Hochschule Aalen

Projektbericht
Studiengang : Informatik

Entwicklung einer Augmented Reality Applikation zur Wiedererkennung bereits eingelernter Objekte

von

Michael Schlosser

75984

Betreuender Mitarbeiter : Dr. Marc Hermann

Einreichungsdatum : 3. August 2021

Eidesstattliche Erklärung

Hiermit erkläre ich, **Michael Schlosser**, dass ich die vorliegenden Angaben in dieser Arbeit wahrheitsgetreu und selbständig verfasst habe.

Weiterhin versichere ich, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben, dass alle Ausführungen, die anderen Schriften wörtlich oder sinngemäß entnommen wurden, kenntlich gemacht sind und dass die Arbeit in gleicher oder ähnlicher Fassung noch nicht Bestandteil einer Studien- oder Prüfungsleistung war.

Ort, Datum Unterschrift (Student)

Kurzfassung

Das Einlernen und Wiedererkennen von beliebigen Objekten ist im Bereich des Maschinellen Sehens immer noch ein aktuelles Thema. Ein exemplarischer Anwendungsfall hierfür, wäre dem Nutzer die Möglichkeit zu bieten durch eine Kamera ein Objekt einzulesen und das Kamerabild bei Erkennung des Objektes mit bildunterstützenden Elementen zu ergänzen.
Die geringe Rechenleistung von Mobile und Embedded Devices stellt jedoch den Einsatz dieser Technologie auf diesen Geräten vor eine Herausforderung. Verschiedene Ansätze, um die Rechenintensität eines Einlern-/ und Wiedererkennungsalgorithmus gering zu halten, sind bereits verfügbar und einsetzbar.
Im Rahmen dieser Arbeit wird eine Android-Applikation für mobile Endgeräte entwickelt, welche den im obigen beschriebenen Anwendungsfall erfüllen kann. Zum Einlernen und Wiedererkennen werden bereits trainierte Deep Learning Architekturen mithilfe der Methode „Modellgeneralisierung“ erweitert, um die Kernfunktionalität erbringen zu können.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Abkürzungsverzeichnis

1. Einleitung

Die Einleitung dient dazu, beim Leser Interesse für die Inhalte Praxissemesterberichts zu wecken, die behandelten Probleme aufzuzeigen und die zu ihrer Lösung entwickelten Konzepte zu beschreiben.

2. Grundlagen

Zur Lösung der Problemstellung sind verschiedene Grundlagengebiete zu beherrschen, die zum einen bereits im Rahmen des Informatikstudiums verinnerlicht werden konnten und zum anderen Gebiete und Themen, die erst im Laufe der Recherche und Implentierung der Projektarbeit erarbeitet worden sind.

2.1. Deep Neural Networks

Konventionelle Methoden des Maschinellen Lernens sind in der Verarbeitung von rohen Daten, wie beispielsweise den Pixeln von Bildern, ineffizient. Deep Learning bezeichnet ein Teilgebiet des Maschinellen Lernens, welches in diesen Anwendungsfällen eine bessere Alternative darstellt. Ein Deep Neural Network (DNN) ist ein mehrschichtiges Rechenmodell, welches sich vor allem in den Gebieten der Bild- und Spracherkennung als *state-of-the-art* etablieren konnte.
Beim Deep Learning wird zwischen Training und Inferenz unterschieden. Beim Training werden in das Netz annotierte Trainingsdaten gegegeben, welche vom Netz prozessiert werden. Daraufhin werden die Gewichte des Netzes kalibriert um die Merkmale der eingelesenen Trainingsdaten wiedererkennen zu können.
Beim Vorgang der Inferenz erzeugt das Netz eigenständig Ausgaben. Der produktive Einsatz von DNNs entspricht demnach dem Vorgang der Inferenz [6, 5].

2.1.1. Deep Convolutional Neural Networks

Bei den meisten Bildklassifizierungs- und Objektdetektionsapplikationen werden sogenannte Faltungsschichten (*engl.: Convolutional Layers*) verwendet, da sich diese als sehr effizient in diesem Anwendungsfall erweisen konnten.
In der Mathematik bezeichnet eine Faltungsschicht das Überlappen zweier Funktionen.
Eine Faltungsschicht besteht aus sogenannten Convolutional Filters mit der Größe $D_K \times D_K \times M$ wobei D_K der Höhe und Breite des Filters und M der Anzahl an Kanälen beziehungsweise der Tiefe des Filters entspricht. Der Filter gleitet nun über das Eingabebild F und multipliziert F mit dem Filter. Die Ergebnisse dieser Multiplikation werden als Skalarprodukt zu einem Wert zusammengefasst. Nach-

dem der Filter über alle Pixel des Bildes gegleitet ist ergibt sich eine neue kleinere Matrix, welchere als gefaltetes Merkmal (*engl.: Convolved Feature*) bezeichnet wird.

2.1.2. Mobilenet

Bei der rapiden Weiterentwicklung von DNN-Architekturen wird das Augenmmerk stets auf die Verbesserung der Genauigkeit des Modells gelegt. Da die Latenzzeit und Größe des Netzes zunehmen, ist der Einsatz auf Geräten mit limitierter Rechenleistung für Echtzeitsysteme nur schwer möglich.
Mobilenets sind DNNs, die speziell für Aufgaben im Bereich des Maschinellen Sehens auf mobilen Endgeräten und eingebetteten Systemen entwickelt wurden. Ein Mobilenet Modell basiert auf sogenannten Depthwise Separable Convolutions (DSCs), welche die Faltungsschichten in einem Deep Convolutional Neural Network (DCNN) hinsichtlich der Netzgröße und Inferenzzeit optimieren.
Der Rechenaufwand beim Prozessieren einer Faltungsschicht ist durch die hohe Anzahl an nötigen Multiplikationen enorm. DSCs berechnen die Faltung in mehreren Phasen. In der ersten PHase wird die sogenannte Tiefenfaltung angewendet, welche die herkömmliche Faltung lediglich auf einem Kanal berechnet. In der zweiten Phase wird die sogenannte Punktfaltung angewendet, wobei die Merkmale aus der Tiefenfaltung miteinander kombiniert werden. Dies verringert den Rechenaufwand deutlich.
Der Aufbau des Mobilenet v2, welches Anwendung im gegebenen Anwendungsfall findet, wird in Abbildung 2.1 dargestellt [3].

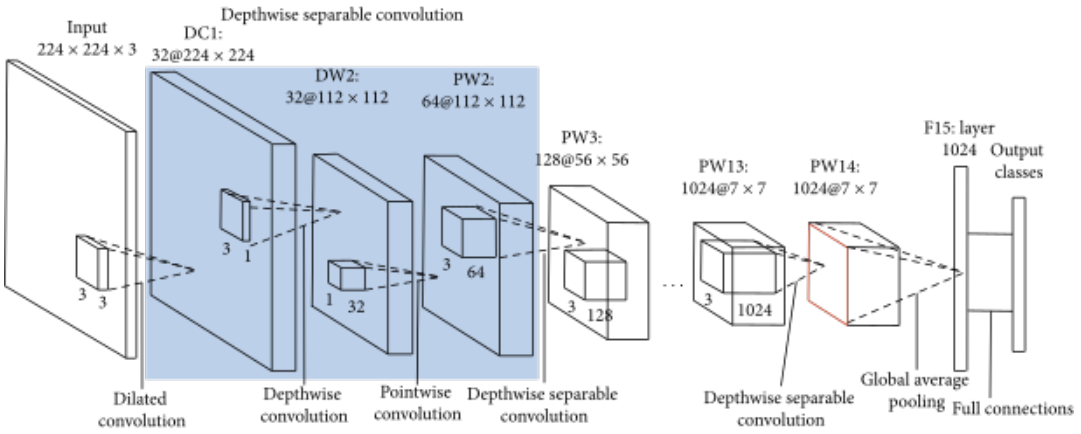

Abbildung 2.1.: Architektur eines Mobilenet v2
Quelle: https://www.hindawi.com/journals/cin/2020/8817849/fig2/ abgerufen am 30.07.2021

2.1.3. Imagenet

2.2. Wiedererkennung von Objekten

Das Einlernen und Wiedererkennen von Objekten ist im Bereich des Maschinelles Lernens noch heute eine Herausforderung für eingebettete beziehungsweise mobile Endgeräte, aufgrund der geringen Rechenleistung, die zur Verfügung steht. Es ist möglich diese Aufgabe mithilfe einer Cloudlösung auf einen externen leistungsstarken Server zu übertragen. Aufgrund von Nachteilen, wie beispielsweise Sicherheitsrisiken, die beim Übertragen von sensitiven Daten auftreten, wird im Rahmen dieser Arbeit eine Applikation programmiert, die den gegebenen Anwendungsfall mit der dem Endgerät zur Verfügung stehenden Hardware löst.
Ein naiver Ansatz, um dieses Problem lösen zu können ist ein Convolutional Neural Network (CNN) bei Eingang neuer Trainingsdaten erneut zu trainieren mitsamt den bereits vorhandenen Daten. Ein derart riesiger Lernvorgang ist jedoch nicht praktikabel und könnte mehrere Stunden oder sogar Tage dauern. Wenn lediglich die neuen Daten für das Training verwendet werden, resultiert dies im bekannten Phänomen des *Katastrophalen Vergessen* [9].

2.2.1. Katastrophales Vergessen

Neuronale Netze tendieren dazu beim Einlernen von aufeinanderfolgenden unterschiedlichen Aufgaben das bereits gelernte Wissen zu vergessen. Dieses Phänomen wird als Katatrophales Vergessen beziehungsweise als *Katastrophale Inferenz* bezeichnet. Die Ursache hierfür ist, dass Gewichte von Neuronen, die für eine vorherige Aufgabe wichtig sind, überschrieben werden, um die neue eingelernte Aufgabe zu erfüllen [4].
Für den gegebenen Anwendungsfall bedeutet dies, dass alte bereits eingelernte Objekte nicht mehr erkannt werden können, wenn ein bereits trainierter Datensatz mit unbekannten Daten weitertrainiert wird. Lösungsansätze für dieses Problem werden im Folgenden erklärt.

