Technology
Arts Sciences
TH Köln

Technische Hochschule Köln
Fakultät für Informatik und Ingenieurwissenschaften

Praxisprojekt WS18/19 - Dokumentation

Entwicklung einer Objekterkennung für Regal-Systeme, mit Hilfe von Bildverarbeitung, in einem neuronalen Netz

Vorgelegt an der TH Köln
Campus Gummersbach
im Studiengang
Medieninformatik

ausgearbeitet von:
Torben Krause
(Matrikelnummer: 11106885)

Prüfer: Prof. Dr. Martin Eisemann

Gummersbach, 28.12.2018

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Motivation

Das richtige Verwalten von Lebensmitteln, welche gekühlt im Kühlschrank aufbewahrt werden müssen, ist nicht immer leicht. Oft müssen schlecht gewordene Lebensmittel weggeschmissen werden, weil sie vergessen werden oder es fehlen Lebensmittel, welche gebraucht werden. Gerade in Haushalten mit mehreren Personen, wie zum Beispiel Familien oder Wohngemeinschaften, ist das Kühlschrankmanagement schwierig, da nicht jeder Bewohner weiß, welche Lebensmittel aus dem Kühlschrank genommen wurden und welche hineingelegt werden. Der häufigste Grund ist dabei die unzureichende Kommunikation unter den Bewohnern. Aus diesem Grund werden schnell verderbende Lebensmittel leicht vergessen oder Produkte durch fehlerhafte Absprache mehrfach oder nicht gekauft.
Eine Idee um dieses Problem zu lösen ist den Kühlschrankinhalt mithilfe einer Kamera, welche im Inneren des Kühlschranks angebracht ist, aufzuzeichnen und die enthaltenden Lebensmittel mit einem neuronalen Netz zu klassifizieren. Aus den Ergebnissen der Analyse wird nun eine Produktliste vom Inneren des Kühlschranks erstellt und anschließend in einer Datenbank gespeichert. Über eine mobile Anwendung können die erhobenen Daten dem Benutzer zur Verfügung gestellt werden.

1.1 Zielsetzung

Ziel dieses Projektes ist die Entwicklung eines künstlichen neuronalen Netzes, welches mehrere vordefinierte Klassen zuverlässig erkennt und zuordnen kann. Das Netz soll außerdem so konzipiert werden, dass Produkte unabhängig ihrer Position im Bild erkannt werden können. Desweiteren soll das neuronale Netz auch mit leicht verdeckten Objekten zurecht kommen.

1.2 Anforderungen

In diesem Kapitel sollen die Anforderungen an das zu entwickelnde künstliche neuronale Netz, mit Hilfe der Anforderungsschablonen von „Die SOPHISTen“ [SOP13], definiert werden.

1.2.1 Funktionale Anforderungen

- Das System muss fähig sein Bilder zu verarbeiten.
- Das System muss fähig sein die trainierten Klassen zu erkennen und einzuordnen.
- Das System soll fähig sein mehrere Objekte auf einem Bild gleichzeitig zu erkennen und deren Position festzustellen.

1.2.2 Organisationale Anforderungen

- Das System wird so gestaltet sein, dass die existierenden Klassen unkompliziert erweitert werden können.
- Das System wird so gestaltet sein, dass es mit wenig Aufwand betrieben werden kann.
- Das System soll so gestaltet sein, dass es einfach auf ein Endgerät transferiert werden kann.
- Die Installation benötigter Module soll für den Benutzer so einfach wie möglich gemacht werden.

1.2.3 Qualitative Anforderungen

- Das zu trainierende Netz muss eine möglichst hohe Genauigkeit aufweisen.
- Das zu entwickelnde Netz soll auf den aktuellen Plattformen arbeiten und neue Technologien unterstützen.
- Das System soll so ausgelegt werden, dass die Geschwindigkeit der Objekterkennung möglichst hoch ist.
- Das System soll fähig sein leicht verdeckte Objekte richtig zu identifizieren.
- Das System soll fähig sein bei schlechten Bildverhältnissen gute Ergebnisse zu liefern.

2 Neuronale Netze

Die Bezeichnung „neuronale Netze“ kommt aus der Biologie und beschreibt die Funktionsweise des menschlichen Gehirns. Das Gehirn eines Menschen besteht aus ungefähr 10^{11} Nervenzellen [Ert13, 265ff.], den so genannten Neuronen. Diese sind untereinander verbunden und tauschen Informationen aus. Wird eine bestimmte Reizschwelle überschritten, werden Signale an das nächste Neuron weitergegeben. Ein neuronales Netz kann somit als eine Verbindung aus vielen Nervenzellen bezeichnet werden, welche Informationen verarbeiten und weiterleiten. Das neuronale Netz ist unter anderem für das menschliche Lernen zuständig.

2.1 Künstliche neuronale Netze

Ein künstliches neuronales Netz ist dementsprechend eine Nachahmung eines menschlichen Gehirns, welches auf Grundlage mathematischer Formeln arbeitet. Auf die verwendeten Formeln wird in dieser Arbeit nicht eingegangen, sondern auf die Funktionsweise von künstlichen neuronalen Netzen [Ert13, S. 247–285]. Künstliche neuronale Netze (KNN) stellen einen Bestandteil des maschinellen Lernens dar. Oft werden KNN für die Klassifizierung von Datensätzen (Bildern, Statistiken, Profile in sozialen Netzwerken) verwendet. Das Ergebnis der Klassifikation ist eine Zugehörigkeitswahrscheinlichkeit der Eingabedaten zu den definierten Klassen.

