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, das 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 neuronales 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, welches Informationen verarbeitet und weiterleitet. Das neuronale Netz ist unter anderem für das menschliche Lernen zuständig.

2.1 Künstliches neuronales Netz

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, eher soll die Funktionsweise von künstlichen neuronalen Netzen erläutert werden [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.

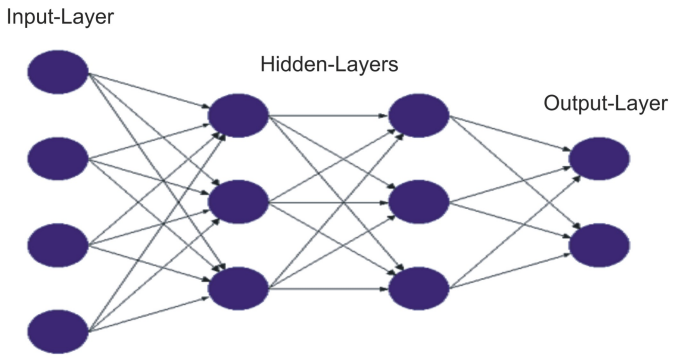

Abbildung 1: Schematischer Aufbau eines künstlichen neuronalen Netzes [Ang18]

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. Jeder 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 wurden. 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 Faltendes neuronales Netz

Anders als bei einem KNN, welches auf Grundlage des menschlichen Gehirns entworfen wurde, orientiert sich ein „Convolutional Neural Network“(CNN) an dem Konzept des menschlichen Sehens [LeC+89]. Dabei 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.

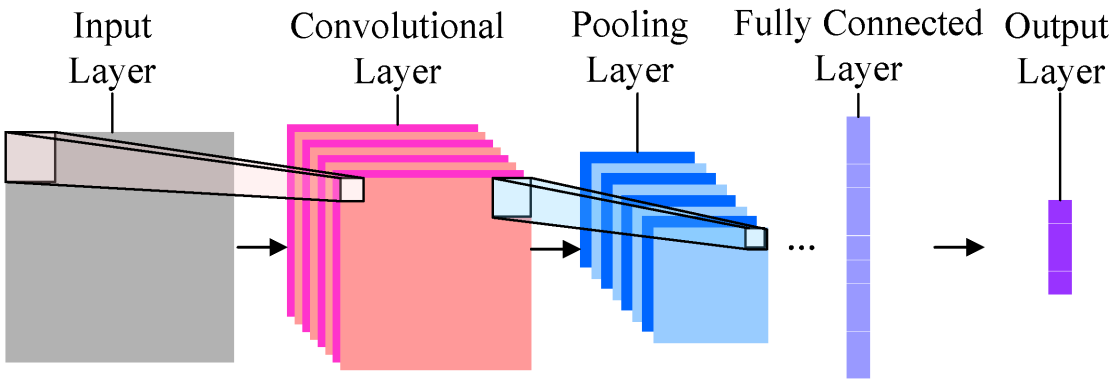

Abbildung 2: Funktionsweise eines Convolutional Neural Network[Pen+16]

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 gemachte Entscheidungen 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 diese 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 sowie eine Entwicklungsumgebung benötigt. In den kommenden Unterpunkten wird die Entwicklungsumgebung sowie alle benötigten Bibliotheken aufgeführt.

3.1.1 Verwendete Hardware Konfiguration

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 Acht fache Zeit benötigt. Der Vorteil liegt hier eindeutig bei der Grafikkarte, durch die Optimierung in Matrixmultiplikationen und wird aus diesem Grund für das Training verwendet.

3.1.2 Anaconda python

Die im späteren aufgeführten CNN's sowie die Tensorflow API wurden in der Programmiersprache Python erstellt, weswegen wir zum um-trainieren mit Python arbeiten. Des weiteren bietet Python eine Vielzahl an verfügbaren Bibliotheken welche wir benötigen. Anaconda [Hea18] ist eine „Open-Source-Distribution"für die Programmiersprache Python. Mit dieser können wir gleichzeitig mehrere Versionen von Python generieren und die notwendigen Bibliotheken installieren. Anaconda bietet den Vorteil einfach zwischen den verschieden Python Versionen wechseln zu können
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, wird 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 Plattform unabhängige 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 können. Tensorflow kann auf allen CPU's betrieben werden, oder mit Nvidia Grafikkarten, welche CUDA unterstützen.