2.2.2. Transfer Learning

Jedes menschliche Gehirn, ist dazu fähig Wissen, welches für eine Aufgabe benötigt wird, auf eine andere Aufgabe zu übertragen. So ist es beispielsweise mit geringem Aufwand möglich das Autofahren zu erlernen, wenn man bereits Motorrad fahren kann. Da es eine ähnliche Aufgabe ist, wird nicht versucht die neue Aufgabe von Grund auf neu zu lernen, sondern das Wissen, welches in der Vergangenheit erlernt werden konnte auf ein neuen ähnlichen Anwendungsfall zu transferieren.

Im traditionellen Ansatz des Maschinellen Lernens wird ein Datensatz aufbereitet, welcher für eine bestimmte Aufgabe zugeschnitten ist.
Die grundlegende Aufgabe, die in dieser Arbeit zu erfüllen ist, ist jedoch das Einlernen von neuen Trainingsdaten. Hierfür kann der Ansatz des sogenannten Transfer Learnings Anwendung finden.
Beim Transfer Learning wird ein bereits trainiertes Machine Learning (ML)-Modell verwendet. Das bereits errungene Wissen kann hierbei weiterverwendet werden, um mit einem kleineren Trainingsdatensatz eine neue ähnliche Aufgabe erfüllen zu können. Erkannte Merkmale wie Formen, Kanten und Ecken können demnach durch den Wissenstransfer mit einem Modell geteilt werden. Bei dieser Vorgehensweise werden die unteren Schichten, welche die nicht-klassenspezifischen Merkmale erkennen eingefroren. Dies bedeutet, dass die Gewichte der Neuronen unveränderbar sind. Die oberen Schichten sind weiterhin trainierbar. Durch Training eines Datensatzes, welcher den neu einzulernenden Daten ähnelt, werden lediglich die klassenspezifischen Merkmale trainiert [9]. In Abbildung 2.2 wird diese Architektur mit dem Traditionellen ML-Ansatz verglichen.
Wenn stets neue Klassen eingelernt werden sollen und diese Daten sich nicht innerhalb einer Trainingsstapels befinden, tritt jedoch das Phänomen des Katastrophalen Vergessens auf und früher eingelernte Objekte werden überschrieben [2].

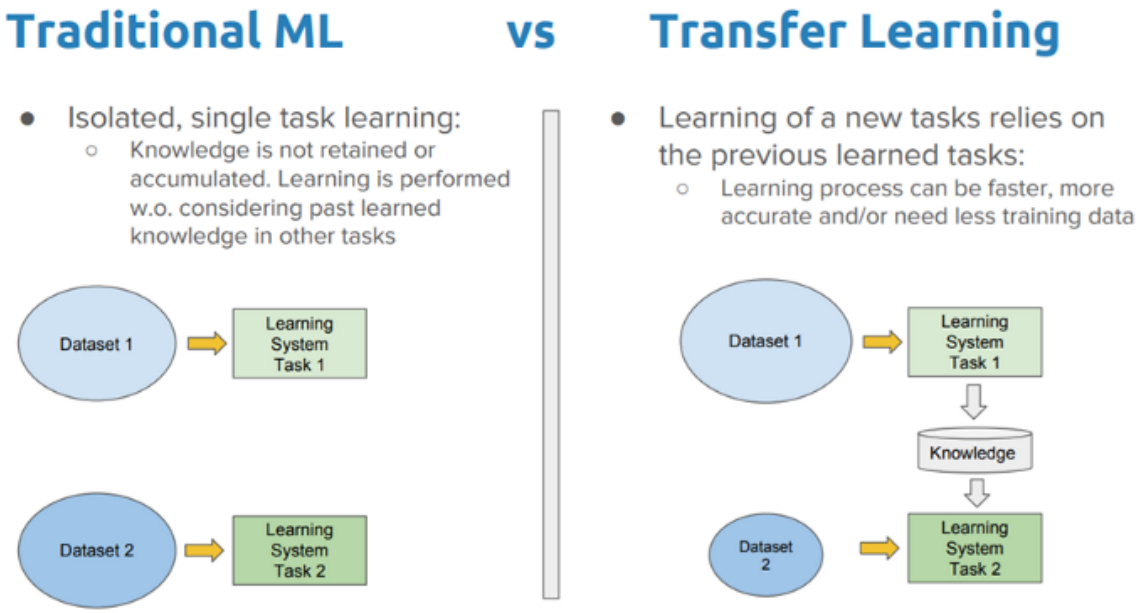

Abbildung 2.2.: Vergleich des Aufbaus eines Transfer Learning Models im Gegensatz zum traditionellen Maschinellen Lernen
Quelle: [8]

2.2.3. Latent-Replay-Algorithmus

Das grundlegende Konzept beim Latent-Replay-Algorithmus bildet das sogenannte *Replay* beziehungsweise *Rehearsal*. Bereits trainierte Trainingsdaten werden hierfür in einem separaten Datenbank persistent gespeichert und bei Eingang neuer Trainingsdaten ebenfalls mitgelernt. Das Problem hierbei ist jedoch, dass die Trainingsdauer massiv zunimmt, da in jeder Trainingsepoche mehrere Trainingsiterationen durch das Neuronale Netz notwendig sind. Zudem ergibt sich aus dem Abspeichern von alten Trainingsdaten ein Speicherproblem auf eingebetteten Systemen.
Der Latent-Replay-Algorithmus erweitert den Ansatz des Rehearsals, um eben diese Schwierigkeiten zu beheben. Hierfür werden nicht rohe Bilddaten abgespeichert sondern Aktivierungen von Neurononen innerhalb einer Schicht der Netzarchitektur. Bei einer Trainingsiteration werden neue Eingangsdaten durch Schichten des Netzen weitergeleitet, welche für das Erkennen von generischen Merkmalen zuständig sind, wie beispielsweise Ecken und Kanten. Anschließend werden diese Daten in die Latent-Replay-Schicht, welche die Speicherung der alten Trainingsdaten übernimmt, weitergeleitet, wobei die Eingangsdaten nun mit den Aktivierungen der alten Trainingsdaten vermischt werden. Die Schichten, die oberhalb der Latent-Replay-Schicht angesetzt sind, sind für die Erkennung von klassenspezifischen Merkmalen zuständig [7]. In Abbildung 2.3 wird der eben erklärte Aufbau dargestellt.
Das Prinzip des Transfer Learnings kann dementsprechend mithilfe des Latent-Replay-Algorithmus erweitert werden und dadurch das Phänomen des Katastrophalen Vergessens beheben, durch das Mittrainieren von alten Trainingsdaten.

2.3. Android

2.3.1. Activities und Fragments

2.3.2. Shared Preferences

Shared Preferences bezeichnen einen Key-Value-Store für primitive Daten in Form einer XML-Datei. Genutzt kann dies über die dafür zur Verfügung gestellten Android-Bibliothek, welche ein Shared Preferences-Objekt erzeugt. Dieses Objekt beinhaltet einen Zeiger auf den Pfad der jeweiligen Datei. Mithilfe von Shared Preferences können beispielsweise Konfigurationsparameter, die persistent abgespeichert werden sollen mithilfe einer einfachen Application Programming Interface (API) zugänglich gemacht werden

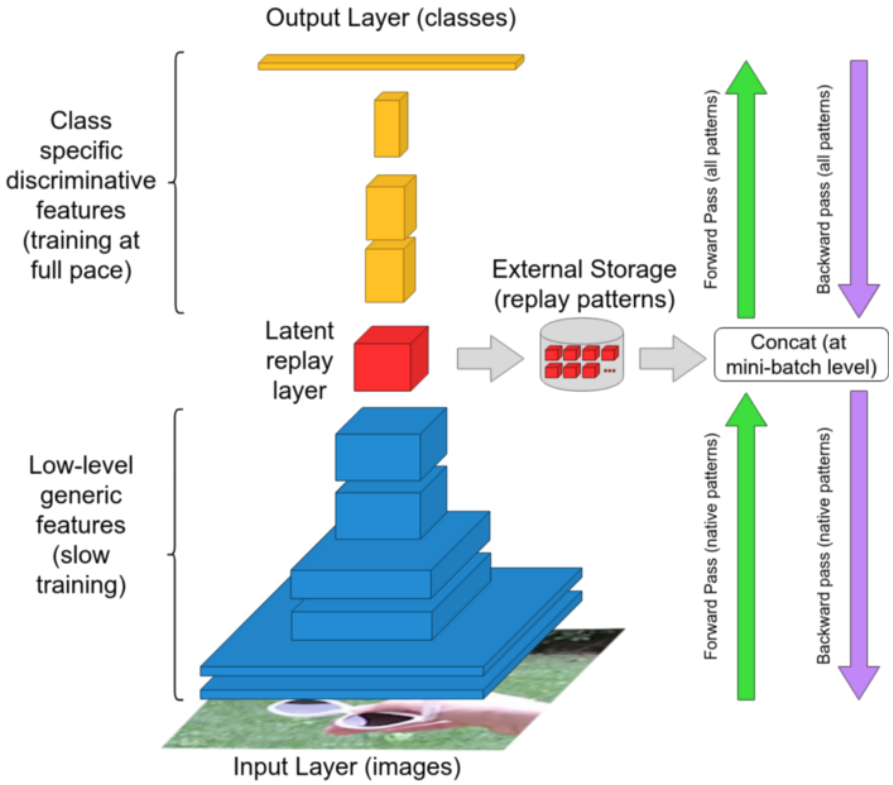

Abbildung 2.3.: Architekturdiagramm des Latent Replay
Quelle: [7]

3. Anforderungsanalyse

Zur Erarbeitung der Anforderungen der gegebenen Aufgabenstellung, werden diese hinsichtlich ihrer Umsetzungsrelevanz gegliedert und gewichtet.

- **Must-Have-Anforderungen** sind unbedingt umzusetzen. Sie umfassen die Kernfunktionalitäten, die zur Lösung der gegebenen Aufgabenstellung essentiell sind.
- **Should-Have-Anforderungen** beschreiben Eigenschaften des Systems, die vorteilhaft für die Lösung der gegebenen Aufgabenstellung sind und einen großen Mehrwert für die Software bieten, jedoch nicht zwingend erforderlich sind.
- **Could-Have-Anforderungen** sind optionale Anforderungen an Eigenschaften des Systems, die ebenfalls einen relevanten Mehrwert bieten können, welcher jedoch nicht zwingend erforderlich für die Lösung der gegebenen Aufgabenstellung ist.
- **Nice-To-Have-Anforderungen** sind ebenfalls optionale Anforderungen an Eigenschaften des Softwaresystems. Diese sind jedoch von untergeordneter Bedeutung.