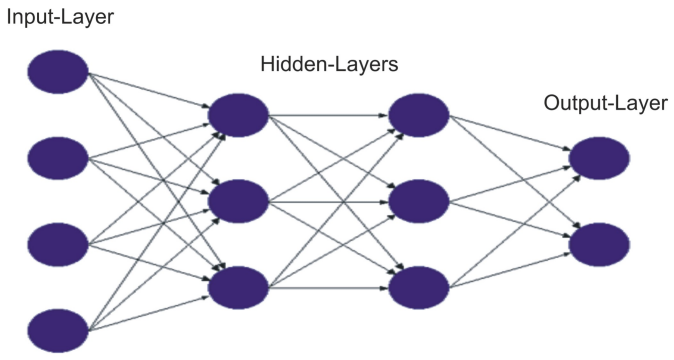

Abbildung 1: Schematischer Aufbau eines künstlichen neuronalen Netzes [Ang18]

Der Aufbau eines künstlichen neuronalen Netzes kann grundsätzlich in drei Schichten aufgeteilt werden (Abbildung 1). Die erste Schicht ist der „Input Layer“. Dieser „Layer“ bekommt alle Eingangsdaten von einem beliebigen Datensatz. Jedes Pixel auf einem Bild wird mit einem Neuron verbunden, welche sich in der Gewichtung und dem Schwellwert unterscheiden können. Wird der Schwellwert überschritten, gibt das Neuron die verarbeiteten Informationen an die nächste Schicht weiter. Die zweite Schicht besteht aus einem

oder mehreren „Hidden Layer“. Jeder „Layer“ ist in dieser Schicht auf verschiedene Merkmale trainiert. Anfangs werden nur grobe Strukturen erkannt, wie beispielsweise Geraden oder Kanten. Diese Strukturen werden detaillierter, desto mehr „Layer“ durchlaufen werden. Die dritte Schicht besteht aus dem „Output Layer“. Die Muster, die in den vorherigen Layern erkannt und zusammengesetzt wurden, vergleicht der „Output Layer“ mit den trainierten Klassen und gibt eine prozentuale Wahrscheinlichkeit aus, welche Klassen in dem Bild enthalten sind.

2.2 Rückpropagierung

Die Rückpropagierung, oder im Englischen auch „Backpropagation“ [Ert13] ist Teil des überwachten Lernens. Diese Vorgehensweise beschreibt das Lernen aus Fehlern. Es werden Trainingsdaten verwendet, welche feste Klassifizierungen als Zielwerte haben. Eingabedaten welche durch ein neuronales Netz verarbeitet werden, liefern zunächst einen Ausgabewert mit einer Identifizierung und Wahrscheinlichkeit. Dieser wird auf Grundlage des Zielwertes verglichen. Aus der Abweichung von Ausgabewert und Zielwert, wird ein Fehler berechnet. Daraus wird ermittelt welches Neuron den größten Einfluss auf die fehlerhafte Ausgabe hatte. Die Gewichtung des betroffenen Neurons wird anschließend angepasst, damit sich der Fehlerwert verringert [Goo+16].

2.3 Faltende neuronale Netze

Anders als bei einem KNN, welches auf Grundlage des menschlichen Gehirns entworfen wurde, orientiert sich ein faltendes neuronales Netz, auch „Convolutional Neural Network“ (CNN) genannt, an dem Konzept des menschlichen Sehens [LeC+89]. Bei einem CNN werden maschinell kleine Bereiche visueller Informationen so simuliert, als wären sie benachbarte Sehnerven im Auge. Gerade für die Verarbeitung von Bilddaten werden häufig „Convolutional Neural Networks“ eingesetzt [Goo+16, S. 326]. Voraussetzung für die Verwendung eines CNN ist die matrixartige Anordnung der Datenpunkte, wie beispielsweise bei einem digitalen Bild.

Bei einem „Convolutional Layer“ (Abbildung 2) wird über die Pixel des Eingabebildes eine Schablone gelegt. Die Schablone wird je nach Bildgröße mehrfach horizontal, wie auch vertikal verschoben [Goo+16, S. 327–335]. Diese Layer haben meist eine ungerade quadratische Abmessung (3x3, 5x5, 7x7). Die frühen Schichten bilden zunächst ein Skalarprodukt der betroffenen Pixel und können dadurch grobe Kanten erkennen. Jeder Layer ist dabei auf eine bestimmte Art von Mustern trainiert. Je mehr Layer durchlaufen werden, desto genauer werden die Muster, welche das neuronale Netz erkennt.

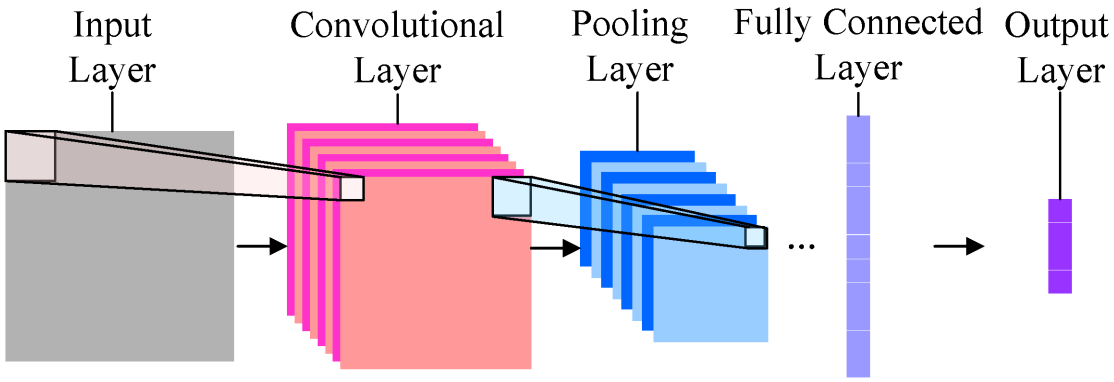

Abbildung 2: Funktionsweise eines Convolutional Neural Network [Pen+16]

