HAW HAMBURG

Seminararbeit

Huy-Bao Le, Paul Duy An Nguyen

Bekleidungsauswertung auf einem Embedded System

Betreuung durch: Prof. Dr. Stephahn Pareigis
Eingereicht am: 05. Juli 2020

Fakultät Technik und Informatik
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abstract — Diese Seminararbeit befasst sich mit dem alltäglichen Problem der Entscheidung über die Bekleidung mit Bezug auf die aktuelle Wetterlage. Die Erfassung der Bekleidung nutzt eine Objekterkennung aus dem Deep-Learning Bereich. Die Auswertung über die Auswahl der Bekleidung wird dann über eine Heuristik getroffen. Die Erfassung und Auswertung werden auf einem Embedded System (Raspberry Pi 4) in Kombination mit einer Edge-TPU (Coral USB Accelerator) und einer Kamera (Raspberry Pi Camera Module v2) realisiert. Die Objekterkennung erreicht eine mAP von 0,56 und mAR von 0,75. Über die Edge-TPU konnte die Inferenzzeit von ca. 200ms auf 10ms durchschnittlich reduziert werden, welches eine Echzeit-Anwendung ermöglicht.

Keywords: Maschinelles Lernen, SSD, MobileNet V2, Eingebettete Systeme, Objekterkennung, Echtzeit-Anwendung

1 Einleitung$_{(L)}$

Vorweg: für die Bewertung dieser Arbeit wurde bei den Überschriften der Sektionen angegeben welcher von den Autoren diese bearbeitet haben. Sektionen von Huy-Bao Le geschrieben und bearbeitet sind mit einem (L) markiert, Sektionen von Paul Duy An Nguyen sind mit einem (N) versehen.

Die Objekterkennung ist ein kleiner Teil des Maschinellen Lernens (ML). Neben der Nutzung in verschieden Bereichen wie in der Automobilindustrie, Logistik und Luftfahrt, kann die Objekterkennung auch im Alltag hilfreich sein. Neben den Objektklassen, besteht die Objekterkennung aus der Informationsgewinnung aus Ort, Größe und eigentlichem Objekt in einem vorhandenen Bild. Das Ziel ist es alle Objekte bestimmter Klassen in einem Bild zu erkennen, dabei ist es egal wie groß diese sind, wo diese im Bild sind, die Pose, als auch teilweise abgeschnittene Objekte durch die Kameraperspektive oder äußerer Einflüsse.

Durch die stetige Verbesserung von Embedded Systems, mit immer kleiner werdenen Mikroprozessoren und gleichzeitig zunehmender Leistung, ist es möglich sehr energieeffiziente und leistungsfähige Geräte in der Größenordnung einer Kreditkarte zu bauen.

Einer dieser Geräte ist der Raspberry Pi 4 [11], ein Desktop Computer mit diversen Schnittstellen zur Verknüpfung von Geräten. In Kombination mit dem Raspberry Pi Kamera Modul, welches diverse Auflösungen von Bildern und Videos unterstüzt [10], und einem USB Beschleuniger von Google [12], bildet das die Grundlage an Hardware für die Umsetzung dieser Seminararbeit.

Beide Dinge vereint führt zu der Idee dieses Projekts: Bekleidungsauswertung auf einem Embedded System. Die Idee ist es auf einem portablem Gerät in Kombination von aktuellen Wetterdaten die momenten getragenen Klamotten zu idenzifieren und über eine eigene Heurisktik zu analysieren und zu bewerten. In Anbetracht des Ziels muss die Objekterkennung sowie die Evaluation der Kleidung in Echtzeit erfolgen. Die Schwierigkeit hierbei ist die limitierte Leitung des Raspberry Pis sowie die geeignete Wahl einer Architektur für die Objekterkennung. Als Programmiersprache für diese Seminararbeit wird Python verwendet, bei der diverse Frameworks für ML existieren.

Die Grundlage der Objekterkennung soll ein minimales, effizientes, schnelles und doch akkurates Ergebnis auf Embedded Systemen erzielen. Eine Kombination der SSD Architektur [19] und der MobileNet V2 Architektur [24] verbindet all diese Eigenschaften. Als Evaluierungsmetrik wird die COCO-Object Dectection-Metrik [7] verwendet, welche die Ergebnisse über die Mean Average Precision (mAP) und der Mean Average Recall (mAR) wiedergibt.

Um das Model geeinget zu trainieren wird ein Datensatz benötigt, welcher verschiedenste Bekleidungen beinhaltet. Dafür wurde der Datensatz von DeepFashion2 [26] verwendet. Dieser besitzt über 490000 Bilder aus 13 Kleidungskategorien von kommerziellen Bildern bis zu privaten Bildern. Die Daten sind aufgeteilt in ein Training- und Validierung-Set.

Das Training mit großen Datensätzen erfordert viel Zeit und Hardwareresourcen. Um das Trainieren möglichst effizient zu realisieren wurde für das Traininieren der Objekterkennung auf eine Cloud-Computing Instanz ausgelagert. Cloud-Computing bietet dem Anwender on-demand Zugriff auf Serverinstanzen an, welche sich über SSH steuern lässt. In Verbindung mit einer NVIDIA GTX 1080 TI GPU und der TensorFlow (TF) Object Detection API [16] kann das Trainieren der Objekterkennung in einer kurzen Zeit erfolgen im Vergleich zu Trainingsaufwänden ohne GPU. Diverse Cloud-Computing Plattformen bieten Nutzern von Lehrbetrieben freie Kredits an, welche für die Verwendung des Servers frei genutzt werden kann. Ziel ist das Training kostengünstig und schnell zu gestalten. Der Trainingsfortschritt wird in einem Log festgehalten, welche später über TensorBoard

[15] visualisiert werden kann. Mit TensorBoard können Evaluationsmetriken und Objekterkennungen visuell dargestellt werden.

Um das trainierte Model dann der Hardware anzupassen muss das Model für das Deployment auf den RaspberryPi mit Unterstützung der Edge-TPU portiert werden. Mit dem *TF-Lite Converter* und *EdgeTPU Compiler* [1] wird das Model in ein TF-Lite Model, welches für die EdgeTPU optimiert ist, übersetzt[9]. Das Model muss in einer quantisierter Form (8-bit Integer) vorliegen, um die Edge-TPU effektiv auszunutzen. Dafür wurde eine Post-Training Quantisierung durchgeführt welche das 32-bit Floating-Point Model in ein 8-bit Integer Model konvertiert.

Um eine geeignete Bewertung der aktuellen Kleidung zu bekommen soll das aktuelle Wetter ermittelt werden. Dafür wird die OpenWeatherMap API (OWM) [21] verwendet. Mit einem HTTP-Request wird die aktuelle Wetterlage mit sämtlichen Informationen wie die Position, Temperatur, Windgeschwindigkeiten und -richtung eines gewählten Ortes erhalten. Diese werden für die endgültige Bewertung mit einbezogen. Ein minimales grafisches Interface (GUI) zur Visualisierung und Nutzereingaben wurde mit Qt [22] umgesetzt. Qt ist eine freie Open-Source Framework zur Erstellung von grafischen Oberflächen. Mit OpenCV [17], einer Bibliothek mit Fokus auf Echtzeit Computer-Vision, welche diverse Plattformen und Programmiersprachen unterstützt.

1.1 Zielsetzung$_{(L)}$

Im folgenden wird eine Grundlage geschaffen für die Umsetzung dieser Echtzeitanwendung. Verwendet werden folgende Hardware: der Raspberry Pi, die Kamera und der USB Beschleuniger. Mit der limitierten Rechenleistung und dem Beschleuniger soll eine ausreichend schnelle Anwendung geschaffen werden, die in Echtzeit agieren soll. Die Grundstruktur des Systems sowie die Auswahl der Frameworks und der Architekturen für Software muss mit Hinsicht auf dieses Ziel, des minimalen Ressourcenverbrauchs, untersucht und umgesetzt werden.

Mit einem Proof-of-Concept soll auf den zuvor genannten Grundlagen eine Objekterkennung entstehen, welches in Echtzeit die aktuelle Kleidung erkennt und in Verbindung mit aktuellen Wetterdaten eine Bewertung abgibt, ob die Kleidung zum Wetter passt.