Im Folgenden werden die Anforderungen an das zu entwickelnde Softwaresystem thematisch gruppiert und unterteilt in funktionale und nicht-funktionale Anforderungen.

3.1. Funktionale Anforderungen

Funktionale Anforderungen erklären, welche Funktionen und Dienste vom Software-System bereitzustellen sind und insbesondere die Beziehungen zwischen den Ein- und Ausgabedaten.

3.1.1. Lauffähigkeit als Android-App

Eine auf einem aktuellen Android-Betriebssystem lauffähige Applikation soll entwickelt und auf einem Smartphone in den Betrieb genommen werden. Es wird gefordert, dass der Nutzer des Software-Systems die im Smartphone integrierte Kamera nutzt und diese als Eingabe für das Einlernen und die Wiedererkennung eines Objektes nutzt. – *Gewichtung:* Must-Have-Anforderung.

3.1.2. Entwicklung einer graphischen Benutzeroberfläche

Es wird gefordert eine Graphische Benutzeroberfläche zur Verfügung zu stellen, mit welcher der Nutzer interagieren kann. Der Touchscreen des Smartphones sollte das Haupteingabegerät des Nutzers zur Steuerung der Applikation sein. – *Gewichtung:* Must-Have-Anforderung.

3.1.3. Einlernen von Objekten mithilfe einer Kamera

Es wird gefordert, dass ein Objekt mit der integrierten Smartphone-Kamera, bei guten Licht- und Kontrastverhältnissen eingelernt werden kann. Bei erfolgreicher Abspeicherung der Daten, soll dem Nutzer dies mitgeteilt werden. Der Nutzer hat nun die Möglichkeit das eingelernte Objekt mit Daten zu versehen, wie beispielsweise einem Namen. Die Informationen über das nun eingelernte Objekt werden in einer Datenbank persistent abgespeichert, um sie wieder abrufen zu können. – *Gewichtung:* Must-Have-Anforderung.

3.1.4. Wiedererkennung bereits eingelernter Objekte mithilfe einer Kamera

Ein bereits eingelerntes Objekt soll bei guten Licht- und Kontrastverhältnissen vom Softwaresystem in Echtzeit wiedererkannt werden. – *Gewichtung:* Must-Have-Anforderung.

3.1.5. Markierung der wiedererkannten Objekte

Wenn ein Objekt von der Applikation erkannt wird, soll es beispielsweise durch das Einblenden des Namens des Objektes dem Nutzer mitgeteilt werden, dass das eingelernte Objekt wieder auf dem Display des Smartphones zu sehen ist. – *Gewichtung:* Should-Have-Anforderung.

3.1.6. Graphisches Overlay bei wiedererkannten Objekten

Die beim eingelernten Objekt gespeicherten Daten sollen als ein graphisches Overlay auf dem Bildschirm des Smartphones dargestellt werden, wenn das Objekt erneut erkannt worden ist. – *Gewichtung:* Must-Have-Anforderung.

3.2. Nicht-funktionale Anforderungen

Im Folgenden werden die Einschränkungen und Qualitätsmerkmale an die Entwicklung und den Betrieb des Systems erklärt.

3.2.1. Lauffähigkeit auf einem Smartphone

Die Kompatibilität und ein angemessenes Laufzeitverhalten auf einem aktuellen Android-basierten Smartphone wird gefordert, um eine benutzerfreundliche Applikation bereitstellen zu können. – *Gewichtung:* Must-Have-Anforderung.

3.2.2. Dauer und Umstände des Erlernens der Objekte

Ein Objekt soll bei guten Lichtverhältnissen in annehmbarer Zeit eingelernt werden. – *Gewichtung:* Must-Have-Anforderung.

3.2.3. Benutzerfreundlichkeit

Die Graphische Benutzeroberfläche ist so intuitiv wie möglich zu gestalten, um die Bedienung des Systems ohne größeren Einarbeitungsaufwand erlernen zu können. – *Gewichtung:* Should-Have-Anforderung.

3.2.4. Wartbarkeit und Erweiterbarkeit

Änderungen und Erweiterungen des Software-Systems sollten mit hinnehmbarem Aufwand bewerkstelligt werden können. Demnach sollte der Quellcode modular, strukturiert und dokumentiert sein. – *Gewichtung:* Should-Have-Anforderung.

3.3. Anwendungsfälle

Die zu entwickelnde Applikation soll dem Hauptakteur „User“ die in Abbildung 3.1 dargestellten Anwendungsfälle erfüllen können. Eine genaue Beschreibung wurde in Anhang A verlagert, um den Lesefluss nicht zu stören.

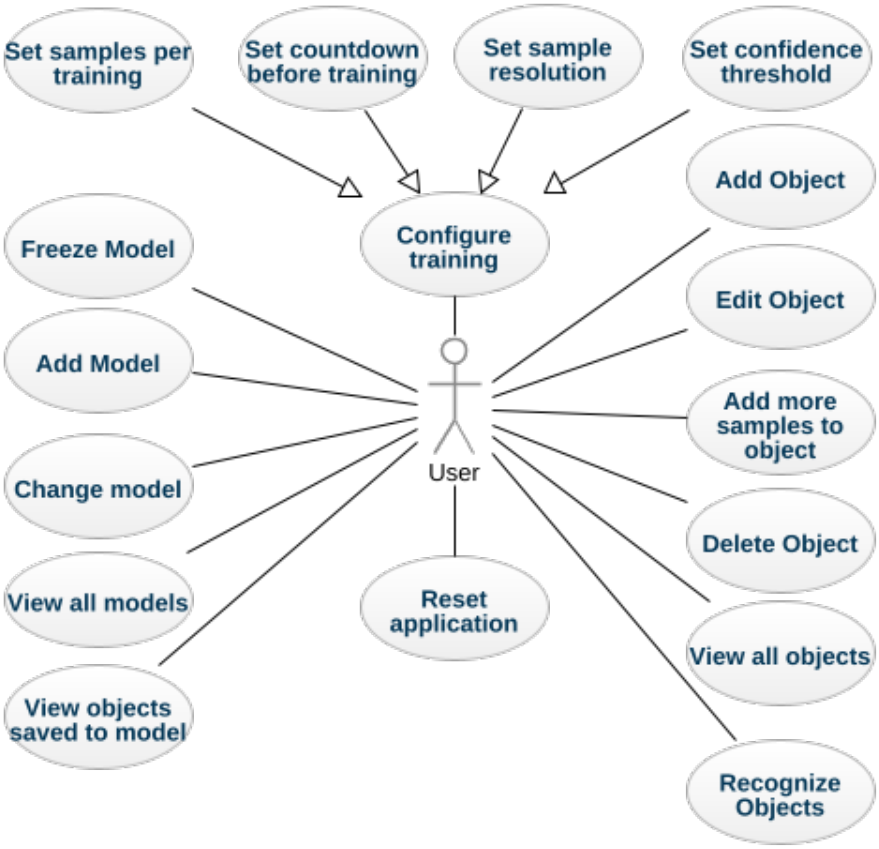

Abbildung 3.1.: UML-Anwendungsfall-Diagramm
Quelle: Eigene Darstellung

4. Entwurf

Auf der Basis der im vorangegangenen Kapitel erstellten Anforderungsanalyse und der im Grundlagenkapitel aufgearbeiteten theoretischen Kenntnisse wird ein Lösungskonzept erarbeitet.

4.1. Grundlegende Designentscheidungen

Die für den Entwurf der Applikation zugrundelegenden Entscheidungen, welche für die erfolgreiche Implentierung entscheidend sind, werden im Folgenden erklärt.

4.1.1. Tensorflow Lite und Continual Learning

Tensorflow Lite ist ein Open-Source Deep Learning Framework, welches für die Inferenz von Deep Learning Modellen auf Mobilen Endgeräten und eingebetteten Systemen entwickelt wurde [1].
Aufgrund einer bereits vorhandenen Beispielimplementierung einer Transfer Learning Applikation, wurde eine Basis für die ersten Ansätze einer Anbindung der Wiederkennungsfunktionalität unter der Nutzung von Transfer Learning für die Applikation, die in dieser Projektarbeit zu entwickeln ist, geschaffen. Tensorflow Lites stellt ein API zur Verfügung, welche die Nutzung von Transfer Learning vereinfacht. Zudem ist es durch ein Python-Skript möglich selbst generierte Tensorflow Lite-Modelle für das Transfer Learning vorzubereiten.
Um einen Latent-Replay Algorithmus in die Applikation zu übertragen, wurden Anpassungen übernommen nach dem Vorbild von Demosthenous et al. [2], welche im Rahmen ihrer Forschung den Transfer Learning Ansatz mit dem Continual Learning Ansatz von Pellegrini et al. [7] verglichen haben.

4.2. Entwurfsmuster

Um sich auf bereits bewährte Entwurfsansätze bestehender Software beziehen zu können, werden unter anderem mehrere Entwurfsmuster aus dem *GoF-Buch* genutzt. Dies ermöglicht es qualitativ hochwertige Software zu entwickeln, welche

durch das Anwenden von allgemein bekannten Mustern von anderen Entwicklern leichter nachvollzogen werden kann.

4.2.1. Ports-und-Adapter

Um ein möglichst wart- und erweiterbares Softwareprodukt zu entwickeln, wurde das *Ports-und-Adapter-Muster*, soweit dies möglich war, angewendet, um Third-Party-Dependencies, wie beispielsweise Tensorflow Lite, aber auch um die Persistenz möglichst abstrakt zu halten und eine Bindung an konkrete Implementierungen und Technologien zu vermeiden. Dies erhöht die Testfähigkeit enorm, da das *Dependency-Inversion-Prinzip* angewendet wird und die Abhängigkeiten der Geschäftslogik so gering wie möglich gehalten wird.

4.2.2. Factories

Um die Erzeugung von Objekten ähnlicher Klassen, wie beispielsweise Dialogboxen, möglichst konsistent zu halten, werden diese mithilfe des *Factory*-Musters erstellt, soweit dies möglich war.

4.2.3. Observer

Um eine starke Kopplung der Komponenten zu vermeiden, wird das *Observer*-Muster so weit dies möglich ist verwendet. Es findet vor allem im Aktualisieren der Graphischen Oberfläche, aber auch bei der Interkommunikation von entkoppelten Dialogfenstern und Fragments, sowie bei der Beobachtung von asynchronen Prozessen Anwendung.