Der „Pooling Layer“, oder auf Deutsch Vereinfachungsschicht, (Abbildung 2) funktioniert ähnlich wie der „Convolutional Layer“ [Goo+16, 336f.]. Auch bei dieser Methode wird eine Schablone über die Pixel des Bildes gelegt und horizontal wie auch vertikal verschoben. Hierbei werden die Farbwerte der beteiligten Pixel verarbeitet. Es kann entweder der höchste Farbwert eines Pixels übernommen oder der durchschnittliche Farbwert berechnet werden. Die letzten Schichten des Netzes bestehen aus dem „Fully Connected Layer“, welcher den normalen Aufbau eines künstlichen neuronalen Netzes darstellt [LeC+89, S. 14]. Das bedeutet alle Ausgaben werden, wie bei einem KNN, mit jedem Neuron verbunden. Die letzte Schicht übergibt seine Ausgaben an den Output Layer, welcher die Zuordnungswahrscheinlichkeiten der Objekte zu den trainierten Klassen berechnet.

3 Durchführung

In diesem Kapitel wird die praktische Durchführung beschrieben und die getroffene Entscheidungen werden erläutert. Zu Beginn werden die verwendeten Frameworks und die Entwicklungsumgebung vorgestellt. Da für das Erstellen eines neuronalen Netzes eine hohe Anzahl an Trainingsdaten, sowie viel Rechenleistung benötigt wird, folgt eine Evaluation von vortrainierten Netzen, welche für dieses Projekt verwendet werden sollen. Im Anschluss werden die zu verwendenden Klassen und Trainingsdaten erstellt. Abschließend werden die ausgewählten Netze mit den generierten Trainingsdaten trainiert.

3.1 Entwicklungsumgebung

Um ein bestehendes neuronales Netz neu zu trainieren, werden neben dem neuronalen Netz weitere Frameworks und eine Entwicklungsumgebung benötigt. In den kommenden Unterpunkten werden die Entwicklungsumgebung sowie alle benötigten Bibliotheken aufgeführt.

3.1.1 Verwendete Hardwarekonfiguration

Das Netz wurde auf einem Windows 8.1 Betriebssystem trainiert. Berechnet wurden die Trainingsschritte von einer Nvidia GeForce GTX 980 GPU mit 4GB VRAM (Grafikspeicher), einer Intel Core i5 CPU und 16GB RAM (Arbeitsspeicher). Für einen Trainingsschritt, mit der verwendeten Konfiguration, benötigt die Grafikkarte durchschnittlich 250ms-600ms. Im Vergleich hat die CPU, mit 2-6 Sekunden pro Schritt, ungefähr die achtfache Zeit benötigt. Durch die Optimierung der Matrixmultiplikationen, liegt der Vorteil eindeutig bei der Grafikkarte. Aus diesem Grund wird diese für das Training verwendet.

3.1.2 Anaconda Python

Die später aufgeführten CNN's sowie die Tensorflow API wurden in der Programmiersprache Python erstellt, weshalb diese auch als Programmmiersprache ausgewählt wurde. Des Weiteren bietet Python eine Vielzahl von verfügbaren Bibliotheken an, welche für das Erstellen der Trainingsdaten und das Trainieren der Netze benötigt werden. Anaconda [Hea18] ist eine „Open-Source-Distribution“ für die Programmiersprache Python. Diese ermöglicht es einfach zwischen den verschieden Python-Versionen zu wechseln.
In diesem Projekt wurde mit der Python Version 3.6.7 gearbeitet.

- Installation von Python 3.6.7: conda create -n <Name> python=3.6.7

3.2 Frameworks

Um ein künstliches neuronales Netz zu trainieren, werden ein Framework und mehrere Programm-Bibliotheken benötigt. Die Auswahl wird im Folgenden aufgezählt und kurz erläutert.

3.2.1 Tensorflow

Bei Tensorflow [LLC18c] handelt es sich um eine plattformunabhängige Programmbibliothek unter Open-Source-Lizenz, die sich für Aufgaben rund um maschinelles Lernen und künstliche Intelligenz (KI) einsetzen lässt. Ursprünglich entwickelte Google die Software für den internen Bedarf. Das Framework bietet vielfältige Einsatzmöglichkeiten und gestattet es, lernende neuronale Netze zu erstellen und diese zu verändern. Es zeichnet sich durch seine gute Skalierbarkeit aus und ist auf unterschiedlichen Systemen vom Smartphone bis zu Clustern mit vielen Servern einsetzbar. Außerdem stellt Google verschiedene trainierte Modelle bereit, welche für das Projekt verwendet werden können. Tensorflow kann auf allen CPU's betrieben werden oder mit Nvidia Grafikkarten, die CUDA unterstützen.

- Installation in Python: pip install –upgrade gpu-tensorflow

Zusätzlich wird die API von Tensorflow (Models) für die Objekterkennung verwendet. In dieser wird eine Grundstruktur für das Trainieren eines CNN zur Verfügung gestellt, welche man je nach Verwendungszweck anpassen kann.

3.2.2 Benötigte Bibliotheken

Für das Arbeiten mit Tensorflow und CNN's werden einige spezielle Python-Bibliotheken benötigt. Diese Bibliotheken werden im Folgenden aufgeführt und erläutert. Die Installation geschieht hier über den „Package Installer“ von Python.

Pillow

Pillow ist eine „Imaging Library “ für Python. Diese Bibliothek wird dafür genutzt Bilddateien öffnen, verändern und speichern zu können.

- Installation in Python: pip install –upgrade pillow

lxml

lxml ist eine Weiterführung der XML Sprache für Python. lxml erweitert dabei den Elementebaum von XML stark. Verwendet wird lxml, um die Markierungen von Objektklassen und deren Positionen auszulesen. Durch die Arbeit mit Elementbäumen werden

nur die benötigten Informationen herausgefiltert, woraus ein besseres Laufzeitverhalten resultiert.

