Darstellung generieren
    ploty = np.linspace(0, img.shape[0]-1, img.shape[0] )
    left_fit_x = left_fit_[0]*ploty**2 + left_fit_[1]*ploty +
    left_fit_[2]
    right_fit_x = right_fit_[0]*ploty**2 + right_fit_[1]*ploty
    + right_fit_[2]

    out_img[nonzero_y[left_lane_index],
    nonzero_x[left_lane_index]] = [255, 0, 100]

    out_img[nonzero_y[right_lane_index],
    nonzero_x[right_lane_index]] = [0, 100, 255]

    return out_img, (left_fit_x, right_fit_x), (left_fit_,
right_fit_), ploty
```

B.7 Plotten

```
### Alle Kommentare werden in Blau geschrieben.

def draw_lanes(img, left_fit, right_fit):
    ploty = np.linspace(0, img.shape[0]-1, img.shape[0])
    color_img = np.zeros_like(img)
    left_side = np.array([np.transpose(np.vstack([left_fit,
    ploty]))])
    right_side =
    np.array([np.flipud(np.transpose(np.vstack([right_fit,
    ploty])))])

    points = np.hstack((left_side, right_side))
    cv2.fillPoly(color_img, np.int_(points), (0,200,255))
    inv_perspective = inv_perspective_warp(color_img)
    inv_perspective = cv2.addWeighted(img, 1, inv_perspective,
    0.7, 0)
    return inv_perspective
```