4.2.4. Fragments

Da das Entwickeln einer Android-Applikation zu den Anforderungen gehört, ist zu entscheiden, ob man eine Applikation auf Basis mehrerer Android-Aktivitäten (*engl.: Activities*) erstellt, oder eine Activity mithilfe von angefügten sogenannten Fragmenten (*engl: Fragments*) entwickelt. Das Nutzen von Fragments ermöglicht die Interkommunikation der Systemkomponenten über die Activity, zu welcher diese gehören. Da in mehreren Anwendungsfällen das Zusammenarbeiten mehrerer Systemkomponenten notwendig ist, ist die Nutzung von Fragments der Nutzung von mehreren Aktivitäten vorzuziehen.
Die Oberfläche der Applikation kann mithilfe von XML-Code generiert werden. Programmatisch wird die Oberfläche an die Funktionalität, welche in Java-Code

geschrieben wurde, gebunden.

4.3. Entitätsklassen

Die für die Geschäftslogik grundlegenden und technisch unbedingt notwendigen Entitäten werden in Abbildung 4.1 dargestellt. In den folgenden Unterkapiteln werden diese erläutert.
Zu allen Entitäten wird eine einzigartige fortlaufende Identifikationsnummer (ID) gespeichert.

Abbildung 4.1.: Entity-Relationship-Diagramm
Quelle: Eigene Darstellung

4.3.1. Modell

Um dem Nutzer eine Unterscheidungsmöglichkeit zu bieten wird zu dem Modell ein Name abgespeichert.
Der absolute Pfad zu einer lokalen Binärdatei, die die derzeitige Parametrisierung des Modells festlegt wird zusätzlich abgespeichert, anstatt die Binärdaten direkt als Binary Large Object (Blob) zu speichern. Dies vereinfacht es die Applikation bei Bedarf um eine „Modell teilen“-Funktion zu erweitern.
Zusätzlich wird ein boolescher Wert zu jedem Modell abgespeichert, welcher entscheidet, ob das Modell bereits eingefroren wurde, um den Nutzer warnen zu können, wenn er im Begriff ist bereits bestehende Objekte aufgrund des Phänomen des Katastrophalen Vergessen zu überschreiben.

4.3.2. Objekt

Das Objekt ist die Kernentität der Applikation. Im Rahmen dieser Projektarbeit wurden einem Objekt diverse Attribute, wie beispielsweise dem Objekttyp oder dem generischen Attribut „Weitere Daten“ angefügt. Es ist jedoch möglich die zusätzlichen Daten, außer dem bezeichnenden Objektnamen, leer zu lassen, da diese keine Auswirkungen auf die Funktionsweise der Applikation haben.
Jedes Objekt hat eine Beziehung zu einem bestimmten Modell. Dies kann durch das Speichern einer Modell-Identifikationsnummer als Fremdschlüssel realisiert werden.
Ein Modell stellt eine Anzahl an Endknoten zur Verfügung, welche sich direkt auf die Endknoten der Netzarchitektur des zugrundeliegenden DCNN projizieren lassen. Demnach ist zu jedem Objekt eine Modelposition zu speichern. Anhand dieser Kartierung ist eine lose Kopplung zwischen Objekt und Modell und eine generische Erstellung von beliebig vielen Modellen und demnach beliebig vielen Objekten, anknüpfend an bestimmte Modelle, möglich.
Außerdem wird dem Objekt automatisch ein Zeitstempel hinzugefügt, um dem Nutzer darüber Auskunft geben zu können, wann das jeweilige Objekt eingespeichert wurde. Zudem ist ein Zeitstempel für mögliche weitere Datenabfragen und beispielsweise Erweiterungen der Objektübersicht mit einer Nach-Datum-Sortierungsfunktion sinnvoll.
Sobald ein Objekttraining gestartet wird, wird die erste Bitmap, die in der Trainings-Vorbereitungs-Pipeline eingeht persistent abgespeichert, um dem Nutzer eine Vorschau des Objektes zu geben, falls es in Vergessenheit geraten ist, um welches Objekt es sich handelt.
Die Speicherung der Anzahl der Trainingsdaten pro Objekt sind von monitorischer Bedeutung. Durch das Speichern und die spätere Anzeige dieser Daten kann der Nutzer entscheiden, ob es sinnvoll ist einem Objekt weitere Trainingsdaten hinzuzufügen. Mögliche Erkennungsprobleme wie beispielsweise einer ungleichen

Verteilung an Trainingsdaten können dadurch leichter festgestellt werden.

4.3.3. Trainingsdaten

Die Aktivierungen im Bottleneck-Layer des zugrundeliegenden Mobilenet v2, werden persistent abgespeichert, mitsamt der jeweiligen Modellposition und der Modell-Identifikationsnummer, zu welchem die Trainingsdaten gehören. Erforderlich ist dies aufgrund der Implementierung eines Continual Learning Ansatzes unter Verwendung des Latent-Replay Algorithmus, da alte Trainingsdaten bei einem Objekttraining vermischt werden, um das Phänomen des Katastrophalen Vergessens zu lösen.
Ein Zeitstempel wird zu jedem Trainingsdatenfeld hinzugefügt, um lediglich die letzten n Daten in den Latent-Replay-Puffer zu überführen.

4.3.4. Latent-Replay-Puffer

Der Latent-Replay-Puffer wird nach jeder Trainingssitzung aktualisiert mit den zuletzt hinzugefügten Trainingsdaten.
Zudem wird die Modellidentifikationsnummer als Fremdschlüssel und die Modellposition, zu welchem das jeweilige Trainingsdatenfeld gehört mitabgespeichert, um das Vermischen von neuen Trainingsdaten und alten Trainingsdaten, nach dem Prinzip des Latent-Replay-Algoritmhus zu ermöglichen.

4.4. Architektur

Zur Realisierung der Applikation auf einem Android-Endgerät, werden die Komponenten der zugrundelegenden Architektur aus Activities und Fragments bestehen. Eine Activity-Komponente wird für den Startbildschirm erstellt, welche die Haupt-Activity für die Funktionalität der App einleitet. Mithilfe von Fragments, die an diese Activity anknüpfen kann die Kommunikation und der Datenaustausch zwischen verschiedenen Fragments stattfinden.
Abbildung 4.2 zeigt die Abhängigkeiten der im Folgenden erklärten verwendeten Komponenten.
Im Folgenden werden die Verantwortlichkeiten der zu implementierenden Fragments erklärt und die Anwendungsfälle aus Anhang A werden zugewiesen.

- *Model Overview Fragment*: Dient der Übersicht über alle Modelle, sowie der Möglichkeit, die Objekte die im jeweiligen Modell gespeichert sind anzuschauen. Zudem kann in diesem Fragment ein weiteres Modell hinzugefügt werden,

oder das aktuell ausgewählte Modell mit einem ebenfalls bereits bestehenden Modell ausgewechselt werden. Demnach werden die Anwendungsfälle M1-M4 von dieser Komponente erfüllt.

- *Object Overview Fragment*: Dient der Übersicht über alle Objekte und deren gespeicherten Informationen, sowie der Möglichkeit Objekte zu editieren und zu löschen. Zudem soll dem Fragment die Verantwortlichkeit hinzugefügt werden, dem jeweiligen Objekt weitere Trainingsdaten hinzuzufügen. Demnach werden die Anwendungsfälle O3-O6 von dieser Komponente erfüllt.

- *Settings Fragment*: Konfigurationen, die der Nutzer selbst zur Laufzeit vornehmen kann, werden innerhalb dieses Fragments in einer typischen Einstellungen-Ansicht realisiert. Demnach werden die Anwendungsfälle E1-E5 von dieser Komponente erfüllt. Weitere Konfigurationsmöglichkeiten zur Verfügung zu stellen, wie zum Beispiel das Aktivieren eines Dunkelmodus ist ebenfalls geplant.

- *Camera Fragment*: Stellt die Hauptfunktionalität des Einlernens und Wiedererkennens zur Verfügung. Eine Kameravorschau, die mit einer Fokusbox und anderen Overlays, die zum Anzeigen von Informationen dienen, ausgestattet ist, soll in Echtzeit aktualisiert werden. Durch eine Dialogbox wird zum Einlernen eines Objektes eine Dialogbox aufgerufen, um diesem optional Informationen hinzuzufügen. Innerhalb dieses Fensters wird der Trainingsvorgang initiiert. Demnach werden die Anwendungsfälle O1 und O2 von dieser Komponente erfüllt.

- *Help Fragment*: Falls bei der Bedienung der Applikation Schwierigkeiten auftreten sollten, stellt das Help Fragment diverse Anleitungen und Nutzungsabläufe textuell und gegebenfalls durch Videoanleitungen dar.

Durch Anwendung dieser Architektur, wird die sogenannte *separation of concerns* gewährleistet und das *Single-Responsibility-Principle* angewendet.
Weitere Komponenten sind der Datenbankhelfer, welcher eine zu implementierende Schnittstelle an die Applikation zur Realisierung von Persistenz-Operationen beinhaltet. Implementiert werden, kann diese Schnittstelle beispielsweise von einem SQLite-Datenbankhelfer.
Zur globalen Konfiguration verschiedener Parameter, welche auch nach Neustart der App vorhanden sein sollen, wird ebenfalls eine Schnittstelle zur Verfügung gestellt, welche beispielweise durch Verwendung der *Shared Preferences* implementiert werden kann.
Um die Übersichtlichkeit und der darausfolgenden besseren Wart- und Testbarkeit der Applikation zu gewährleisten, werden Dialogboxen ausgelagert und mithilfe einer Schnittstelle und einer *Factory-Method* erzeugt. Dialogboxen werden zur Interaktion mit dem Nutzer häufig verwendet. Das Hinzufügen oder Ändern von Daten, sowie das Anzeigen von längeren Hinweisen wird über Dialogboxen an den Nutzer

kommuniziert. Bei gefährlichen Aktionen, wie beispielsweise dem Löschen von Objekten oder dem Zurücksetzen der Applikation wird der Nutzer ebenfalls durch eine Dialogbox auf die möglichen Auswirkungen der Aktion hingewiesen.
Statische Methoden, die dem Parsen von Zeichenketten, oder dem Vorbereiten der Kameradaten an die Transfer Learning API dienen, werden in eine *Utilities*-Komponente ausgelagert, um den Code, der die Kernfunktionalität realisiert möglichst übersichtlich und kompakt zu halten.
Die Anbindung der *Transfer Learning API* erfolgt in Form einer Wrapper-Klasse, welche die API um weitere Verantwortlichkeiten erweitert, wie beispielsweise dem Kontinuierlichen Trainieren und Wiedererkennen innerhalb einer While-Schleife und dem Ausführen des Latent-Replay Algorithmus. Aufgrund der Ausführung des Algorithmus muss eine Abhängigkeit zur Schnittstelle des Datenbankhelfers erfolgen, da die persistente Abspeicherung von Aktivierungen früherer Trainingsdaten unabdingbar für den Replay-Vorgang ist. Zudem übernimmt der Wrapper die Erstellung des Modells und koordiniert die API an die Parameter-Datei des ausgewählten Modells.