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

Verwendet wird zusätzlich die API von Tensorflow (Models) für die Objekterkennung. 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 benötigt man einige spezielle Python Bibliotheken. Diese 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 um 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 dafür, um die Markierungen von Objektsklassen und deren Positionen auszulesen. Durch die Arbeit mit Elementbäumen

werden nur die benötigten Informationen herausgefiltert, woraus einem bessern Laufzeitverhalten resultiert.

- Installation in Python: pip install –upgrade lxml

Cython

Python ist im vergleich zu C eine recht langsame Programmiersprache, was die Ausführungzeit des Programmcodes wesentlich erhöht. Cython ist eine Programmbibliothek und auch Programmiersprache, welche die wesentlich langsame 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, das 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 Web Applikation die es ermöglicht Dokumente welche Live Code, Abbildungen oder Text beinhalten zu teilen. Dabei können Inhalte von Dateien im Browser aufgerufen und ausfeführt 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 Formlen 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 enfache selektierung und Inditzierung von Daten. In diesem Projekt wird Pandas für die Erhebung von Informationen aus den Trainingsdaten verwendet und für die Generierung von Graphen.

- 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 das anpassen der Bild Helligkeit.

- 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

In Hinsicht auf Trainierte 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, welche über 1.5 Millionen schichten in 90 Klassen trainiert wurden. Da es sich um häufige auftretende Objekte in einer realen Umgebung handelt, sind die Modelle gut für unser vorhaben geeignet. In der folgenden Tabelle, wurden ein paar dieser Modelle ausgesucht und verglichen.

In der Tabelle, werden 6 mögliche Modelle aufgeführt wobei diese anhand der durchschnittlichen Geschwindigkeit, der Genauigkeit und eines Ausgabetyps verglichen werden. Die zweite Spalte, COCO 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 auf alle ausgeführten Trainings Durchläufe des Systems. Es stellt als keine Prozentuale Wahrscheinlichkeit dar. In der letzten Spalte wird die Ausgabeform beschrieben. Hierbei kann zwischen Boxen und Masken unterschieden werden. Dabei wird je nach Ausgabetyp entweder ein viereckiger Kasten (Abbildung **??**) um das object erstellt, oder es wird eine Maske (Abbildung **??**) über dieses gezogen. In der Auswahl befinden sich jedoch keine Modelle welche eine Maskierung ausgeben.

Modell Name	Geschwindigkeit	COCO 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
ssd_mobilenet_v1_fpn_coco	53 ms	32	Boxen
ssd_resnet_50_fpn_coco	76 ms	35	Boxen

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

(a) Makierung durch Maskierung [LLC18a] (b) Makierung 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 6 Klassen um-trainiert. 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 ergeben das die Erkennung fehlerfrei funktioniert. Das Testbild (Abbildung 4 zeigt, das alle Karten mit einer hohen Wahrscheinlichkeit identifiziert werden konnten.

Das erfolgreiche Training und testen des Netzes hat die Verwendung des Ausgewählten Modells bestärkt. Die Vorgehensweise sieht in den weiteren Schritten eine Definition und Festlegung der Trainings-Daten für das benötigte neuronale Netz.

Abbildung 4: Analysiertes Bild, durch das Testnetz

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, für eine Klassifizierung, müssen zunächst Trainings und Testdaten generiert werden. Für das Training werden optimaler weise mehrere Tausend Trainings- und Testbilder benötigt, damit eine möglichst hohe Genauigkeit erreicht werden kann. Auch die Trainingsphase dauert in der Ausführung einige Stunden.