- Installation in Python: pip install –upgrade lxml

Cython

Python ist im Vergleich zu C eine recht langsame Programmiersprache, bei der die Ausführungzeit des Programmcodes wesentlich erhöht ist. Cython ist eine Programmbibliothek und auch Programmiersprache, welche die wesentlich langsamere Programmiersprache Python erweitert und in der Ausführung beschleunigt. Dabei wird Python-Code in C kompiliert oder es kann externer Code von C eingebunden werden, um ihn für die Ausführung zu nutzen. Das hat den Vorteil, dass die Geschwindigkeit des Programms erhöht wird. Gerade für das Trainieren künstlicher neuronaler Netze wird viel Rechenleistung benötigt.

- Installation in Python: pip install –upgrade Cython

Jupyter

Jupyter, oder auch „Jupyter Notebook“, ist eine open-source Webapplikation, die es ermöglicht Dokumente, welche Live Code, Abbildungen oder Text beinhalten zu teilen. Dabei können Inhalte von Dateien im Browser aufgerufen und ausgeführt werden. Verwendet wird diese Bibliothek, um die korrekte Installation von Tensorflow zu überprüfen.

- Installation in Python: pip install –upgrade jupyter

Matplotlib

Die Programmbibliothek „matplotlib“ ermöglicht es mathematische Formeln in der Python-Konsole oder im Code zu verarbeiten und zu visualisieren. Matplotlib wird verwendet, um die Erkennung von Objekten und deren Positionen auf einem Bild dazustellen.

- Installation in Python: pip install –upgrade matplotlib

Pandas

Pandas ist eine Programmbibliothek von Python. Der Name stammt von „Paneldata“, was eine Bezeichnung von Datensätzen ist. Diese Bibliothek ermöglicht eine einfache Selektierung und Indizierung von Daten. In diesem Projekt wird Pandas für die Erhebung von Informationen aus den Trainingsdaten und für die Generierung von Graphen verwendet.

- Installation in Python: pip install –upgrade pandas

OpenCV-Python

OpenCV steht für „Open Source Computer Vision Library“ und ist eine freie Programmbibliothek für maschinelles Sehen. Diese Bibliothek ermöglicht die Aufteilung eines Bildes in verschiedene Bildbereiche. Außerdem können weitere Bildoptimierungen vorgenommen werden, wie zum Beispiel das Anpassen der Bildhelligkeit.

- Installation in Python: pip install –upgrade opencv-python

3.3 Abwägung von neuronalen Netzen

In der folgenden Abwägung werden einige verfügbare Netze aufgeführt, die für Tensorflow zur Verfügung gestellt werden. Dabei werden die aufgeführten Modelle in Genauigkeit und Geschwindigkeit verglichen.

3.3.1 Auswahl des Netzes

Hinsichtlich trainierter Netze bietet „Tensorflow“ einige Möglichkeiten an. Hierbei handelt es sich um die „Tensorflow detection model zoo“ [LLC18d], welche mit verschiedensten Datensätzen ausgiebig trainiert wurden. Bei dem COCO Datenset handelt es sich um „Common Objects In Context“ [Con18] und beinhaltet rund 330 Tausend Bilder, die über 1.5 Millionen Schichten in 90 Klassen trainiert wurden. Da es sich um häufig auftretende Objekte in einer realen Umgebung handelt, sind die Modelle gut für das Vorhaben geeignet. In der folgenden Tabelle wurden ein paar dieser Modelle ausgesucht und verglichen.

Tabelle 1: Trainierte Modelle auf dem COCO Dataset [LLC18d]

Modell Name	Geschwindigkeit	mAP	Ausgabe
faster_rcnn_inception_v2_coco	58 ms	28	Boxen
faster_rcnn_resnet50_coco	89 ms	30	Boxen
rfcn_resnet101_coco	92 ms	30	Boxen
faster_rcnn_resnet101_coco	106 ms	32	Boxen
mask_rcnn_inception_resnet_v2_atrous_coco	771 ms	36	Maske
faster_rcnn_nas	1833 ms	43	Boxen

In der Tabelle werden sechs mögliche Modelle aufgeführt, in der diese anhand der durchschnittlichen Geschwindigkeit, der Genauigkeit und eines Ausgabetyps verglichen werden. Die zweite Spalte mAP (mean average Persition) gibt die mittlere durchschnittliche Genauigkeit an, mit welcher es Objekte erkennt. Berechnet wird diese durch die Genauigkeit der richtigen Ergebnisse im Vergleich zu allen ausgeführten Trainingsdurchläufen des Systems. Die mAP stellt also keine prozentuale Wahrscheinlichkeit dar. In der letzten Spalte wird die Ausgabeform beschrieben. Hier kann zwischen Boxen und Masken unterschieden werden. Je nach Ausgabetyp wird dabei entweder ein viereckiger Kasten (Abbildung 3b) um das Objekt erstellt, oder es wird eine Maske (Abbildung 3a) über dieses gezogen. In der Auswahl befinden sich ein Model, welches eine Maskierung ausgibt. Dieses hat aber zusätzlich eine langsame Laufzeit.

(a) Maskierung [LLC18a] (b) Markierung durch Kästen [LLC18b]

Abbildung 3: Ausgaben von neuronalen Netzen

Bei dem Vergleich der Modelle fällt auf, dass alle eine ähnliche Genauigkeit aufweisen und sich nur leicht unterscheiden. In der Geschwindigkeit jedoch werden die Unterschiede erkennbar größer. Durch die hohe Geschwindigkeit und vergleichsweise hohe Genauigkeit, wird vorerst das faster_rcnn_inception_v2_coco Modell getestet und ausgewertet.