1.2 Struktur der Arbeit$_{(N)}$

In Kapitel 2 werden Anforderung für die Hardware beschrieben. Die Hardware umfasst drei wesentliche Bauteile, bestehend aus das auswählte Embedded System, in diesem Fall ein Raspberry Pi 4, die Kamera für die Bilderfassung sowie der USB Beschleuniger, welches das Objekterkennungsmodel beschleunigt.

Kapitel 3 beinhaltet den Teil der Objekterkennung. In diesem Kapitel wird die Auswahl der Architekturen beschrieben und Informationen zum Datensatz näher beschrieben. Die Bewertungskriterien der COCO-Metrik werden erläutert und diverse Tools zur Umsetzung des Projektes werden vorgestellt. Des Weiteren wird beschrieben wie das Trainieren einer Objekterkennung auf einer Cloud-Computing Instanz erfolgt, sowie die Nutzung von Docker. Anschließend werden Optimerungsansätze vorgestellt.

In Kapitel 4 werden die Prozesse und Abläufe auf dem Embedded System beschrieben. Hierfür wird eine Pipeline-Architektur verwendet für das Verarbeiten der Kameradaten. Diese werden dann für die Objekterkennung vorbereitet und evaluiert. Zusätzlich werden die verwendeten Tools für die Bearbeitung der Pipeline-Sequenz mit OpenCV, die Gestaltung und Verwendung der GUI mit Qt5 und zuletzt die Verwendung der OpenWeatherMap API beschrieben. Ebenfalls wird hier der verwendete Algorithmus für die Bewertung der Bekleidung geschildert.

Kapitel 5 beinhaltet die Evaluation der einzelnen Bestandteile der Seminararbeit. Hier wird die trainierte Objekterkennung mit den geführten Logdaten in TensorBoard analysiert und bewertet. Im weiteren Teil wird die Performance der gesamten Objekterkennung auf dem Embedded System näher betrachtet.

In den letzten beiden Kapiteln, 6 und 7, werden die erzielten Ergebnisse in Anbetracht möglicher zukünftiger Arbeiten analysiert. Im Fazit wird der Projektverlauf kurz zusammengefasst und die erzielten Erfolge welche Erfolge reflektiert.

2 Hardware

Dieses Kapitel beschreibt die genutzen Hardwarekomponenten. Die Hardware für diese Seminararbeit besteht aus einem Raspberry Pi 4, einem USB ML Beschleuniger sowie einer Kamera für den Raspberry Pi.

2.1 Embedded System$_{(N)}$

Als Embedded System steht ein Raspberry Pi 4 (siehe Abb. 1) zur Verfügung. Dieser besitzt einen Broadcom BCM2711 SoC mit einem 64-bit quad-core ARM Cortex-A72 Prozessor und 2 GB RAM. Für dieses Projekt relevanten Ports ist zu einem die CSI Schnittstelle für die Kamera und die USB 3.0 Schnittstelle mit dem der USB Beschleuniger von Google verbunden ist.

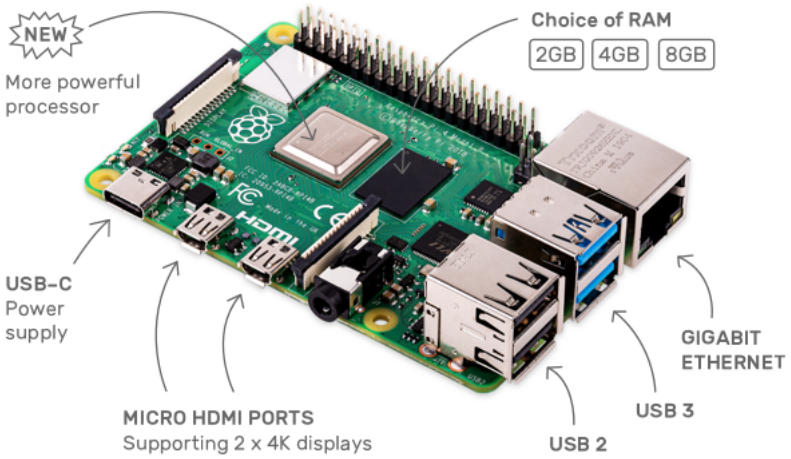

Abbildung 1: Raspberry Pi 4 mit annotierten Ports.

Der Raspberry Pi 4 besitzt neben der Gigabit Ethernet Schnittstelle auch Onboard Wireless Networking um eine Verbindung mit dem Internet zu gewährleisten. Diese wird benötigt, um HTTP-Requests zu erzeugen, um aktuelle Wetterdaten zu erhalten.

2.2 Kamera$_{(N)}$

Verbunden mit dem Raspberry Pi über die Kameraschnittstelle ist das hauseigene Kamera Modul (siehe Abb. 2) in der zweiten Version. Diese wurde im April 2016 veröffentlich

und besitzt einen Sony IMX219 8-megapixel Sensor für die Bilderfassung. Bilder sowie Videos sind in Farbe und diversen Auflösungen möglich.

Tabelle 1: Unterstütze Auflösungen und Features der Raspberry Pi Kamera v2.

#	Auflösung	Ratio	Frame Rate	Video	Bild	Field of View	Binning
1	1920x1080	16:9	0.1-30fps	x		Partial	None
2	3280x2464	4:3	0.1-15fps	x	x	Full	None
3	3280x2464	4:3	0.1-15fps	x	x	Full	None
4	1640x1232	4:3	0.1-40fps	x		Full	2x2
5	1640x922	16:9	0.1-40fps	x		Full	2x2
6	1280x720	16:9	40-90fps	x		Partial	2x2
7	640x480	4:3	40-90fps	x		Partial	2x2

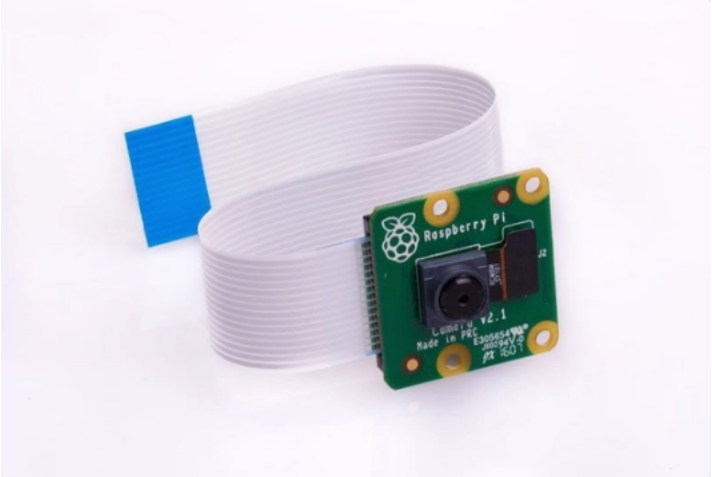

Abbildung 2: RasberryPis Kamera Modul v2.

Die verwendete Auflösung von 640×480 Pixeln ist genug um ein gut detailiertes Bild darzustellen. Für der Inferenzzeit hat die Bildgröße keine Auswirkung, da die Objekterkennung mit einer festen Bildgröße arbeitet, wobei im Vorraus die Bildgröße verändert werden muss. Die Veränderung der Bildgröße hat dabei keine signifikanten Performance-Auswirkungen.

2.3 Beschleuniger für Maschinelles Lernen$_{(L)}$

Der USB Beschleuniger von Google (siehe Abb. 3) fügt dem System einen Edge-TPU Koprozessor über eine USB-Schnittstelle hinzu. Die entwickelte Anwendungsspezifische

integrierte Schaltung (ASIC) ist in der Lage bis zu 4 Billionen Fix-Point, in unserem Fall 8-Bit Integer Datenformat, Operationen in der Sekunde durchzuführen, dadurch lassen sich Rechenoperationen des ML auf die Edge-TPU ausgelagern, dadurch lassen sich die Inferenzzeiten deutlich verkürzen werden. Für die effektive Nutzung des Beschleunigers sollte dieser in an einem USB 3.0 Port verbunden sein und die Objekterkennung muss angepasst werden. Das Vorgehen der Anppassung wird im nächsten Kapitel beschrieben.

Abbildung 3: Googles USB Beschleuniger.