Wie in der Auswahl des Neuronalen Netz bereits beschrieben wird wegen der begrenzten Zeitressource ein trainiertes Netz von COCO verwendet. der Vorteil bei der Verwendung eines bereits trainierten Netzes besteht darin, das 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

Wie gerade schon angeschnitten, bekommen bei Netzen welche neu trainiert werden jedes Neuron zunächst ein zufälliges Anfangs-Gewicht und Schwellwert. Beim starten der Trainingsphase werden Daten in das Netz eingeführt. Jedes Neuron verarbeiten nun al-

le Eingangssignale mit der zugewiesenen Gewichtung und geben 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 so stark verändert je mehr Einfluss diese auf das resultierende Ergebnis hatten. Dieser Vorgang wird nun viele male wiederholt. Das ist die kurze Fassung, wie ein Künstliches Neuronales Netz lernt.

3.4.2 Auswahl der Klassen

Bei der Auswahl der Klassen wurde darauf geachtet, die Genauigkeit des Netzes zu erkennen, indem Klassen ausgewählt werden, welch eine gewisse Ähnlichkeit aufweisen jedoch unterschiedliche Bezeichnungen haben. Beispiele sind hier Flaschen, Tetra Packs oder auch Schachteln.

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

3.5 Trainieren

Damit das zu entwickelnde Netz für die ausgewählten Klassen verwendet werden kann, muss ein großer Datensatz generiert werden. Wie im letzten Abschnitt beschrieben, werden Klassen ausgewählt, welche eine hohe Ähnlichkeit aufweisen. Dadurch soll überprüft werden, ob das neuronale Netz auch Objekte mit nur detaillierten Unterschieden auseinanderhalten kann.
Als Testobjekte wurden zwei Tetra Packs, zwei Flaschen und zwei Aufstriche verwendet. Die Einteilung der Klassen wird in der folgenden Tabelle gezeigt.

Tabelle 2: Ausgewählte Klassen und Struktur

Klasse	Bezeichnung	Bereich
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, dafür wurden zum einen, Bilder von Online-Suchmaschinen, je nach Verwendbarkeit für das Netz heruntergeladen. Dabei sollte das Bild eine Größe von 200 KB nicht übersteigen, damit das Training erfolgreich verläuft und nicht zusätzlich mehr Zeit benötigt. Des weiteren wurden Objekte der aufgeführten Klassen (Tabelle2), in verschiedensten Winkeln und Lichtverhältnissen Fotografiert. Die aufgenommenen Bilder übersteigen mit 3-5 MB die maximale Größe der Testdaten. Aus diesem Grund, wurden die Bilder mit einem Programm um den Faktor 0.2 verkleinert. Verwendet wird für die Komprimierung ein in Python geschriebenes Programm (reziser.py).

Damit die API von Tensorflow weiß, welche Klassen auf dem Bild zu sehen sind und wo sich diese im Bild befinden, 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 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.

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 Kaptitel 3.4.2 ausge-

wählten Klassen aufgeführt werden. Außerdem brauchen wir die Konfigurationsdatei des verwendeten Netzes (faster_rcnn_inception_v2_coco). Das Training wird mit Ausführung des training.py Programms gestartet. Ab diesem Zeitpunkt, benötigt das Training viel Zeit.

Zusätzlich Parameter:
–logtostderr
–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 benötigten config Datei an

Damit die Ergebnisse so gut wie möglich ausfallen, wird das Netz ca. 10 Stunden, mit der in Kapitel? beschriebenen Konfiguration Trainiert. Dabei schafft das Programm 200.000 Trainings- und Testdurchläufe und ist kurz vor Abschluss der Trainingsphase bei einem durchschnittlichen „loss“von 0,03. Alle 5 Minuten werden die Zwischenstände gespeichert. Aus dem letzten Standpunkt des Trainings wird mit hilfe des export_inference_graph.py ein Sogenannter „frozen inference graph“generiert, welcher unser Künstliches neuronales Netz darstellt.