Das ausgewählte Modell wurde mit sechs Klassen umtrainiert. Die Klassen bestanden aus einem Skart-Kartenset von neun bis Ass [Edj18]. Rund fünf Stunden wurde das Netz trainiert und hat dabei 60 Tausend Testdurchläufe absolviert. Eine Überprüfung des Netzes hat eine Genauigkeit von 97 bis 99 Prozent ergeben. Auch Tests ergaben das die Erkennung fehlerfrei funktioniert. Das Testbild (Abbildung 4) zeigt, dass alle Karten mit einer hohen Wahrscheinlichkeit identifiziert werden konnten.

Abbildung 4: Analysiertes Bild durch das Testnetz

Das erfolgreiche Training und Testen des Netzes hat die Verwendung des ausgewählten Modells bestärkt. Die weitere Vorgehensweise besteht aus einer Definition und Festlegung der Trainingsdaten für das benötigte neuronale Netz.

3.4 Trainingsdaten generieren

Zu Beginn hat ein untrainiertes neuronales Netz in seinen Neuronen zufällige Gewichtungen und Schwellwerte, da es bis zu diesem Zeitpunkt noch keine Informationen darüber hat, welche Objekte es erkennen soll und welche Eigenschaften diese haben. Damit das zu entwickelnde neuronale Netz die Informationen erhält, die für eine Klassifizierung notwendig sind, müssen zunächst Trainings- und Testdaten generiert werden. Für das Training werden optimalerweise mehrere Tausend Trainings- und Testbilder benötigt, damit eine möglichst hohe Genauigkeit erreicht werden kann. Die Trainingsphase dauert in der Ausführung einige Stunden.

Wie bereits in einem vorherigen Kapitel beschrieben, wird wegen der begrenzten Zeitressource ein vortrainiertes Netz verwendet, welches mit den COCO Testdaten trainiert wurde. Der Vorteil eines vortrainierten Netzes besteht darin, dass die Gewichtungen der frühen Schichten gut eingestellt sind und diese nicht verändert werden müssen. Lediglich die letzten Schichten sollen mit den eigenen Daten umtrainiert werden. Durch die guten Einstellungen der ersten Schichten werden weniger Trainings- und Testdaten benötigt, um eine hohe Genauigkeit des neuronalen Netzes zu erreichen.

3.4.1 Funktionsweise

Bei einem untrainierten Netz werden jedem Neuron zunächst ein zufälliges Anfangsgewicht sowie ein Schwellwert zugewiesen. Beim Starten der Trainingsphase werden Daten in das Netz eingeführt. Jedes Neuron verarbeitet nun alle Eingangssignale mit der zugewiesenen Gewichtung und gibt die Ergebnisse an das nächste Neuron weiter.

Im Output-Layer wird nun das Ergebnis der Berechnungen ermittelt. Das Ergebnis des Netzes wird mit der tatsächlichen Bezeichnung des Objektes verglichen. Lag das Netz bei der Bestimmung falsch wird zunächst der entstandene Fehler berechnet und die Neuronen werden dementsprechend angepasst. Dabei werden die Gewichtungen der Neuronen umso stärker verändert, je mehr Einfluss diese auf das resultierende Ergebnis hatten. Dieser Vorgang wird nun viele Male wiederholt.

3.4.2 Auswahl der Klassen

Bei der Auswahl der Klassen wurde darauf geachtet die Genauigkeit des Netzes zu erkennen, indem Klassen ausgewählt werden, welche Ähnlichkeiten aufweisen, jedoch unterschiedliche Bezeichnungen haben. Beispiele sind hier Flaschen, Tetrapacks oder auch Schachteln.

- Klasse: 1 Name: „Milch, Packung“
- Klasse: 2 Name: „Orangensaft, Packung“
- Klasse: 3 Name: „Wasser, Flasche“
- Klasse: 4 Name: „Bier, Flasche“
- Klasse: 5 Name: „Brunch, Aufstrich“
- Klasse: 6 Name: „Margarine, Aufstrich“

Damit das zu entwickelnde Netz für die ausgewählten Klassen verwendet werden kann, muss ein großer Datensatz generiert werden. Um Aussagen über die Genauigkeit des Netzes machen zu können, wurden Objekte mit hoher Ähnlichkeit verwendet. Als Testobjekte wurden zwei Tetrapacks, zwei Flaschen und zwei Aufstriche verwendet. Die Einteilung der Klassen ist in der folgenden Tabelle aufgeführt.

Tabelle 2: Ausgewählte Klassen und Struktur

Klassen Nr.	Klassenbezeichnung	Bilder
1	Milch - Packung	1-99
2	Orangensaft - Packung	100-199
3	Wasser - Flasche	200-299
4	Bier - Flasche	300-399
5	Brunch - Aufstrich	400-499
6	Margarine	500-599
7	Gemischt	600-699

Zunächst müssen Testdaten in Form von Bildern generiert werden, wozu Bilder aus dem Internet sowie selbst erstellte Fotografien der Objekte verwendet werden, welche aus verschieden Winkeln und unterschiedlichen Lichtverhältnissen aufgenommen wurden. Dabei sollte ein Bild die Größe von 200 KB nicht übersteigen, damit das Training erfolgreich verläuft und nicht unnötig viel Zeit benötigt. Die selbst erstellten Bilder wurden deshalb um den Faktor 0.2 verkleinert. Verwendet wurde für die Komprimierung das in Python geschriebene Programm reziser.py.

Zum Erstellen eines Ground True müssen die Bilder gelabelt werden. Hierfür wurde das Programm labelImg.py verwendet. Beim Labeln werden die zu klassifiezierenden Objekte auf dem Bild mit den Positionsdaten und der Bezeichnung markiert und in einer XML Datei abgespeichert. Nach dem Markieren werden die Daten in Trainings- und Testdaten aufgeteilt. 80 Prozent kommen in den Trainingsordner und 20 Prozent in den Testordner. Für die Durchführung des Trainings werden die Informationen der einzelnen XML-Dateien in zwei CSV-Dateien zusammengefasst. Diese Dateien beinhalten Tabellen in denen alle notwendigen Informationen zu den Bildern aufgeführt werden. Verwendet wird dafür das Programm xml_to_csv.py.