3 Objekterkennung

Die Objekterkennung ist eine Technologie im Bereich Computer Vision und Image Processing, welches versucht Objekte in einem Bild zu erkennen und zu klassifizieren. Im Projektkontext wird die Objekterkennung über einen Deep-Learning Ansatz genutzt um die Bilddaten auszuwerten.

3.1 Deep Learning Ansatz$_{(N)}$

In dieser Seminararbeit wird primär die SSD Methode/Architektur [19] als Objekterkennung verwendet. Diese unterscheidet sich mit anderen State-of-the-Art Objekterkennungsmethoden, die getrennt Bounding-Boxen und Klassifizierungen durchführen, es müssen also $1 + m$ (Eine Bounding-Boxen Inferenz und m Klassifizierungen) Inferenzen durchgeführt werden, wobei eine SSD nur eine Inferenz benötigt für die gleiche Anzahl von Bounding-Boxen und die zugehörigen Klassifizierungen.

3.1.1 SSD$_{(N)}$

Der SSD Ansatz produziert eine endliche Menge von Bounding-Boxen und Klassifizierungswahrscheinlichkeiten relativ zu den Default-Boxen und zur Größe der genutzten Feature-Maps. Das SSD Netzwerk baut auf einer reduzierten Basis-Netzwerk-Achitektur auf, wie z.B. VGG-16 [25], welches als Klassifizierer vorgeschlagen wird.

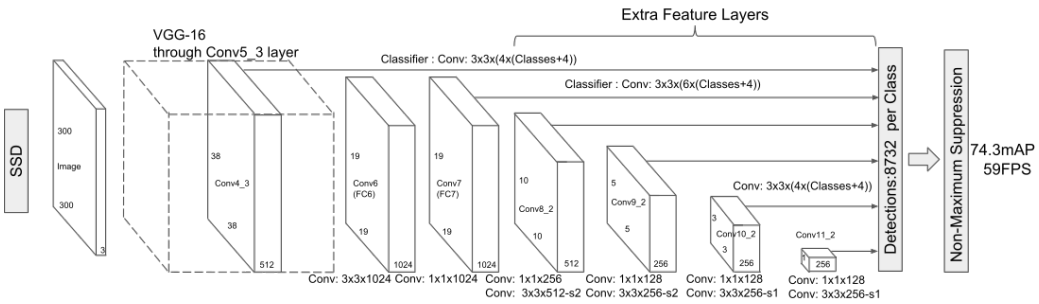

Abbildung 4: SSD Model (Mit VGG-16 als Basis-Model).

Zwei weitere Faltungs-Schichten (*convolutional feature layers*) mit den Kernelgrößen $3 \times 3 \times (n \times 4)$ und $3 \times 3 \times (n \times k)$ (k für die Anzahl der Klassen, n für die Anzahl der Erkennung

pro Zelle) werden jeweils an verschiedenen Enden (Feature-Layer) des reduzierten Basis-Netzwerkes hinzugefügt (siehe Abb. 4). Diese produzieren pro Zelle der abgezweigten Feature-Maps n Erkennungen, die aus den Wahrscheinlichkeiten pro Klasse, die relativen Offsets in x- und y-Richtung und relative Längen zu den dazugehörigen Default-Boxen bestehen (siehe Abb. 5).

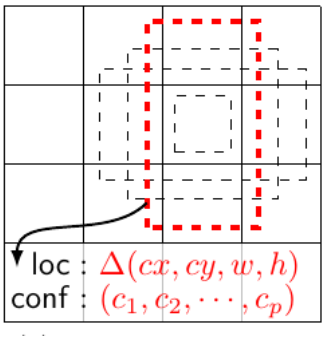

Abbildung 5: Eine potenzielle Abzweigung einer Feature-Map, die pro Zelle jeweils $n = 4$ Erkennungen besitzt. Pro Erkennung werden Klassenwahrscheinlichkeiten und relative Offsets/Längen gebildet.

Die Default-Boxen (*default bounding boxes*) werden pro Feature-Map Zelle vor dem Trainieren anhand der im SSD Paper [19] vorgestellten Verhältnisse und Berechnungen (*scales und aspect ratios*) ermittelt.

Es entstehen pro Bild mit der Größe 300×300 und dem Nutzen der SSD Methode 8732 Erkennungen, welche über die Anwendung des Non-Max-Suppression Algorithmus [6] reduziert werden. Folglich werden Nachbarschaftserkennungen, die potenziell das gleiche Objekt beschreiben, eliminiert (siehe Abb. 6).

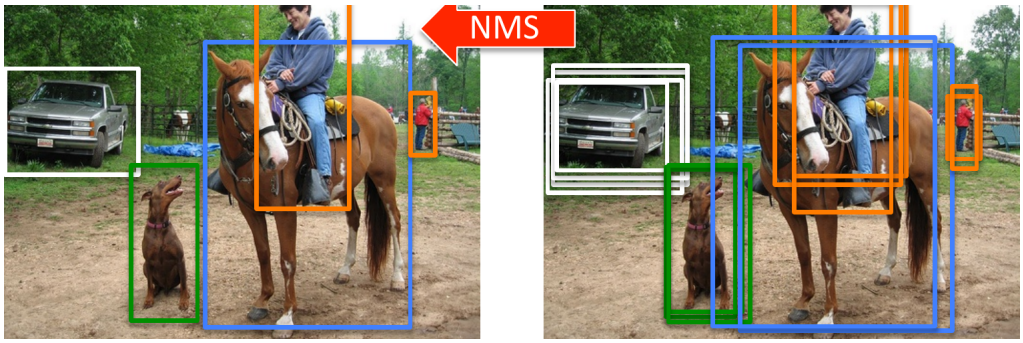

Abbildung 6: NMS Algorithmus Beispiel aus [6].

3.1.2 MobileNet V2$_{(L)}$

Die Nutzung der VGG-16 Basis-Netzwerk-Architektur, welche in SSD [19] genutzt wird, besitzt Bedenklichkeiten bezüglich der Performance (Inferenzzeit und Speicherbedarf) auf Embedded-Geräten. Untersuchungen in *Benchmark Analysis of Representative Deep Neural Network Architectures* [5] haben deutliche Performance Verbesserungen bezüglich Inferenzzeit und Speicherbedarf zwischen der MobileNet V2 Architekture [24] und VGG-16 auf einem NVIDIA Jetson TX1 gemessen. Die Inferenzzeit kann durch die Auswahl der Basis-Netzwerkarchitektur bis zu 90% reduziert werden.

Tabelle 2: Inferenzzeit (in ms) vs Batchgröße auf einem NVIDIA Jetson TX1 [5].

DNN	1	2
MobileNet-v2	20.51	14.58
VGG-16	151.52	169.92

Das Ziel von MobileNet v2 ist es trotz Reduktion der Netzwerkparameter, gute Lernfähigkeit zu erreichen. Im dem Paper wurden die *Inverted Residual Blocks* sowie die Idee von den *Linear Bottlenecks* demonstriert. Die *Inverted Residual Blocks* verbinden dabei zwei Layer, welche geringe Anzahl an Feature Maps besitzen 7. Das ist möglich indem *Depthwise Convolution Layer* verwendet werden, welches die Anzahl der Parameter eines *Inverted Residual Blocks* reduziert. Durch die *Inverted Residual Blocks* verliert das Netzwerk an Performance, da die verbunden Layer der Skip Connection die Anzahl der Feature Map wieder reduziert. Um den entgegen zu wirken, wurden die *Linear Bottlenecks* eingeführt. Diese fügen der letzten Faltung des Blocks eine Linearität hinzu bevor das Ergebnis mit der Aktivierungsfunktion ausgewertet wird.

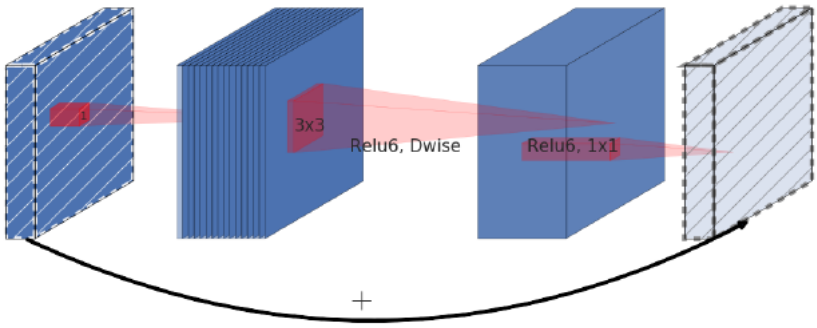

Abbildung 7: Inverted Residual Block, anders als normale Residual Blöcke werden hier zwei Layer verbunden mit wenigen Feature Maps.