Mitgegebenen Parameter:
–input_type image_tensor
–pipeline_config_path legacy/training/faster_rcnn_inception_v2_coco.config
Gibt den Pfad zur config Datei an.
–trained_checkpoint_prefix legacy/training/model.ckpt-XXXX
Gibt den Pfad zur letzten Speicherstand des Trainings an.
–output_directory inference_graph
Bestimmt den Ort an welchen der Graph exportiert wird.

Nach Erstellung der „Frozen Inference Graph“, welcher die Gewichtingen 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 die Testzwecke geschrieben. Im weiteren wird der Testverlauf beschrieben und Ausgewertet.

4 Testergebnisse

In diesem Kapitel soll die Zuverlässigkeit sowie Geschwindigkeit, des neuronalen Netzes geprüft und ausgewertet werden. Dazu wird das Netz mit nicht markierten Bildern getestet, wobei nur Testbilder verwendet werden, welche nicht im Trainings- und Testdatensatz vorhanden waren. Es soll sichergestellt werden, dass das Netz die Objekte nicht durch das Training kennt. Anschließend wird überprüft wie zuverlässig das Netz ist und ob es alle Objekte erkannt hat.
Tensorflow bietet mit dem Tool Tensorboard die Möglichkeit den fortschritt des Trainings grafisch darzustellen. In der folgenden Grafik wird der auftretende „Loss „in Verhältnis zu der Trainingszeit gestellt.

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

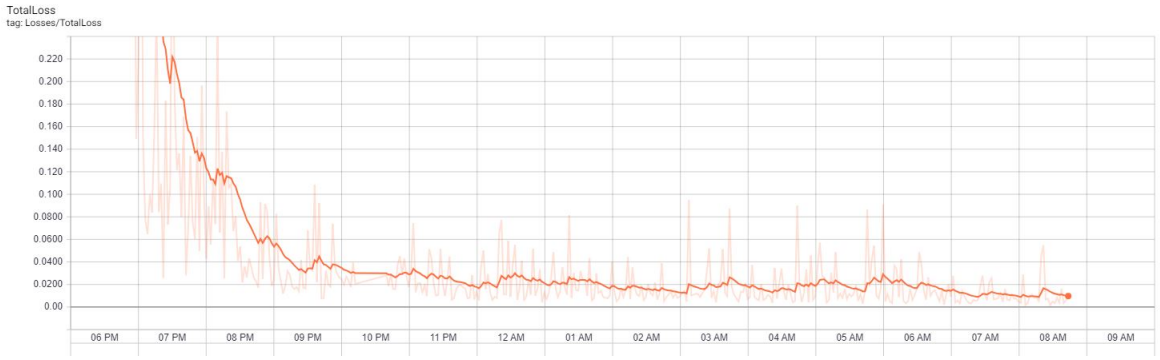

Abbildung 5: Loss Graph des umtrainierten inceptionV2

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. Um die Grafik zu veranschaulichen, wurden kurzfristige Schwankungen geglättet. Zu beginn wird festgestellt, das die ersten 4 Stunden des Trainings den größten fortschritt aufweisen. In dieser Zeit, ist der „Loss“von anfangs 2.0 auf 0.05 gesunken. Dabei wurden rund 60.000 Trainingseinheiten durchlaufen.
Die ersten Testergebnisse sehen gut aus. Alle beinhaltenden Objekte konnten erkannt und zugewiesen werden. Diese wurden auch mit einer hohen Wahrscheinlichkeit identifiziert. Lediglich bei der „Orangensaft - Packung“hat das Netz einen Fehler gemacht. Diese wurde als Milch erkannt und das auch mit einer recht hohen Wahrscheinlichkeit. Zurückzuführen wäre das durch die hohe Ähnlichkeit der Formen.

Aus diesem Grund, soll überprüft werden, ob beim aufsetzen des Neuronalen Netzes Fehler gemacht wurden. Außerdem soll ein weiteres Vor-trainiertes Netze mit dem Datensatz getestet werden, Genauigkeit untereinander zu vergleichen. Das dafür ausgewählte

Abbildung 6: Erster Test des Neuronalen Netzes

Netz ist das faster_rcnn_resnet101_coco. Dieses Modelle wurden aus der Tabelle? ausgewählt, da die mittlere durchschnittliche Genauigkeit.