3.5 Trainieren

Nachdem alle Bilddaten gelabelt wurden und die CSV-Dateien für die Trainings- und Testdaten erstellt wurden, kann das Training beginnen. Zusätzlich zu den Traningsdaten wird eine Labelmap (labelmap.pbtxt) benötigt, in welcher die aus Tabelle 3.4.2 ausgewählten Klassen aufgeführt werden. Außerdem wird die Konfigurationsdatei des verwendeten Netzes (faster_rcnn_inception_v2_coco) benötigt. Das Training wird mit Ausführung des training.py-Programms gestartet. Ab diesem Zeitpunkt, benötigt das Training viel Zeit.

Zusätzliche Parameter:
–logtostderr
Dieser Parameter gibt an, wie Fehler protokolliert werden.
–train_dir=training/
Dieser Parameter gibt den Pfad zum Ordner an, in welchem die Trainingsergebnisse gespeichert werden sollen.
–pipeline_config_path=training/faster_rcnn_inception_v2_coco.config
Dieser Parameter gibt den Pfad zu der Konfigurationsdatei an.

Damit die Ergebnisse so gut wie möglich ausfallen, wird das Netz ca. 10 Stunden mit der in Kapitel 3.1.1 beschriebenen Konfiguration trainiert. In dieser Zeit wurden 200.000 Trainingsdurchläufe durchgeführt. Kurz vor dem Abschluss der Trainingsphase wird ein durchschnittlicher „loss“ von 0,03 erreicht. Der „loss“ beschreibt den Fehler, welchen das Netz bei der Bestimmung eines Datensatzes gemacht hat und versucht diesen, durch eine Anpassung der Gewichtungen betroffener Neuronen, zu minimieren. Alle fünf Minuten werden die Zwischenschritte gespeichert. Nach Abschluss des Trainings kann, mit Hilfe des export_inference_graph.py-Programms, ein sogenannter „frozen inference graph“ generiert werden, welcher unser künstliches neuronales Netz darstellt.

Zusätzliche Parameter:
–input_type image_tensor
Dieser Parameter bestimmt den Eingabetyp des neuronalen Netzes.
–pipeline_config_path legacy/training/faster_rcnn_inception_v2_coco.config
Dieser Parameter gibt den Pfad zur Konfigurationsdatei an.
–trained_checkpoint_prefix legacy/training/model.ckpt-XXXX
Dieser Parameter gibt den Pfad zum letzten Speicherstand des Trainings an.
–output_directory inference_graph
Dieser Parameter bestimmt den Ort, an welchen der Graph exportiert wird.

Nach Erstellung des „frozen inference graph“, welcher die Gewichtungen und Objektklassen enthält, ist das Netz an sich fertig und bereit für den praktischen Test. Dafür wurde aus der object_detection_tutorial.ipynb-Datei von der Tensorflow API ein ausführbares Programm für Testzwecke geschrieben. Im nächsten Abschnitt wird der Testverlauf beschrieben und ausgewertet.

4 Testergebnisse

In diesem Kapitel sollen die Zuverlässigkeit und die Geschwindigkeit des neuronalen Netzes geprüft und ausgewertet werden. Dazu wird das Netz mit nicht markierten und dem Netz unbekannten Bildern getestet. Es soll sichergestellt werden, dass das Netz die Objekte nicht bereits durch das Training kennt. Anschließend wird überprüft, wie robust das Netz gegenüber falscher Klassifizierungen ist und ob alle Objekte erkannt werden können. Tensorflow bietet mit dem Tool Tensorboard die Möglichkeit den Fortschritt des Trainings grafisch darzustellen. In der folgenden Grafik wird der auftretende „Loss „ins Verhältnis zu der Trainingszeit gestellt.

- tensorboard –logdir=legacy/training –host=127.0.0.1

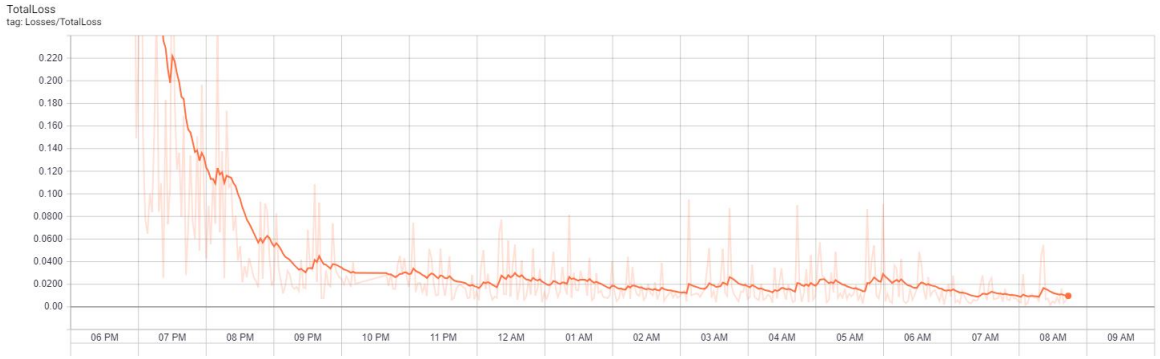

Abbildung 5: Loss-Graph des umtrainierten inception_v2

Der „loss“ stellt dabei keine Prozentangabe dar, sondern ist eine Zusammenfassung der Fehler, die für jedes Beispiel in Trainings- oder Validierungssätzen gemacht wurden. Je kleiner der „loss“ dabei ist, desto höher ist die Wahrscheinlichkeit, dass die Klassifizierung richtig ist. Um die Grafik zu veranschaulichen, wurden kurzfristige Schwankungen geglättet. Innerhalb der ersten vier Stunden des Trainings konnte der größte Fortschritt beobachtet werden. In dieser Zeit, ist der „loss“ von anfänglich 2.0 auf 0.05 gesunken. Dabei wurden rund 60.000 Trainingseinheiten durchlaufen.
Bei den ersten Testergebnissen, welche in Abbildung 6 zu sehen sind, konnten alle Objekte erkannt und einer Klasse zugewiesen werden. Jedoch wurde die „Orangensaft - Packung“, mit einer hohen Wahrscheinlichkeit, als „Milch - Packung“ falsch klassifiziert, was auf die hohe Ähnlichkeit der Objekte zurückzuführen ist.