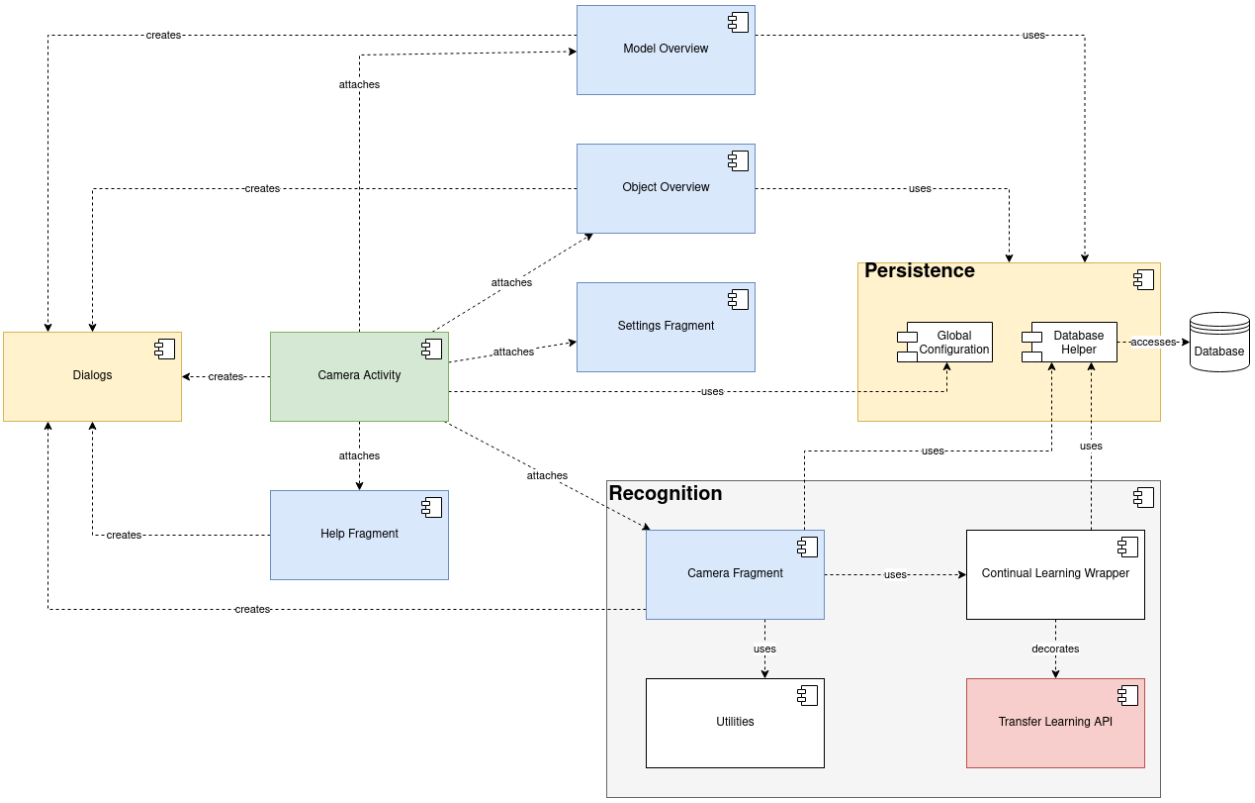

Abbildung 4.2.: UML-Komponentendiagramm
Quelle: Eigene Darstellung

4.5. Feinentwurf

Im Folgenden werden die Strukturen und Abläufe, der in Kapitel 4.4 beschriebenen Komponenten erklärt.
Um eine Übersichtlichkeit der Klassendiagramme zu gewährleisten, wurden nicht alle Methoden und Komponenten aufgeführt. Die Lücken werden mit der Zeichenfolge „ ... “ gekennzeichnet.

4.5.1. Klassendiagramme

Im Folgenden wird die statische Struktur der Komponenten auf Klassenebene erklärt. Um eine Übersichtlichkeit des Entwurfes gewährleisten zu können, werden Komponenten nach ihren Verantwortlichkeiten und erfüllten Anwendungsfällen getrennt beschrieben.

Persistenz-Komponente

Die Persistenz des Objektes wird mithilfe von Schnittstellen angebunden und lässt sich in eine Komponente für Globale Konfigurationsparameter und eine Komponente für die Anbindung einer Datenbank aufteilen. Die Datebankhelfer-Schnittstelle stellt Methoden zur Abfrage, Einfügen und Löschen der strukturiert gespeicherten Entitätsklassen zur Verfügung. In der Schnittstelle für die globalen Konfigurationsparameter, werden Methoden zur Verfügung gestellt, die primitive Daten, die persistent für Abläufe der Applikation gespeichert werden müssen, abgefragt und gespeichert.
Abbildung 4.3 zeigt den Aufbau der Komponenten auf Klassenebene und deren Beziehungen zu anderen Komponenten. Durch die Verwendung von Schnittstellen wurde das Dependency Inversion Principle angewendet. Bei Bedarf ist es dadurch möglich die SQLite Datenbank durch eine andere auszutauschen unter der Bedingung, dass alle Methoden der Schnittstelle implementiert werden.

Kamera Fragment

Das Kamerafragment enthält Methoden zur Ansteuerung des Continual Learning Modell Wrappers und zur Konfiguration und Vorbereitung der Kameradaten. Zudem werden die in der Datenbank gespeicherten Objekte an die Endknoten des Modells angeheftet, um bei einer Wiedererkennung die Daten des Objektes wiederanzeigen lassen zu können.
Aufgrund des Einsetzens des MVVM-Patterns wird ein Viewmodel für das Fragment aggregiert, welches unter Verwendung von Livedaten die Graphische Oberfläche

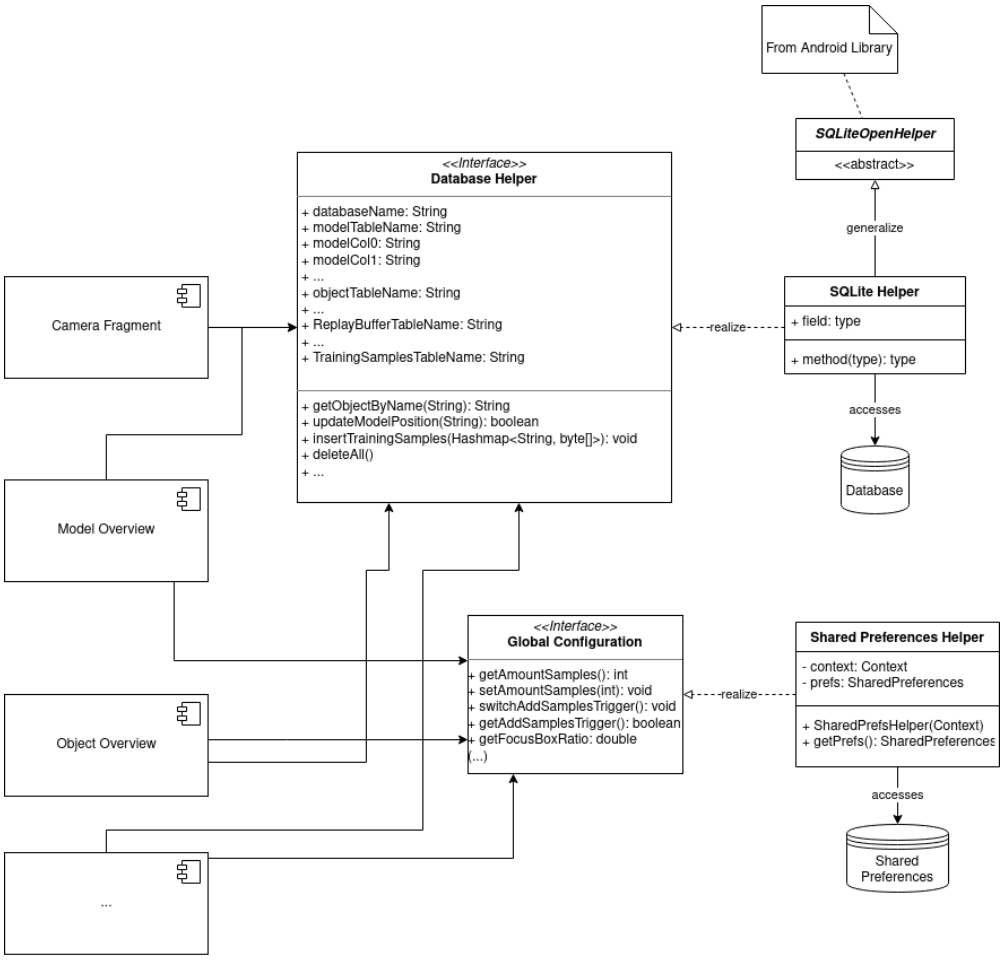

Abbildung 4.3.: UML-Klassendiagramm der Persistenz Komponente
Quelle: Eigene Darstellung

kontinuierlich aktualisiert. Die Livedaten werden hierbei innerhalb eines anonymen Observers, welcher in der von Fragment geerbten „*onViewCreated*“-Methode erzeugt werden.
Um den Wertebereich des Trainingsstatus zu optimieren, wird eine Enumeration mit den Status „Noch nicht gestartet “, „Gestartet “ und „pausiert“ eingebunden.
Um die Daten in Echtzeit und eine möglichst nicht blockierende Applikation zu schreiben, wird eine Unterklasse zum Hinzufügen von Trainingsdatenanfragen als Thread ausgeführt, welche zudem einen Countdown vor dem Starten der Methode zum Trainingsdaten hinzufügen ausführt.
Der Continual Learning Model Wrapper erweitert die von Tensorflow Lite zur Verfügung gestellte Transfer Learning API, um den Latent-Replay-Algorithmus und Methoden um die Modellparameter in einer separaten Datei abzuspeichern und zu laden.
Die Transfer Learning API stellt die wichtigsten Methoden für das Training, die Inferenz und dem Speichern und Laden von Parameterdaten zur Verfügung. // auch auf Funktionsweise und Bereitstellung der Methoden der TFLITE API eingehen