Das zweite Modell welches trainiert wurde ist das faster_rcnn_resnet101_coco mit einer mAP von 32 dafür ist es aber auch wesentlich langsamer. Für einen Trainingsdurchlauf benötigt das Modell mit rund 600ms im durchschnitt, mehr als doppelt so lange wie das faster_rcnn_inception_v2_coco. Es benötigt dafür aber auch nur die Hälfte an Durchläufen, um eine ähnlich niedrigen 'Loss' zu erreichen.

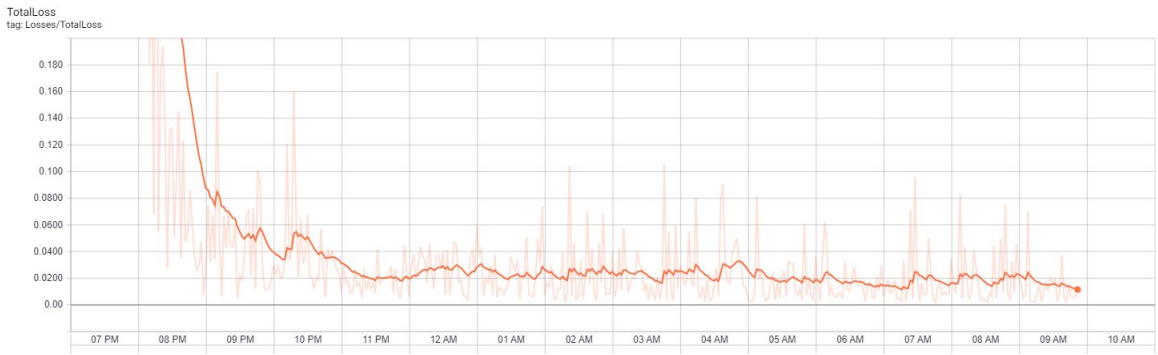

Abbildung 7: Loss Graph des umtrainierten resnet101

Testergebnisse der einzelnen Klassen zeigen auch das die Genauigkeit besser ist. Der Orangensaft welcher nicht identifiziert wurde, konnte jetzt erreicht werden. Dabei gab es wiederum das Problem das dieser je nach Winkel nicht immer richtig eingeordnet wurde. Um die Stabilität der ausgaben weiter zu erhöhen, sollen die Trainingsdaten weiter angehoben werden und das letzte Modell welches die höchste „mAP“hat soll auf den Datensatz trainiert werden.

Weitere Test mit dem Trainierten Netz haben ergeben, das die Erkennung der Oran-

gensaft Packung funktioniert, solange das Bild und die Oberflächen der Objekte nicht zu dunkel werden. Das Bild (Bild) zeigt alle Trainierten Objekte mit welchen das Modell faster_rcnn_resnet101_coco trainiert wurde. Hier kann man erkennen das alle Klassen richtig zugeordnet werden konnten.

Abbildung 8: Objekt erkennung des optimierten resnet101

Abgesehen von den leichten schwächen in der Erkennung ähnlicher Objekte hat das generieren eines Neuronalen Netzes Funktioniert. Schlechte Lichtverhältnisse haben die Ausgabe außerdem negativ beeinflusst. Im nächsten Kapitel sollen die Ergebnisse des Projektes, mit den Anforderungen verglichen und ausgewertet werden.

5 Fazit

Wie im vorherigen Kapitel beschrieben, hat die Entwicklung eines Künstlichen Neuronalen Netzes gut funktioniert. Lediglich bei ähnlichen Objekten, mit nur kleinen unterschieden, gab es Probleme mit der richtigen Identifizierung. Um ein finales Fazit geben zu können soll in diesem Kapitel auf die Anforderungen eingegangen werden, welche zu Beginn des Projektes festgelegt wurden.