Aufgrund der falschen Klassifizierung wurde ein weiteres vortrainiertes Netz mit dem generierten Datensatz trainiert und getestet. Das dafür ausgewählte Netz ist das faster_rcnn_resnet101_coco. Dieses Models wurden aus der Tabelle 3.4.2 ausgewählt, da die mittlere durchschnittliche Genauigkeit vergleichsweise hoch ist und es die unterschiedlichen Objekte besser unterscheiden kann.

Abbildung 6: Erster Test des neuronalen Netzes

Das zweite Modell, welches mit dem Datenset trainiert wird, ist das faster_rcnn_resnet101_coco mit einer mAP von 32, welches aber wesentlich langsamer ist. Für einen Trainingsdurchlauf benötigt das Modell, mit rund 600 ms im Durchschnitt, mehr als doppelt so lang, wie das faster_rcnn_inception_v2_coco. Es benötigt dafür aber nur die Hälfte an Durchläufen, um einen ähnlich niedrigen „loss“ zu erreichen.

Die Testergebnisse der einzelnen Klassen zeigen, dass die erreichte Genauigkeit besser ist, da allen erkannten Objekten die richtigen Klassen zugewiesen wurden. Dieses Netz zeigte sich aber deutlich Störungsanfälliger, so dass je nach Winkel nicht immer alle Objekte erkannt wurden. Um die Stabilität zu erhöhen, wurden die Trainingsdaten, in Form von Bildern, erweitert und das resnet101, welches die höhere „mAP“ hat, auf den erweiterten Datensatz trainiert.

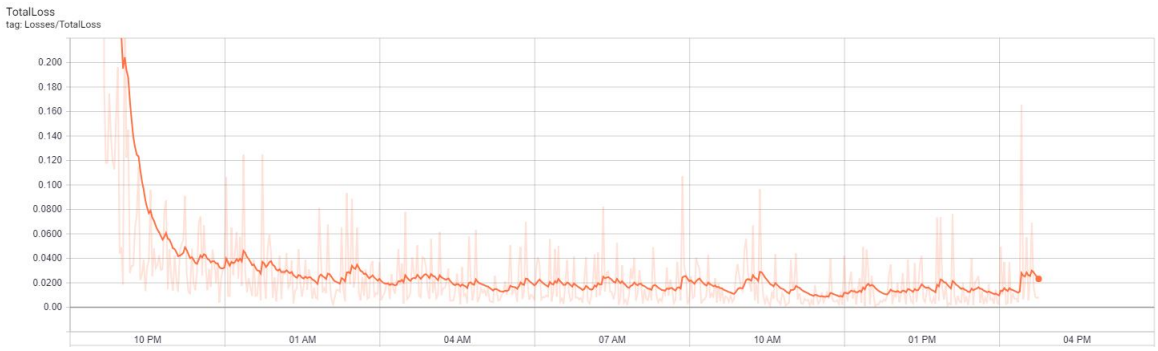

Abbildung 7: Loss-Graph des umtrainierten resnet101

Der der Graph in Abbildung 7 zeigt, das im Vergleich zum Inception_v2 mehr zeit benötigt wird um vergleichbare Ergebnisse zu erzielen. Bereits nach ca. 7 Stunden des Trainings, erreicht das resnet101 einen vergleichbaren „loss“ wie der Vorgänger. Weitere Tests mit Bildern, soll die Zuverlässigkeit des neuronalen Netzes Überprüfungen und mögli-

che schwächen ausfindig machen. Tests mit dem verbesserten Netz haben ergeben, dass die Erkennung der Orangensaftpackung stabiler funktioniert, solange das Bild und die Oberflächen der Objekte nicht zu dunkel werden. Die Abbildung 8 zeigt alle trainierten Objekte, mit denen das Modell faster_rcnn_resnet101_coco trainiert wurde. Hier kann man erkennen, dass alle Klassen richtig zugeordnet werden konnten.

Abbildung 8: Objekterkennung des optimierten resnet101

Im nächsten Kapitel sollen die Ergebnisse des Projektes mit den Anforderungen verglichen und ausgewertet werden.

5 Fazit

Das generieren, beziehungsweise trainieren eines neuronalen Netzes hat gut funktioniert. Leichte Schwächen konnten bei der richtigen Identifizierung ähnlicher Objekte und der Störungsanfälligkeit gegenüber Lichtverhältnissen festgestellt werden. Für das Fazit werden in diesem Kapitel die Testergebnisse mit den Anforderungen, welche zu Beginn des Projektes festgelegt wurden, verglichen.

5.1 Bewertung durch Anforderungen

Die ersten zwei funktionalen Anforderungen, bei denen es um die Möglichkeit geht Bilder zu verarbeiten und in diesen Bildern trainierte Klassen zu erkennen, konnten erfüllt werden. Lediglich die Genauigkeit konnte zunächst nicht zu voller Zufriedenheit erfüllt werden. Die meisten Objekte können bei schlechteren Bild- und Lichtverhältnissen erkannt werden, nur bei zu ähnlichen Objekten kann die Erkennung fehlerhaft sein, wie man bei der Milch und dem Orangensaft beobachten konnte. Mit einer Erhöhung der Trainingsdaten und einer Lichtkorrektur könnte dies allerdings verbessert werden.