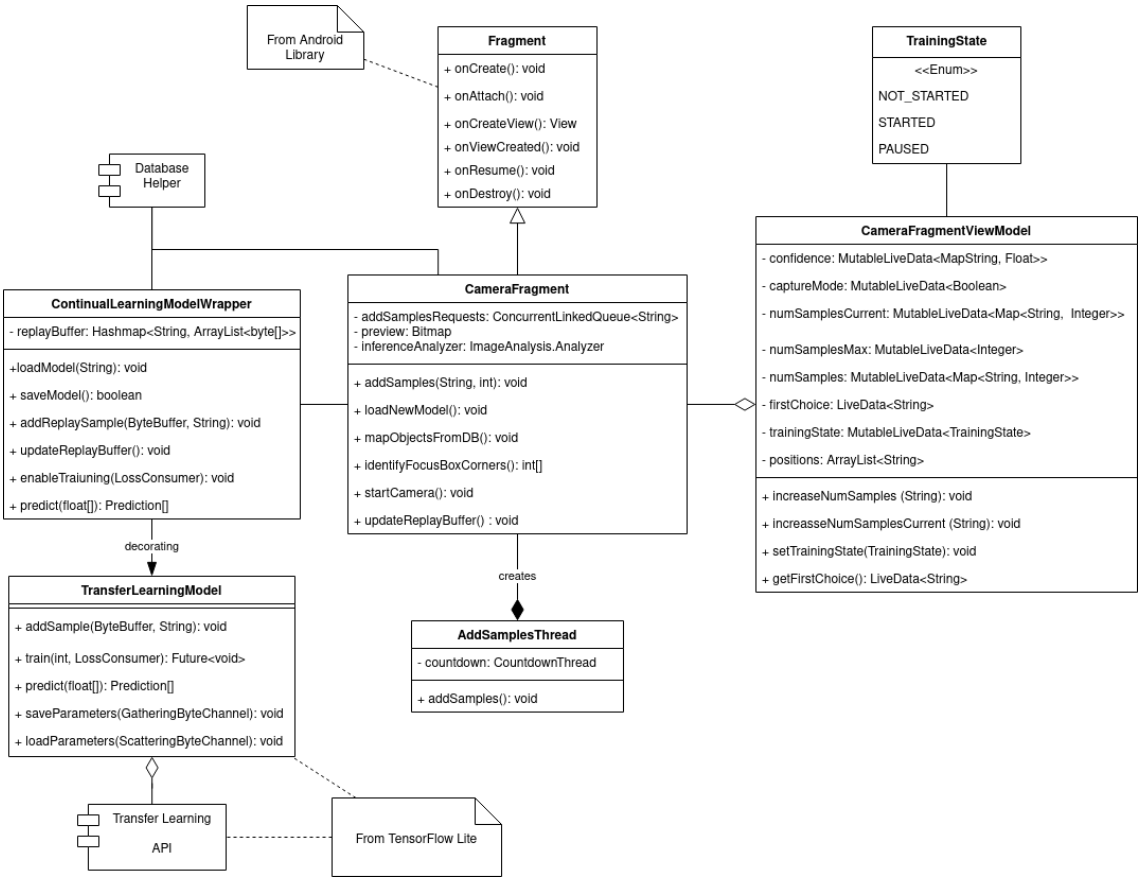

Abbildung 4.4.: UML-Klassendiagramm der Wiedererkennungs-Komponente
Quelle: Eigene Darstellung

Modell- und Objektübersicht

Da die Modell- und Objektübersicht analog aufgebaut sind, wird die Beschreibung der Komponenten auf Klassenebene in einem einzigen Unterkapitel vereint.
Die Verbindung zur Persistenzkomponente ermöglicht es die Listeninstanzen mit Daten aus der Datenbank zu füllen. Zur Erstellung einer angepassten Listenansicht wurde der Listenadapter von Android erweitert. Dieser Listenadapter bietet die Möglichkeit eine Listenansicht zu Erstellen, anhand der Position der zu identifizierenden Elemente. Dadurch können Listelement-spezifische Daten an die Methoden übergeben werden. Ein Beispiel hierfür ist das Einfügen von Click-Listenern auf Buttons, welche anhang dieser Position angesteuert werden. Das Dialog-Paket wird ebenfalls genutzt und fördert somit die Wiederverwendbarkeit der Komponenten. Abbildung 4.5 zeigt die statische Struktur der Modell- und Objektübersicht auf Klassenebene.

4.5.2. Kommunikation der Komponenten

Die *Camera Activity* ist die Hauptkomponente und bietet die Möglichkeit Informationen und Benachrichtigungen, welche zwischen den Fragmenten erfolgen sollen, auszutauschen. Eine Navigationsleiste am unteren Bildschirmrand und ein aufklappbares Seitenmenü bietet dem Nutzer die Möglichkeit, die angezeigten Fragmente zu wechseln und deren Funktionalität zu nutzen.
Die Activity beobachtet bestimmte Aktionen innerhalb der Fragmente das Registrieren als *Observer* bei den Globalen Konfigurationsparametern und koordiniert Befehlsaufrufe an das jeweils benötigte Fragment. Die Komponente zur Globalen Konfiguration wird hierfür von Fragmenten, die interagieren müssen, instanziiert. Abbildung 4.6 stellt diesen Ablauf in Form eines Sequenzdiagramm, am Beispiel des Änderns des ausgewählten Modells graphisch dar.

Durch Nutzung dieses Musters ist eine lose Kopplung der Komponenten möglich und Fragmente können ohne voneinander abhängig zu sein ausgeführt werden. Analog zu diesem Beispiel werden die folgenden Abläufe realisiert:

- Objekttraining auslösen
- Objekttraining Rollback auslösen im Fehlerfall
- Aktualisieren des Replay-Puffers
- Initialisieren der Namen der Ausgabeknoten des Modells (standardmäßig werden die Ausgabeknoten mit aufsteigenden Nummern als Zeichenkette initialisiert)
- Auflösung der Trainingsdaten anpassen

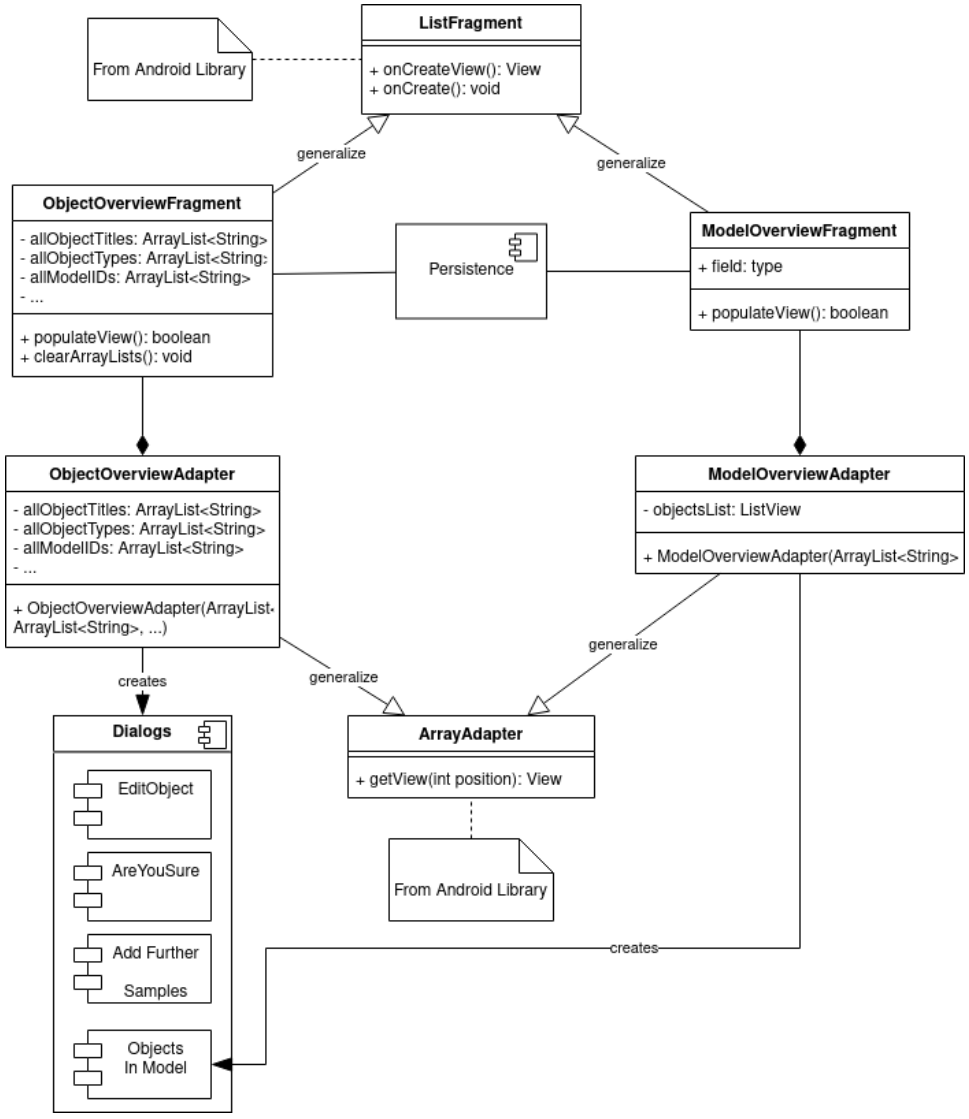

Abbildung 4.5.: UML-Klassendiagramm der Objekt- und Modellübersicht
Quelle: Eigene Darstellung