5.1 Bewertung durch Anforderungen

Die ersten Zwei funktionalen Anforderungen 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, lediglich bei zu ähnlichen Objekten kann die Erkennung fehlerhaft sein, wie man bei der Milch und dem Orangensaft beobachten kann. Mit mehreren Trainingsdaten und einer Licht Korrektur 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 kann man nicht jede erfüllen, weil je höher die Genauigkeit des Netzes ist, desto länger benötigt es um ein Bild auszuwerten. Umgekehrt sinkt die Genauigkeit umso schnelle das Netz arbeitet. Es wurde also darauf geachtet das es bei 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, nur bei zwei Klassen kann nicht immer Korrekt unterschieden werden. Das liegt hauptsächlich an der hohen Ähnlichkeit der zwei Objekte. Die Genauigkeit könnte durch eine höhere Anzahl an Trainings und Testdaten verbessert werden. Außerdem könnte eine Angleichung der Lichtverhältnisse per Bildverarbeitung dafür sorgen, dass die Genauigkeit des Netzes steigt. Zwischen den getesteten Modellen wurde festgestellt, das die Übereinstimmung der Objekte einen wesentlichen unterschied in der Genauigkeit aufweisen, sodass eine etwas höhere Antwortzeit für den Verwendungszweck kein großes Problem darstellt.

Abschließend, kann das Projekt als Erfolgreich angesehen werden, mit kleinen Optimierungen.

6 Aussicht

Die Entwicklung eines neuronalen Netzwerkes hat gezeigt, das es Möglich ist den Inhalt Regal-Systeme mit Hilfe von Objekterkennung zu Identifizieren. Dabei ist aber auch aufgefallen, das mehr Trainingdaten benötigt werden je Ähnlicher sich die Klassen sind. Das bedeutet das 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 das Geplante System wären bis auf die Optimierung des Künstlichen neuronalen Netzes, die Systemschnittstellen welche zum einen das Bild vom Regal aufnimmt, verarbeitet und an das neuronale Netz weiter leitet und zum anderen müssen die Informationen der Bilderkennung an eine Datenbank übergeben werden, welche den Inhalt des Regals speichert. Darüber hinaus müsste eine Anwendung entwickelt werden, mit welcher der Benutzer den Inhalt des Regals managen kann.

Die Optimierung soll aber Für die Nächsten Schritte im Vordergrund stehen. Für die anstehende Bachelorarbeit, soll die Verbesserung des Neuronalen Netzes im Vordergrund stehen. dabei gibt es drei wichtige Punkte welche behandelt werden müssen. Zum einen soll die Genauigkeit der Identifizierung bei ähnlichen Objekten erhöht werden. Außerdem sollen Bildaufnahmen so verarbeitet werden, das Lichtverhältnisse gleich sind oder eine Veränderung dieser Keinen Einfluss auf die Ergebnisse haben. Zusätzlich soll die Auswirkung schlechter Bildverhältnisse ermittelt und Verbessert werden.

Literatur

[Ang18] Bistra Angelova. *Optical character recognition für chinesische Schrift*. 2018. URL: http://pille.iwr.uni-heidelberg.de/~ocr01/nnet.html.

[Con18] Common Objects in Context. *COCO Datenset*. 2018. URL: http://cocodataset.org/#home.

[Edj18] Evan EdjeElectronics. *Testbilder*. 2018. URL: https://github.com/EdjeElectronics/TensorFlow-Object-Detection-API-Tutorial-Train-Multiple-Objects-Windows-10/tree/master/images.

[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://www.anaconda.com/.

[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://vrodo.de/augmented-reality-facebook-gibt-software-fuer-objekterkennung-frei/.

[LLC18b] Google LLC. *Bild 3b Umschlossen*. 2018. URL: https://github.com/tensorflow/models/blob/master/research/object_detection/test_images/image1.jpg.

[LLC18c] Google LLC. *Tensorflow*. 2018. URL: https://www.tensorflow.org/.

[LLC18d] Google LLC. *Tensorflow detection model zoo*. 2018. URL: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md.

[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: http://www.mdpi.com/2078-2489/7/4/61.

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