Die organisationalen Anforderungen konnten auch zum Großteil erfüllt werden. Klassen können unkompliziert erweitert werden, indem Testdaten der gewünschten Klasse generiert und gelabelt werden. Das Trainieren und Ausführen des neuronalen Netzes ist zwar im Moment nicht mit einer Datei ausführbar, aber die notwendigen Schritte sind nicht aufwendig. Die Installation aller notwendigen Module und Bibliotheken kann durch die Eingabeaufforderung oder der „shell“ ausgeführt werden. Lediglich für Anaconda muss man eine manuelle Installation durchführen. Das Netz kann auf jedem Endgerät, mit der benötigten Entwicklungsumgebung, betrieben werden. Für mobile Endgeräte ist das Netz jedoch, wegen der benötigten Rechenleistung, nicht geeignet.

Bei den qualitativen Anforderungen muss ein Gleichgewicht zwischen Genauigkeit und Geschwindigkeit gefunden werden. Denn umso höher die Genauigkeit des Netzes sein soll, desto länger benötigt es um ein Bild auszuwerten. Umgekehrt sinkt die Genauigkeit, umso schneller das Netz arbeitet. Es wurde also darauf geachtet, dass das Netz in beiden Punkten solide Ergebnisse erzielt. Der Schwerpunkt lag aber auf der Genauigkeit des Netzes. Leicht verdeckte Objekte können auch bis zu einem gewissen Punkt erkannt und richtig zugeordnet werden. Die Genauigkeit kann beispielsweise durch mehrere Trainingsdaten erhöht werden.

5.2 Bewertung der Netze

Das neuronale Netz funktioniert größtenteils wie gewollt. Um die richtige Identifikation von ähnlichen Objekten zu steigern, sollte die Anzahl an Trainings- und Testdaten erhöht werden. Außerdem könnte eine Angleichung der Lichtverhältnisse per Bildverarbeitung dafür sorgen, dass die Genauigkeit des Netzes steigt. Beim testen der Modelle wurde festgestellt, dass die Übereinstimmung von erkannten Objekten der Testbilder, teilweise fehlerhaft waren. Da eine etwas längere Verarbeitungszeit kein allzu großes Problem darstellt, wurde auf einem Modell mit einer höheren mAP und Verarbeitungszeit trainiert.

Auch das Testen des neuronalen Netzes mit leicht verdeckten Objekten ist zuverlässig. In Abbildung 9a kann festgestellt werden, dass es mit Verdeckungen zurecht kommt, jedoch nur bei guter Ausleuchtung. Bei schlechten Lichtverhältnissen arbeitet das Netz nicht immer zuverlässig (Abbildung 9b)

(a) Testbild bei guter Belichtung

(b) Testbild bei schlechter Belichtung

Abbildung 9: Testbilder mit leichten Überlappungen in verschiedenen Lichtverhältnissen

6 Aussicht

Die Entwicklung des neuronalen Netzwerkes hat gezeigt, dass es möglich ist den Inhalt von Regal-Systemen mit Hilfe von Objekterkennung zu identifizieren. Dabei ist aber auch aufgefallen, dass mehr Trainingsdaten benötigt werden, je ähnlicher die Klassen sind. Das bedeutet, dass für das geplante System eine große Anzahl an Trainingsdaten generiert werden müssen. Grundsätzlich ist es aber möglich mit den Ergebnissen weiter arbeiten zu können.

Für die Realisierung des geplanten Systems sind weitere Optimierungen des neuronalen Netzes erforderlich. Zum einen soll die Genauigkeit der Identifizierung bei ähnlichen Objekten erhöht werden. Zum anderen sollen Optimierungen im Bereich der digitalen Bildverarbeitung vorgenommen werden, um die Identifikation robuster gegen Störeinflüsse, wie Lichtverhältnisse oder Bildrauschen, zu machen.

Literatur

[Ang18] Bistra Angelova. *Optical character recognition für chinesische Schrift*. 2018. URL: https://bit.ly/2VfsQOL.

[Con18] Common Objects in Context. *COCO Datenset*. 2018. URL: https://bit.ly/2DltzbB.

[Edj18] Evan EdjeElectronics. *Testbilder*. 2018. URL: https://bit.ly/2RjFVHz.

[Ert13] Wolfgang Ertel. „Grundkurs Künstliche Intelligenz“. In: *Eine praxisorientierte Einführung* 3 (2013).

[Goo+16] Ian Goodfellow u. a. *Deep learning*. Bd. 1. MIT press Cambridge, 2016.

[Hea18] Anaconda Corporate Headquarters. *Anaconda Python*. 2018. URL: https://bit.ly/2EtAZdT.

[LeC+89] Yann LeCun u. a. „Backpropagation applied to handwritten zip code recognition“. In: *Neural computation* 1.4 (1989), S. 541–551.

[LLC18a] Google LLC. *Bild 3a Maskierung*. 2018. URL: https://bit.ly/2BMVrBU.

[LLC18b] Google LLC. *Bild 3b Umschlossen*. 2018. URL: https://bit.ly/2QRkQF0.

[LLC18c] Google LLC. *Tensorflow*. 2018. URL: https://bit.ly/1MWEhkH.

[LLC18d] Google LLC. *Tensorflow detection model zoo*. 2018. URL: https://bit.ly/2EtFjJP.

[Pen+16] Min Peng u. a. „NIRFaceNet: A Convolutional Neural Network for Near-Infrared Face Identification“. In: *Information* 7.4 (2016). ISSN: 2078-2489. DOI: 10.3390/info7040061. URL: https://bit.ly/2GGNqUP.

[SOP13] Die SOPHISTen. „Schablonen für alle Fälle“. In: *SOPHIST GmbH* (2013).