Aus diesem Grund wird für die Seminararbeit, die in SSD [19] verwendete VGG-16 Basis-Netzwerk-Architektur ersetzt durch die von MobileNet V2 [24]. Die gesamte MobileNet V2 Architektur ist in Abb. 8 zu sehen. Anders wie im originalem Paper als die Input Imagegröße gewählt als $300 \times 300 \times 3$.

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

Abbildung 8: Layers der MobileNet V2 Architektur.

3.2 Datensatz$_{(L)}$

Als Datensatz für das Trainieren und Validieren wurde der DeepFashion2 [26] Datensatz verwendet. Durch die Größe des Datensatzes wird eine große Varianz und Toleranz dem Model beigebracht, so dass verschiedenste Kleidungsstücke klassifiziert werden können. Der Datensatz hat dabei eine hohe Varianz aus verschiedenen Aspekten wie Kameraposition, Skalierung der Objekte, sowie verdeckte und abgeschnittene Objekte, welches das Model auf Kleidungsstücke generalisiert. Der Datensatz besteht aus über 390000 Trainingsbildern, sowie über 34000 Validierungsbildern. Neben den Bildern sind auch die Labels, die auch Informationen über Bounding-Boxen besitzt, gegeben. Diese sind in einer JSON-Datei vermerkt und werden extrahiert für die Berechnung der Bewertungsmetriken, welche in der nächsten Sektion näher erläutert werden. Die Unterteilung der Kategorien ist in Abbildung 9 sichtbar. Neben den Informationen der Bounding-Boxen und Labels sind auch Landmark-Daten für Bildsegmentierungen vorhanden, die für diese Seminararbeit nicht von Bedeutung sind.

Für die einfachere Handhabung des Datensatzes wurden die Daten in *Shards* aufgeteilt.

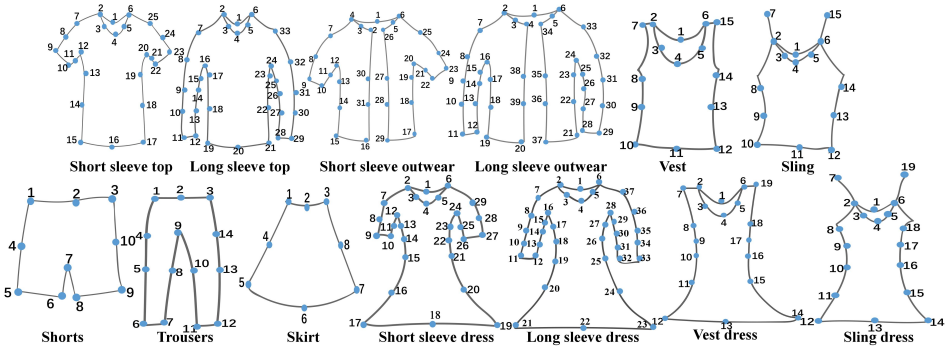

Abbildung 9: Unterteilung der Kategorien des DeepFashion2 Datensatzes.

Ein Shard bündelt mehrere Daten in eine Datei. Vorteilhaft ist die Reduktion des Transportaufwands, da der Datensatz in kompakter Form vorhanden ist.

3.3 Bewertungskriterien$_{(L)}$

Für die Bewertung des Models wird die *COCO Object Detection Metrik* [18] verwendet. Für die Objekterkennung werden weitere Metriken verwendet anstatt der üblichen Präzision und Recall. Erweitert wurden diese durch einen *Confidence Score* und der *Intersection over Union* (IoU). Der *Confidence Score* beschreibt die Wahrscheinlichkeit, dass ein Objekt innerhalb einer Default Box ist. Die *IoU* beschreibt die überlappende Fläche der prognostizierten Bounding-Box und der Ground-Truth Bounding-Box geteilt durch die gesamte Fläche beider Bounding-Boxen (siehe Abb. 10).

Confidence Score und IoU werden verwendet um Festzustellen ob eine Klasssifizierung True- oder False Positive ist. Der Pseudo-Code 1 beschreibt die Definition von True- und False Positives.

```
for each detection that has a confidence score > threshold:

  among the ground-truths, choose one that belongs to the same class and has
  the highest IoU with the detection

  if no ground-truth can be chosen or IoU < threshold (e.g., 0.5):
    the detection is a false positive
  else:
    the detection is a true positive
```

Listing 1: Pseudo Code für Unterteilung in True- und False Positive

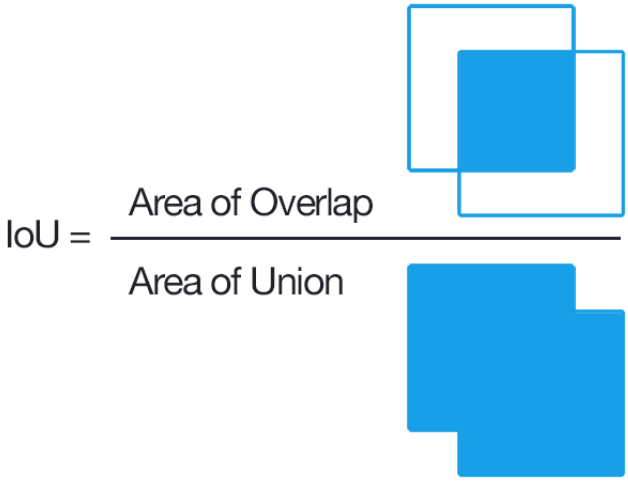

Abbildung 10: Berechnung des Intersection over Union-Werts

Die COCO Metrik verwendet den gemittelten Durchschnitt der Präzision und Recall. Diese werden weiter feiner unterteilt in weitere Metriken, welche die IoU sowie die Größe der erkannten Bounding-Boxen miteinbezieht (siehe Abb. 11).

```
Average Precision (AP):
   AP                  % AP at IoU=.50:.05:.95 (primary challenge metric)
   AP^IoU=.50          % AP at IoU=.50 (PASCAL VOC metric)
   AP^IoU=.75          % AP at IoU=.75 (strict metric)
AP Across Scales:
   AP^small            % AP for small objects: area < 32^2
   AP^medium           % AP for medium objects: 32^2 < area < 96^2
   AP^large            % AP for large objects: area > 96^2
Average Recall (AR):
   AR^max=1            % AR given 1 detection per image
   AR^max=10           % AR given 10 detections per image
   AR^max=100          % AR given 100 detections per image
AR Across Scales:
   AR^small            % AR for small objects: area < 32^2
   AR^medium           % AR for medium objects: 32^2 < area < 96^2
   AR^large            % AR for large objects: area > 96^2
```

Abbildung 11: Weitere Unterteilung der COCO Metriken.

3.4 Tools$_{(N)}$

Für das Erstellen und Trainieren der Objekterkennung wurden verschiedene Tools verwendet. TensorFlow [4] bietet eine ML Framework an und wurde von Google im Jahr

2015 veröffentlich. TensorFlow-Lite [13] vereinfacht das Deployment eines ML-Models auf Embedded Systems. Die Object-Detection API von TensorFlow [16] erleichtert das Definieren eines Objekterkennungsmodels.

3.4.1 TensorFlow und TensorFlow Lite$_{(L)}$

TensorFlow [4] ist eine datenorientierte ML Framework von *Google Brain*. Mathematische Operationen werden in einem *Computational Graph* dargestellt, welches sich unter anderem Differenzieren, Optimieren und in serialisierbarer Form speichern lässt.

Die *Keras API* der TensorFlow Framework bietet einfache Schnittstellen an, die schnelles Protypen eines ML-Model ermöglicht.

Im Projektkontext wurde die Keras API nur indirekt genutzt, da das Erstellen des Objecterkennungsmodels über die *Tensorflow Object Detection API* erfolgt.