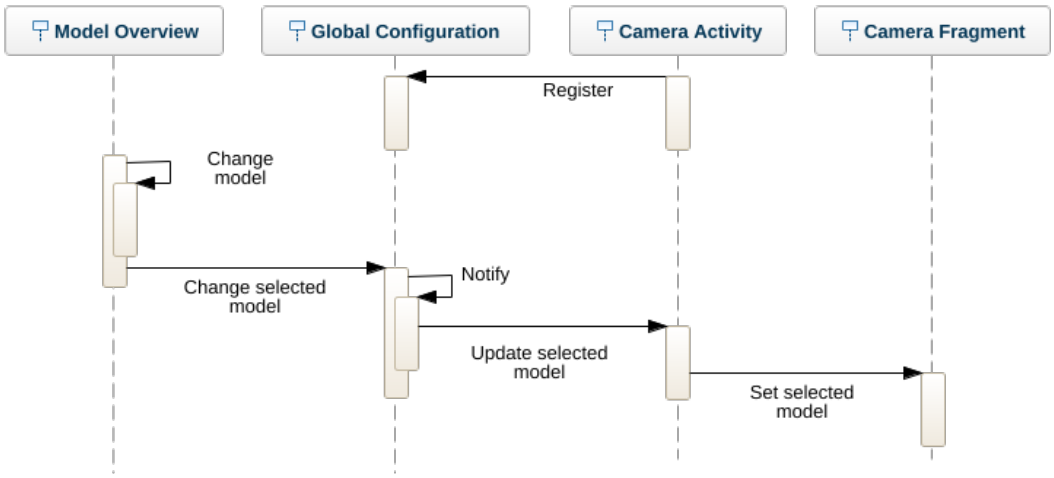

Abbildung 4.6.: UML-Sequenzdiagramm: Ändern des ausgewählten Modells
Quelle: Eigene Darstellung

- Konfidenzschwelle anpassen
- Länge des Countdowns vor dem Training anpassen
- Aktivieren des Dunkelmodus

4.5.3. Vorbereitung und Vorschau der Kameradaten

Das Camera Fragment beinhaltet eine Vorschau in Echtzeit, welche dem Nutzer ermöglicht zu sehen, was die im Endgerät eingebaute Kamera sieht.
Zur Realisierung dieser Funktionalität müssen diverse Konfigurationen vorgenommen werden. Hierzu gehört das Initialisieren der Ausrichtung des Gerätes (Portrait- oder Landscape-Modus), Das Seitenverhältnis, die Auflösung der Kamera und weiteren systembezogenen Einstellungen wie beispielsweise dem automatischen Fokussieren. Zudem wird ein Bildanalyse-Thread zur Vorkonfiguration der Kameraansicht hinzugefügt. Die Inferenz- und Trainingsfunktionalität wird direkt hier eingesetzt, da dies ermöglicht jeden angezeigten Frame asynchron zum Programmablauf zu akquirieren und zu verarbeiten.
Nach Konfiguration der soeben beschriebenen Parameter, wird die Kamera an den Lebenszyklus des Fragments gebunden und demnach bei Öffnen des Fragments gestartet und Schließen beziehungsweise Wechseln geschlossen.

4.5.4. Einlernen von Objekten

Das Einlernen und Wiedererkennen von Objekten sind die umfangreichsten und komplexesten Anwendungsfälle in der Applikation, welche diverse Komponenten involviert und Interprozesskommunikation von nebenläufigen Prozessen erfordert. Die Struktur und der Ablauf des in Anwendungsfall A.6 beschriebenen sichtbaren Verhaltens, wird im Folgenden präziser erklärt.
Nach Betätigung des „+“-Buttons auf der graphischen Oberfläche wird ein Dialogfenster erzeugt. Nach Eingabe der Informationen zum Objekt, löst der „Training starten“-Button im Dialogfenster unter Anwendung des in Kapitel 4.5.2 beschriebenen Ablauf das Objekttraining aus. Parallel dazu werden die eingegebenen Daten des Objektes in der Datenbank gespeichert. Die Activity delegiert den Aufruf an das Camera Fragment, welches die Verantwortlichkeiten für das Einlernen trägt.
Eine Warteschlange bestehend aus Modellpositionen wird hierbei innerhalb eines neuen Threads mit der vorkonfigurierten Anzahl an Trainingsdaten pro Objekttraining gefüllt.
Diese Warteschlange wird kontinuierlich vom Analysierer abgefragt und entscheided dadurch, ob Objekte trainiert, oder lediglich Inferenz ausgeführt werden soll.
Der Analysierer bereitet die stets zugeführten Bitmaps aus der Kamera für das Training vor. Hierbei wird die Bitmap in ein Float-Datenfeld konvertiert, da dies die Datenstruktur ist, mit welchem das zugrundeliegende Deep-Learning-Modell Eingangsbilder prozessiert.
Wenn sich in der Warteschlange Trainingsanfragen befinden, wird die eingehende Bitmap zum Objekt dazugespeichert, damit es bei der Wiedererkennung und in der Objektübersicht als Objektvorschau angezeigt werden kann. Das nun vorbereitete Trainingsdatenfeld wird in den Continual Learning Model Wrapper gegeben und zugleich zu den Trainingsdaten in der Datenbank abgespeichert.
Der Wrapper leitet das Training an die Transfer Learning API weiter, welche das Modell nun für die übergebene Modellposition trainiert. Dieser Trainingsprozess findet kontinuierlich statt, durch das starten eines Trainingthreads im Wrapper.
Parallel dazu wird der Trainingsfortschritt prozentual ermittelt und im View Model aktualisiert. Ein Observer aktualisiert nun während des Trainings einen Ladekreis auf der graphischen Oberfläche.
Falls ein Fehlerfall eintritt, wie beispielsweise dem frühzeitigen Wechseln des Fragments, werden die Änderungen in der Datenbank zurückgesetzt und somit das noch nicht fertig trainierte Objekte wieder gelöscht. Zudem wird der Nutzer über seinen Fehler durch eine Meldung informiert.

4.5.5. Wiedererkennen von Objekten

Das Wiedererkennen von Objekten findet kontinuierlich statt, wenn die Anfragewarteschlange leer ist. Hierfür wird eine Methode Im Modell-Wrapper, welche direkt die Inferenzmethode der Transfer Learning API aufruft, ausgeführt. Die Ergebnisse aktualisieren im angebundenen View Model das erkannte Objekt und die Konfidenz dazu. Mithilfe eines Observers wird die graphische Oberfläche kontinuierlich aktualisiert.
Die eben erklärten Abläufe zum Einlernen und Wiedererkennen von Objekten, werden in Abbildung 4.7 als Aktivitätsdiagramm graphisch dargestellt.

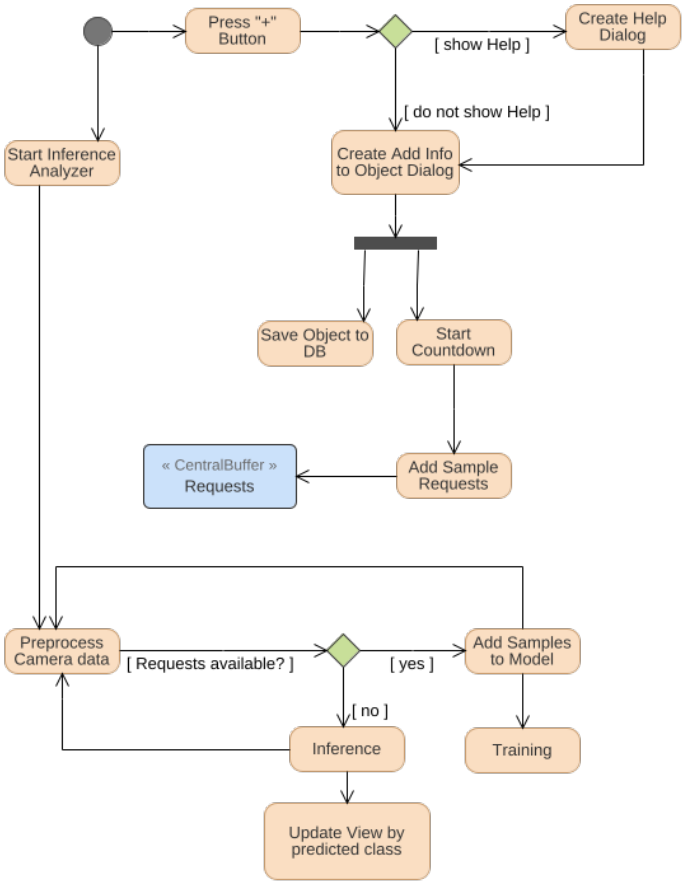

Abbildung 4.7.: UML-Aktivitätsdiagramm: Einlernen und Wiedererkennen von Objekten

Quelle: Eigene Darstellung

4.5.6. Speichern und Laden von Parameterdateien

Eine Parameterdatei bezeichnet die Parametrisierung des Modells, welche binär in einer Datei abgespeichert wird. Um Mehrdeutigkeit zu vermeiden wird die Bezeichnung *model-parameters* mit einer möglichst zufällig generierten Zeichenfolge und dem Suffix *.bin* konkateniert und eine leere Datei wird im Verzeichnis der Applikation erzeugt.
Die Transfer Learning API stellt Methoden zum Speichern und Laden von Modell-Parametern zur Verfügung. Durch das Erzeugen eines Ein-/Ausgabestromes können die Dateien unter Angabe des absoluten Pfades der eben generierten leeren Datei zur Laufzeit der Applikation geladen beziehungsweise mit Binärdaten gefüllt werden.