Unter anderem wird *TensorFlow-Lite* genutzt um das trainierte Model auf dem Embedded Sytem bereitzustellen. Es erfordert eine Konvertierung des vortrainierten Models in ein TensorFlow-Lite Model (.tflite), welches mit dem *TensorFlow Lite Converter* erfolgt. Nachfolgend wird das konvertierte Model mit dem *TensorFlow-Lite Interpreter* ausgeführt, wobei das Model mit Hardware optimierten Operationen ausgeführt werden kann.

3.4.2 TensorFlow Object Detection API$_{(N)}$

Die *TensorFlow Object Detection API* ermöglicht über eine Spezifikationsdatei, ein Model zu konstruieren und zu trainieren.

Die Spezifikationsdatei enthält Informationen bezüglich der Art (z.B. SSD, ...), Basis-Netzwerk-Architektur (MobileNet V2, VGG-16, ...) und die zugehörigen Hyperparameter (Anzahl der Klassen, Default-Bounding-Box Parameter, ...) und Trainingsparameter (Ort der Trainingsdaten, Anzahl der Epochen, Optimizer, ...).

3.5 Trainieren in der Cloud$_{(N)}$

Das Trainieren mit spezialisierter Hardware und Resourcen kann das Trainieren beschleunigen, wodurch hohe Kosten (Stromkosten, Hardwarekosten) entstehen können. Deswegen wurde das Trainieren in einer Cloud-Computing Instanz und unter regelmäßiger Beaufsichtigung durchgeführt.

Eine Cloud-Computing Instanz mit einer NVIDIA® GTX 1080 Ti (11GB GDDR5X Speicher), 4x Intel® 3.00 GHz CPU und 12GB DDR4 Speicher wird genutzt um das SSD+MobileNetV2 Model zu trainieren. Über TensorFlows Docker Image [3] konnte eine schnell-lauffähige Laufzeitumgebung bereitgestellt werden, die bereits die benötigten Bibliotheken besitzt. Dadurch konnte der Einstieg in das Training vereinfacht werden.

Über die Hochverfügbarkeit und gute Netzwerkanbindung des Datencenters wird das Herunterladen und Vorverarbeiten der Trainingsdaten signifikant beschleinigt.

3.6 Optimierung$_{(N)}$

Um das trainierte TensorFlow Model mit TPU Beschleinigung auszuführen, wird eine Post-Quantisierte Variante des Models benötigt, da die EdgeTPU (der Coral USB Accelerator) nur Integer-Operationen unterstützt.

Unter Quantisierung versteht man im Bereich ML, das Reduzieren des Speicherbedarfs (Modelgröße) und das Erhöhen der Performance des Models. Ein typisches Model besteht aus einem Operationsgraphen und Gewichten (Weights), die üblicherweise als 32-bit Floatingwert gespeichert werden. Eine Quantisierung versucht die 32-bit Floatingwerte in Integer Datentypen (z.B. 8-bit Integers) zu Konvertieren ohne die Präzision und Genauigkeit des Models signifikant zu reduzieren. Dies kann während des Trainierens (*quantization-aware training*) oder nach dem Trainieren (*post-training quantization*) erfolgen. Das Konvertieren kann nicht nur den Speicherbedarf reduzieren, sondern auch die generelle Peformance (kürzere Inferenzzeiten) steigern, da Integeroperationen auf Embedded Systemen signifikant performanter sind als Floatingoperationen.

Nach der TensorFlow Lite Konvertierung, Post-Quantisierung und eine weitere Delegationsumwandlung (EdgeTPU Compiler [1]), die versucht TensorFlow-Lite Operationen mit Hardware-optimierte (EdgeTPU) Varianten zu ersetzten, konnten 95% der Operationen auf den USB-Beschleuniger ausgelagert werden (siehe Abb. 12).

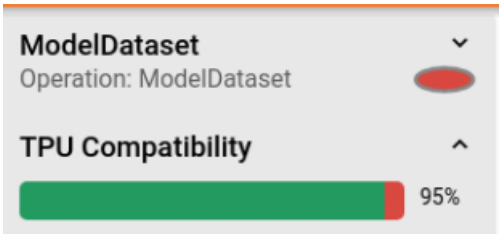

Abbildung 12: Operationskompatibilität mit der EdgeTPU.

4 Software

In diesem Kapitel wird die Software näher beschrieben. Von der Erfassung des Bildes, über die Inferenz mit der Objekterkennung bis zur Darstellung des annotierten Bildes werden die genutzen Frameworks sowie die Nutzung der Software dargestellt.

4.1 Pipeline-Architecture$_{(L)}$

Das Prozessing und Analyse der Kameradaten werden über eine Pipeline-Architektur (sieht Abb. 13) ausgewertet. Bilddaten werden von der Kamera über die OpenCV Framework ausgelesen. Das ausgelesene Bild hat die Dimension $640 \times 480 \times 3$ (RGB-Bild), welches für die Weiterverarbeitung nicht geeignet ist. Infolgedessen wird das Bild auf die Dimension $300 \times 300 \times 3$, über die Lanczos Interpolation, verkleinert. Die Objekterkennung wird nun auf der kleineren Variante des Input-Bildes ausgeführt. Diese gibt uns relative Bounding-Boxen und deren Klassenscores zurück. Während der Scan-Phase werden die Ergebnisse in annotierter Bildform über die GUI dargestellt.

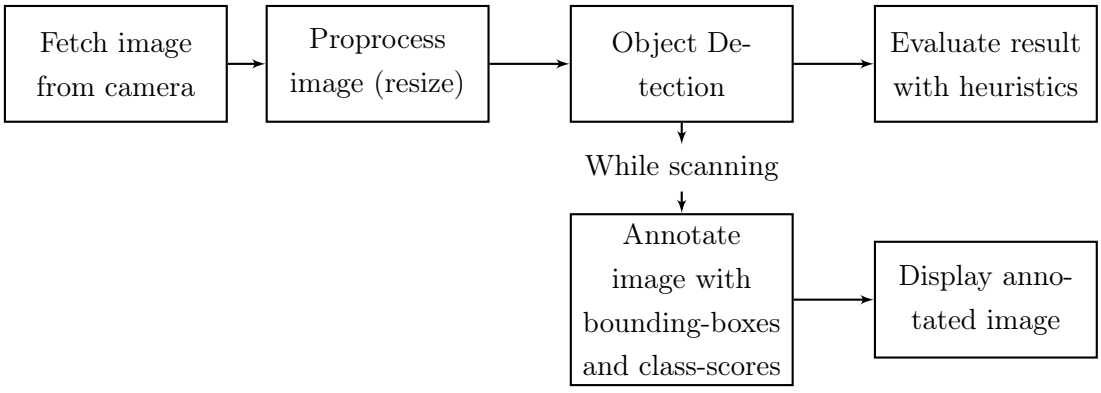

Abbildung 13: Pipeline-Archtektur der Bildverarbeitung

4.2 OpenCV$_{(N)}$

Aus einer empirischen Messung wurde festgestellt, dass die PiCamera [2] Bibliothek im Vergleich zur OpenCV Framework [17] längere Bearbeitungszeiten benötigt, um Bilddaten aus der Kamera zu entnehmen. Die OpenCV Framework ist eine Open-Source Bibliothek für Computer-Vision. Die Framework ermöglichtBilddaten aus der Kamera abzugreifen und zu bearbeiten. Der Zugriff auf die Bilder der Kamera erfolgt mit der Python API von OpenCV, wobei die Bilddaten als Numpy-Arrays gespeichert werden. Dieses erleichtert den Umgang mit Bilderdaten, da die Objekterkennung mit Numpy-kompatiblen Tensoren arbeitet.

4.3 Qt$_{(N)}$

Qt ist ein Framework zur Entwicklung von grafischen Oberflächen. *Qt Quick* ist dabei eine Subtechnologie von Qt, welche die Gestaltung der GUI über eine Markup-Language (QML) vereinfacht. Die verwendete Version in dieser Seminararbeit ist Qt 5.

Die fertige GUI (siehe Abb. 14) besitzt folgende Struktur:

- Aktueller Video Stream für die Anzeige des Bildes über die Kamera.
- Anzeige des aktuellen Wetters.
- Eingabefeld für die Stadt von der das Wetter genutzt werden soll.
- Eingabefeld für die Abfrage des Nutzers ab wann dieser bevorzugt längere Kleidung zu tragen.
- Button für das Starten der Evaluierung.
- Evaluationsfeld für die erkannten Labels.
- Emoji für die Bewertung der Bekleidung in Hinblick auf das aktuelle Wetter.

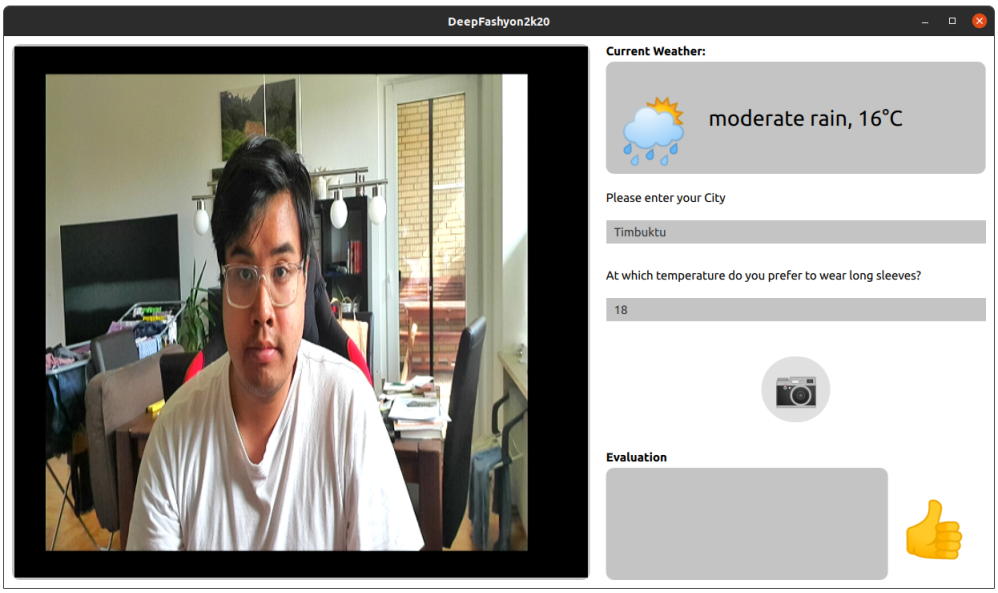

Abbildung 14: GUI für Interaktion mit dem Nutzer.

4.4 OpenWeatherMap(L)

OpenWeatherMap [21] ist ein proprietärer Online-Service, welches Wetterdaten bereit stellt. Die *One Call API* [20] erlaubt es Nutzern, nach der kostenlosen Registrierung und des personalisiertem API-Token, einen HTTP-Request zu erzeugen, um die Wetterdaten zu erhalten. Für den Request ist die Position in Längen- und Breitengrade anzugeben, diese kann durch einen weiteren Request über die *CurrentCurrent Weather Data API* erhalten werden, bei der man über den Namen der Stadt, die Längen- und Breitengraden erhält. Die HTTP-Requests werden mit Hilfe der *Requests* Bibliothek [23] erzeugt.

In Verbindung mit der GUI werden die Wetterdaten zum Start des Programmes geladen. Standartmäßig werden die Wetterdaten aus Hamburg abgefragt, und dann wenn der Scan-Button betätigt wird. Hierfür wird die vom Nutzer eingetragene Stadt ausgelesen.

4.5 Bekleidungsdaten Auswertung(L)

In Abbildung 15 wird der Decision-Tree beschrieben, welche die Bewertung für die erfassten Bekleidungsstücke mit der aktuellen Wetterlage und den Eingaben des Nutzers evaluierst. Die Ergebnisse der erfassten Kleidungsstücke werden hier aufgeteilt in Ober-

und Unterteile. Die von der Objekterkennung erkannten Ober- und Unterteile, mit höchster Wahrscheinlichkeit, werden selektiert. Die Heuristik schließt somit Kleidungskombinationen wie ein kurzes T-Shirt über einem Pullover aus, jedoch auch die Möglichkeit, dass man zu Kleidern noch ein langärmliges Shirt kombinieren kann. Über die aktuellen Wetterdaten wird die gefühlte Temperatur des angegeben Ortes mit der Angabe des Nutzers, über seine Wohlfühltemperatur, verglichen. Da der Datensatz jedoch nicht mit dem Material der Kleidung gelabelt ist, wird bei Regenwetter immer ein *Thinking Emoji* ausgegeben. Für die Anzeige einer positive Bewertung, einen *Thumbs Up Emoji*, ist bei wärmerem Wetter als die Wohlfühltemperatur ein kurzes Ober- und Unterteil nötig. Bei kälterem Wetter, entsprechend anderes herum, ein langes Ober- und Unterteil. Wenn unterschiedlich lange Ober- und Unterteil der Bekleidung erkannt wurden, so wird auch ein *Thinking Emoji* ausgegeben.

Tabelle 3: Ergebnisse der Bewertung und deren Bedeutung.

Emoji Bewertung	Bedeutung
👍	Ober- und Unterteil sind gut gewählt für das aktuelle Wetter
👎	Ober- und Unterteil sind schlecht gewählt für das aktuelle Wetter
🤔	Es regnet oder Ober- und Unterteil sind verschieden lang

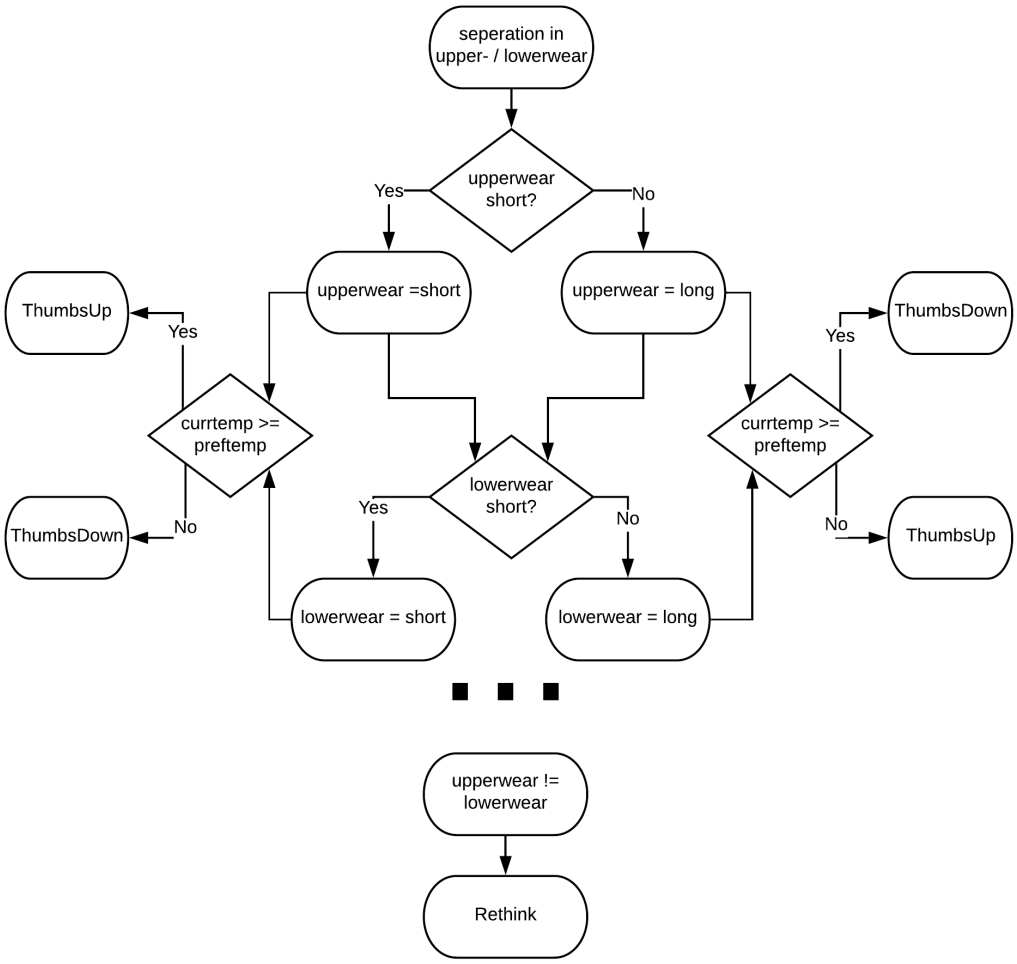

Abbildung 15: Decision-Tree der Bewertung.

5 Evaluation

Zur Evaluation der Seminararbeit werden die Objekterkennung (SSD mit MobileNet V2) und Peformance (Inferenzzeit, FPS) betrachtet.

5.1 Objekterkennung$_{(L)}$

Die Evaluation der Objekterkennung kann durch Log-Dateien in TensorBoard visualisiert werden. Der komplette Trainingsverlauf mit Daten über die angewandten Metriken und die eigentliche Objekterkennung ist hier darstellbar. Folgende Probleme sind während der Evaluierung der Daten aufgefallen:

- Große Objekte, welche große Bounding-Boxen besitzen werden im Vergleich zu kleineren Objekte schwer erkannt (siehe Abb. 16). Ein Grund dafür könnten ungüstige Feature-Map Größen oder Hyperparameter des Models sein, die bei der Nutzung von SSD mit MobileNet V2 entstehen.
- Bilder mit Artefakten, kontrastreichen und gleichfarbigen Hintergrund können die Erkennung deutlich (siehe Abb. 17) erschweren. Fehlende Bildaugmentierugen können die Ursache der fehlenden Erkennung sein.
- Objekte die zu kleinen Teilen teilweise erkennbar sind, werden schwer erkannt (siehe Abb. 18). Die Ursache dazu kann auf den Trainingsdatensatz zurückgeführt werden, da der im Verhältnis zu vollständig sichtbaren Objekten nur wenige Bilder vorhanden sind, die teilweise sichtbar sind.
- Kurze und lange Objekte werden oft verwechselt (siehe Abb. 19). Der Datensatz ist bereits vorannotiert und man kann nicht anhand des Datensatzes klar definieren ab wann die Urheber des Datensatzes Kleidungsstücke für lang erklären.

Abbildung 16: Problem mit zu großen Objekten.

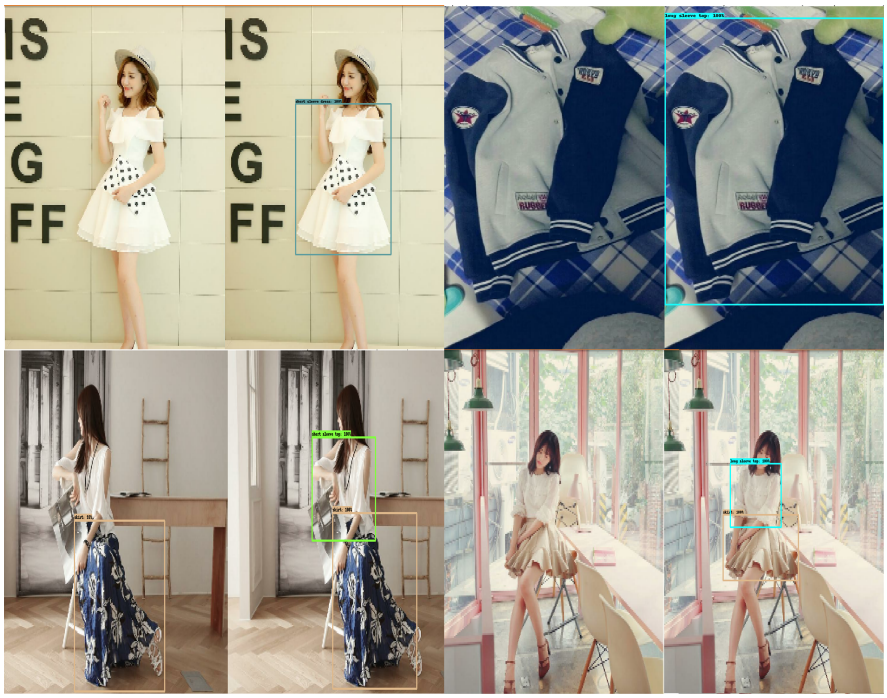

Abbildung 17: Problem mit konstrastreichen und gleichfarbigen Hintergrund.

Abbildung 18: Problem mit abgeschnittenen Objekten.

Abbildung 19: Problem mit der Definition von kurzen- oder langen Kleidungsstücken.

Folgende COCO Object Detection Metriken wurden nach dem Training evaluiert:

Tabelle 4: Ergebnisse der COCO Object Detection Metrik mit dem Deepfashion2 Datensatz.

mAP	mAP(large)	mAP(medium)	mAP(small)
0,5564	0,5608	0,436	-1

mAP@.50IoU	mAP@.75IoU	total loss
0,7282	0,6511	3,458

In Tabelle 4 ist zu erkennen, dass das Model ein Problem mit sehr kleinen Objekten besitzt. Der mAP(small)-Wert bleibt stetig bei -1, welches darauf hinweist, dass keine Objekte kleiner als 32×32 Pixel erkannt wurden. COCO selbst veranstaltet regelmäßig Herrausforderungen und protokolliert die Ergebnisse aller Modelle. Im Vergleich zu den Leaderboards [8] ist das in dieser Seminararbeit trainiertes Model im mittlerem Feld, welches ein gutes Ergebnis darstellt, in Betracht, dass das Model für ein Embedded System optimiert ist.

5.2 Performance (Inferenzzeit, FPS)$_{(N)}$

Die Modelgröße konnte über den *TensorFlow-Lite Converter* um ca. 70% reduziert werden, wobei die Post-Quantisierung eine Reduktion um den Faktor 4 erreicht, welches sich über die Änderung vom 32-bit Floating-Point Datentyps zu einem 8-bit Integer Datentyps erklären lässt (siehe Abb. 5).

Tabelle 5: Speichergröße (in MiB) des trainierten SSD+MobileNetV2 Models.

Vanilla FP-32	TF-Lite	**TF-Lite mit Post-Quantisierung (uint8)**
90	20	6

Im Vergleich zur Inferenzzeit hat die TensorFlow-Lite Konvertierung keinen signifikanten Einfluss. Erst nach der Operationsauslagerung auf die EdgeTPU wird eine deutliche Reduktion (ca. -95%) der Inferenzzeit gemessen (siehe Abb. 6).

Tabelle 6: Inferenzzeiten (in ms) des trainierten SSD+MobileNetV2 Models.

Vanilla TensorFlow	TF-Lite ohne EdgeTPU	**TF-Lite mit EdgeTPU**
200	201	10

6 Future Work

In Kapitel 5, der Evaluation, wurden diverse Probleme identifiziert. Einige Probleme sind auf den Datensatz zurückzuführen. Demzufolge könnte der Datensatz erweiter werden (z.B. mit Data-Mining). Modifikationen am Datensatz könnten zukünfig trainierte Modelle verbessern, indem weitere Daten hingefügt werden, die schlecht erkennbaren Bekleidungstücken enthält. Bildaugmentierungen, wie Kontrasterhöhungen oder Rauschen, könnten die Objekterkennung verbessern.

Mögliche Hyperparameteränderungen, wie die Größen der Feature-Maps oder die Input-Bildgröße, an der Architektur bzgl. SSD und MobileNet V2 könnte die Klassifizierung verbessern. Das Verwenden von anderen Basis-Netzwerk-Architekturen (z.B. MobileNet V3 [14]) an Stelle von MobileNet V2 kann auch hilfreich sein.

7 Fazit

Diese Seminararbeit beschäftigte sich mit der Entwicklung einer Bekleidungserkennung und Bewertung auf einem Embedded Systems auf Basis einer Deep-Learning Objekterkennung.

Die Hardware wurde durch den Veranstalter bereitgestellt, welche zur Realisierung dieser Seminarabeit hilfreich war. Seit der Festlegung des Themas ging es mit der Recherche und Umsetzung los. Für ein ML Model welches Kleidung erkennen soll, wurde der DeepFashion2 Datensatz genutzt, welcher eine große Vielfalt an Bekleidungsstücken besitzt, ausgewählt. Für die Echtzeitanwendung mussten ML Ansätze gewählt werden, welche Objekterkennungen akkurat und schnell durchführen können. Darauf wurde ein ML Model auf Basis der SSD Architektur in Kombination mit MobileNet V2, als Basis-Netzwerk-Architektur, festgelegt. Anschließend wurde die Objekterkennung, mit den vorbereiteten Trainingsdaten aus DeepFashion2, auf einer Compute-Cloud Instanz trainiert. Die Inferenzzeiten und die genutzten Hardware-Resourcen der Objekterkennung wurden, mit einer Post-Quantizierung und das Auslagern von Rechenoperationen auf die EdgeTPU, optimiert, hinsichtlich des Ziel einer Echtzeitanwendung. Darauffolgend wurde eine GUI realisiert, um die Interaktionen mit dem Nutzer zu vereinfachen.

Die Evaluation der entwickelten Anwendung hat ergeben, dass die vorrangetragenen Ziele erreicht wurden. In Bedacht, dass die entwickelte Anwendung auf ein Embedded System mit Softrequirements lauffähig sein soll, wurden gute Ergebnisse bei der Objekterkennung erzielt. Inferenzzeiten von durchschnittlich 10ms zeigen, dass die Anwendung echtzeittauglich ist.

Trotz der erfolgreichen Ausarbeitung wurden dennoch potenzielle zukünftige Ansätze identifiziert, welche weitere Arbeiten motivieren könnten.

Literatur

[1] *EdgeTPU Compiler*. https://github.com/google-coral/edgetpu. – [Online; accessed 01-July-2020]

[2] *PiCamera*. https://picamera.readthedocs.io/en/release-1.13/. – [Online; accessed 01-July-2020]

[3] *TensorFlow Docker Image*. https://hub.docker.com/r/tensorflow/tensorflow/. – [Online; accessed 01-July-2020]

[4] Abadi, Martín ; Agarwal, Ashish ; Barham, Paul ; Brevdo, Eugene ; Chen, Zhifeng ; Citro, Craig ; Corrado, Greg S. ; Davis, Andy ; Dean, Jeffrey ; Devin, Matthieu ; Ghemawat, Sanjay ; Goodfellow, Ian ; Harp, Andrew ; Irving, Geoffrey ; Isard, Michael ; Jia, Yangqing ; Jozefowicz, Rafal ; Kaiser, Lukasz ; Kudlur, Manjunath ; Levenberg, Josh ; Mané, Dan ; Monga, Rajat ; Moore, Sherry ; Murray, Derek ; Olah, Chris ; Schuster, Mike ; Shlens, Jonathon ; Steiner, Benoit ; Sutskever, Ilya ; Talwar, Kunal ; Tucker, Paul ; Vanhoucke, Vincent ; Vasudevan, Vijay ; Viégas, Fernanda ; Vinyals, Oriol ; Warden, Pete ; Wattenberg, Martin ; Wicke, Martin ; Yu, Yuan ; Zheng, Xiaoqiang: *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. – URL http://tensorflow.org/. – Software available from tensorflow.org

[5] Bianco, Simone ; Cadene, Remi ; Celona, Luigi ; Napoletano, Paolo: Benchmark Analysis of Representative Deep Neural Network Architectures. In: *IEEE Access* 6 (2018), S. 64270–64277. – URL http://dx.doi.org/10.1109/ACCESS.2018.2877890. – ISSN 2169-3536

[6] Bodla, Navaneeth ; Singh, Bharat ; Chellappa, Rama ; Davis, Larry S.: *Soft-NMS – Improving Object Detection With One Line of Code*. 2017

[7] Context, Common O. in: *COCO Detection Metrics*. https://cocodataset.org/#detection-eval. 2015. – [Online; accessed 01-July-2020]

[8] Context, Common O. in: *COCO Detection Leaderboard Detection*. https://cocodataset.org/#detection-leaderboard. 2020. – [Online; accessed 02-July-2020]

[9] Coral: *Coral Compability Overview*. https://coral.ai/docs/edgetpu/models-intro/#compatibility-overview. 2019. – [Online; accessed 01-July-2020]

[10] Foundation, Raspberry P.: *Raspbery Pi Camera Module V2*. https://www.raspberrypi.org/products/camera-module-v2/. 2016. – [Online; accessed 01-July-2020]

[11] Foundation, Raspberry P.: *Raspbery Pi 4 Model B*. https://www.raspberrypi.org/products/raspberry-pi-4-model-b/. 2019. – [Online; accessed 01-July-2020]

[12] Google: *USB Accelerator*. https://coral.ai/products/accelerator/. 2016. – [Online; accessed 01-July-2020]

[13] Google: *Tensorflow Lite*. https://www.tensorflow.org/lite. 2020. – [Online; accessed 01-July-2020]

[14] Howard, Andrew ; Sandler, Mark ; Chu, Grace ; Chen, Liang-Chieh ; Chen, Bo ; Tan, Mingxing ; Wang, Weijun ; Zhu, Yukun ; Pang, Ruoming ; Vasudevan, Vijay ; Le, Quoc V. ; Adam, Hartwig: *Searching for MobileNetV3*. 2019

[15] Huang J, Sun C Zhu M Korattikara A Fathi A Fischer I Wojna Z Song Y Guadarrama S Murphy K.: *Speed/accuracy trade-offs for modern convolutional object detectors*. https://www.tensorflow.org/tensorboard. 2019. – [Online; accessed 01-July-2020]

[16] Huang J, Sun C Zhu M Korattikara A Fathi A Fischer I Wojna Z Song Y Guadarrama S Murphy K.: *Speed/accuracy trade-offs for modern convolutional object detectors*. https://github.com/tensorflow/models/tree/master/research/object_detection. 2020. – [Online; accessed 01-July-2020]

[17] Intel Corporation, Itseez: *Image Processing in OpenCV*. https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_table_of_contents_imgproc/py_table_of_contents_imgproc.html. 2000. – [Online; accessed 01-July-2020]

[18] Lin, Tsung-Yi ; Maire, Michael ; Belongie, Serge ; Bourdev, Lubomir ; Girshick, Ross ; Hays, James ; Perona, Pietro ; Ramanan, Deva ; Zitnick, C. L. ; Dollár, Piotr: *Microsoft COCO: Common Objects in Context*. 2014

[19] Liu, Wei ; Anguelov, Dragomir ; Erhan, Dumitru ; Szegedy, Christian ; Reed, Scott E. ; Fu, Cheng-Yang ; Berg, Alexander C.: SSD: Single Shot MultiBox Detector. In: *ECCV*, 2016

[20] OpenWeather: *One Call API*. https://openweathermap.org/api/one-call-api. 2012. – [Online; accessed 01-July-2020]

[21] OpenWeather: *OpenWeatherMap API*. https://openweathermap.org/api. 2012. – [Online; accessed 01-July-2020]

[22] Qt: *Qt Lite in Qt 5.9 LTS*. https://www.qt.io/blog/2017/05/31/qt-lite-qt-5-9-lts. 2017. – [Online; accessed 01-July-2020]

[23] Reitz, Kenneth: *Requests: HTTP for Humans*. https://requests.readthedocs.io/en/master/. 2019. – [Online; accessed 02-July-2020]

[24] Sandler, Mark ; Howard, Andrew G. ; Zhu, Menglong ; Zhmoginov, Andrey ; Chen, Liang-Chieh: MobileNetV2: Inverted Residuals and Linear Bottlenecks. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2018), S. 4510–4520

[25] Simonyan, Karen ; Zisserman, Andrew: Very Deep Convolutional Networks for Large-Scale Image Recognition. In: *International Conference on Learning Representations*, 2015

[26] Yuying Ge, Lingyun Wu Xiaogang Wang Xiaoou Tang Ping L.: *DeepFashion2*. https://github.com/switchablenorms/DeepFashion2. 2019. – [Online; accessed 01-July-2020]

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Gemäß der Allgemeinen Prüfungs- und Studienordnung ist zusammen mit der Abschlussarbeit eine schriftliche Erklärung abzugeben, in der der Studierende bestätigt, dass die Abschlussarbeit „— bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit [(§ 18 Abs. 1 APSO-TI-BM bzw. § 21 Abs. 1 APSO-INGI)] — ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt wurden. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen."

Quelle: § 16 Abs. 5 APSO-TI-BM bzw. § 15 Abs. 6 APSO-INGI

Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versicheren wir,

Herr Huy-Bao Le & Herr Paul Duy An Nguyen

dass wir die vorliegende Seminararbeit – bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit – mit dem Thema:

Bekleidungsauswertung auf einem Embedded System

ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt haben. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Hamburg, 05. Juli 2020

Huy-Bao Le Paul Duy An Nguyen