4.5.7. Anbindung des Latent-Replay Algorithmus

Zu Beginn der Entwicklung der Applikation wurde lediglich auf Basis des Model-Personalization-Beispiels, welches von Tensorflow Lite zur Verfügung gestellt wird, die Funktionalität der Wiedererkennung von Objekten realisiert. Im Laufe der Testphase, wurde jedoch klar, dass das Phänomen des Katastrophalen Vergessens fatal für eine wiederverwendbare Applikation ist. Transfer Learning funktioniert, solange das Modell nicht geschlossen beziehungsweise neu gestartet wird. Bereits das Wechseln der Fragmente zur Laufzeit verursacht das Schließen des Transfer Learning Modells. Dies hat zur Folge, dass die Trainingsdaten aus der letzten Sitzung nicht mehr für das Modell zur Verfügung stehen.
Bereits in der Anfangsphase dieser Projektarbeit, wurde der Latent-Replay Algorithmus von Pellegrini et al. studiert, welcher eine Lösung zum Phänomen des Katastrophalen Vergessens vorstellt [7]. Aufgrund keiner vorhandenen Beispielimplementierung wurde dieser Ansatz jedoch verworfen.
Im Laufe der Implementierungsphase der Applikation dieser Projektarbeit publizierten Demosthenous et al. ein Paper mit zugehörigem offenen Quellcode, welcher die Einbindung des Latent-Replay Algorithmus unter Verwendung der Transfer Learning API massiv erleichtern konnte [2]. Auf Basis dieser Beispielimplementierung wurde der Algorithmus eingebunden. Lediglich wenige Änderungen waren hierfür notwendig.
Sobald der Analysierer Trainingsanfragen erhält, werden während dem Hinzufügen der Trainingsdaten zum zugrundeliegenden Modell, die Daten persistent in der Datenbank abgespeichert, um diese später beim Ausführen des Latent-Replay erneut einzulernen.
Methoden, in welchen die neuen Trainingsdaten an das Modell weitergegeben werden, werden durch eine Replay-Methode erweitert. Daraufhin werden die Trainingsdaten zusammen mit bereits vorhandenen Trainingsdaten zusammen trainiert und das Phänomen des Katastrophalen Vergessens kann gelöst werden.

4.5.8. Belegen der Modellpositionen mit Objekten

Das Anknüpfen der Objekte an die zur Verfügung stehenden Modellpositionen wird im Folgenden erklärt.
Jedes Deep Learning Modell hat am am Ende der Netzarchitektur eine vordefinierte Anzahl an Ausgabeknoten, welche für die endgültige Klassifizierung nach Prozessieren der Eingabedaten einen Wert zwischen 0 und 1 annimmt. Diese ausgegebene Zahl kann als Konfidenzwert, das heißt die Sicherheit mit welcher das Modell die Eingabedaten diesem Ausgabeknoten zuordnen kann, interpretiert werden.
Durch eine festdefinierte Anzahl an Ausgabeknoten, ergibt sich eine Einschränkung der gleichzeitig wiedererkennbaren Objekte, welches durch das Einführen von Objektordnern gelöst wird. Die Bezeichnung Objektordner ist gleichbedeutend mit der auch im Quellcode vorkommenden Bezeichnung Model und wurde gewählt, um den Nutzer mit technischen ML-spezifischen Begriffen nicht zu überfordern.
Es ist demnach möglich Objekte in ein Objektordner in Höhe der Anzahl an verfügbaren Endknoten des Modells zu speichern. In der Modellübersicht, beziehungsweise der Ordnerübersicht, lässt sich die Parametrisierung des Modells durch Selektieren eines anderen bereits bestehenden Modells wechseln.
Bereits gespeicherte Objekte enthalten immer eine Referenz auf das Modell, welches die Bilddaten des Objektes trainiert hat.
Das Camera Fragment View Model hält eine Liste an verfügbaren Modellpositionen. Bei Erzeugung des Fragments wird diese Liste mit der Anzahl an Objekten, die das ausgewählte Modell referenzieren, abgeglichen. Sobald ein Objekt zu diesem Modell gehört und eine Modellposition zugewiesen bekommen hat, wird die entsprechende Position aus der Liste des Viewmodels entfernt.
Beim Hinzufügen von neuen Objekten wird die erste Position, die in der im View Model gespeicherten Liste vorkommt, zum Objekt hinzugefügt und persistent mit diesem abgespeichert. Falls die Liste leer ist, wird der Nutzer darüber informiert, dass der Objektordner, beziehungsweise das Modell, voll ist.

4.5.9. Konfigurierbarkeit mit Einstellungsfragment

5. Implementierung

Transfer Learning, quantisiertes MobilenetV2, Imagenet.

Ein Problem das beim Nutzen des Transfer Learnings aufgetreten ist, ist dass alle Trainingsdaten sich im derzeitigen Trainingsbatch befinden müssen, da sonst das Problem des Katastrophalen Vergessens aufkommt. In diesem Kapitel wird die konkrete Implementierung des im Kapitel **??** entwickelten Lösungskonzepts beschrieben. Hierbei wird auf die konkret verwendeten Entwicklungswerkzeuge etc. Bezug genommen.

Bei Software-Projekten besteht dieses Kapitel typischerweise aus den Phasen Implementierung & Test im **rup!** (**rup!**).

Zum Beispiel kann man hier auch ein kleines Listing einfügen.

```
1  #include<stdio.h>
2
3  int main() {
4      // Kommentar
5      int answer = 20 << 1;
6      answer += 2;
7      printf("Hallöchen Welt!\n");
8      printf("Die Antwort ist: %d\n", answer);
9      return 0;
10 }
```

Quelltext 5.1: Überschrift des Quelltexts

Manchmal hilft auch eine kleine Tabelle:

Details siehe Tabelle **??**.

6. Tests

Aufgabe des Kapitels Inbetriebnahme ist es, die Überführung der in Kapitel 5 entwickelte Lösung in das betriebliche Umfeld aufzuzeigen. Dabei wird beispielsweise die Inbetriebnahme eines Programms beschrieben oder die Integration eines erstellten Programmodules dargestellt.

Bei der Software-Erstellung entspricht dieses Kapitel der Auslieferungsphase (Deployment) im **rup!**.

7. Evaluation

Aufgabe des Kapitels Evaluierung ist es, in wie weit die Ziele der Arbeit erreicht wurden. Es sollen also die erreichten Arbeitsergebnisse mit den Zielen verglichen werden. Ergebnis der Evaluierung kann auch sein, das bestimmte Ziele nicht erreicht werden konnten, wobei die Ursachen hierfür auch außerhalb des Verantwortungsbereichs des Praktikanten liegen können.

8. Zusammenfassung und Ausblick

8.1. Erreichte Ergebnisse

Die Zusammenfassung dient dazu, die wesentlichen Ergebnisse des Praktikums und vor allem die entwickelte Problemlösung und den erreichten Fortschritt darzustellen. (Sie haben Ihr Ziel erreicht und dies nachgewiesen).

8.2. Ausblick

Im Ausblick werden Ideen für die Weiterentwicklung der erstellten Lösung aufgezeigt. Der Ausblick sollte daher zeigen, dass die Ergebnisse der Arbeit nicht nur für die in der Arbeit identifizierten Problemstellungen verwendbar sind, sondern darüber hinaus erweitert sowie auf andere Probleme übertragen werden können.

8.2.1. Erweiterbarkeit der Ergebnisse

Hier kann man was über die Erweiterbarkeit der Ergebnisse sagen.

8.2.2. Übertragbarkeit der Ergebnisse

Und hier etwas über deren Übertragbarkeit.

Literatur

[1] Tensorflow Lite Contributors. *Deploy machine learning models on mobile and IoT devices*. 2021. URL: https://www.tensorflow.org/lite (besucht am 31. 07. 2021).

[2] Giorgos Demosthenous und Vassilis Vassiliades. „Continual Learning on the Edge with TensorFlow Lite“. In: (5. Mai 2021). arXiv: 2105.01946v1 [cs.LG].

[3] Andrew G. Howard u. a. „MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications“. In: (17. Apr. 2017). arXiv: 1704.04861v1 [cs.CV].

[4] James Kirkpatrick u. a. „Overcoming catastrophic forgetting in neural networks“. In: (2. Dez. 2016). arXiv: 1612.00796v2 [cs.LG].

[5] Yann LeCun, Yoshua Bengio und Geoffrey Hinton. „Deep learning“. In: (Mai 2015).

[6] Niall O' Mahony u. a. „Deep Learning vs. Traditional Computer Vision“. In: *in Advances in Computer Vision Proceedings of the 2019 Computer Vision Conference (CVC). Springer Nature Switzerland AG, pp. 128-144* (30. Okt. 2019). DOI: 10.1007/978-3-030-17795-9. arXiv: 1910.13796v1 [cs.CV].

[7] Lorenzo Pellegrini u. a. „Latent Replay for Real-Time Continual Learning“. In: (2. Dez. 2019). arXiv: 1912.01100 [cs.LG].

[8] Dipanjan Sarkar. *A Comprehensive Hands-on Guide to Transfer Learning with Real-World Applications in Deep Learning*. 14. Nov. 2019. URL: https://towardsdatascience.com/a-comprehensive-hands-on-guide-to-transfer-learning-with-real-world-applications-in-deep-learning-212bf3b2f27a (besucht am 12. 05. 2021).

[9] Pavel Senchanka. *Example on-device model personalization with TensorFlow Lite*. 12. Dez. 2019. URL: https://blog.tensorflow.org/2019/12/example-on-device-model-personalization.html (besucht am 12. 05. 2021).

A. Beschreibung der Anwendungsfälle

Nummer	**AF-M1**
Anwendungsfall	**Modell hinzufügen**
Beschreibung	Beliebig viele neue Modelle sollen hinzufügt werden können
Akteur	Nutzer
Auslöser	Der Nutzer betätigt den "+Button in der Modellübersicht oder startet die Applikation zum ersten Mal
Vorbedingung	Der Nutzer befindete sich in der Modellübersicht oder hat bisher noch kein Modell hinzugefügt
Nachbedingung/Ziel	Ein Modell zu welchem weitere Objekte hinzugefügt werden können, wurde gespeichert
Nachbedingung im Sonderfall	Das Modell wurde nicht gespeichert und der Nutzer wird darüber durch eine Fehlermeldung informiert
Standardablauf	1. Dialogfenster öffnet sich 2. Nutzer gibt Namen für Modell ein 3. Neues Modell wird gespeichert und Nutzer wird darüber informiert
Sonderfälle	2a. Ein Modell mit dem Namen existiert bereits → Der Nutzer wird darüber mit einer Fehlermeldung informiert 3a. Das Modell konnte nicht gespeichert werden → Der Nutzer wird darüber mit einer Fehlermeldung informiert

Tabelle A.1.: Anwendungsfall M1: Modell hinzufügen

Nummer	**AF-M2**
Anwendungsfall	**Modell wechseln**
Beschreibung	Das ausgewählte Modell soll gewechselt werden können mit einem anderen bestehenden Modell
Akteur	Nutzer
Auslöser	Der Nutzer befindet sich in der Modellübersicht und betätigt den Radiobutton, der sich links neben dem Modellnamen befindet
Vorbedingung	Es sind bereits mehrere Modelle hinzugefügt worden
Nachbedingung/Ziel	Es wurde ein anderes Modell ausgewählt
Nachbedingung im Sonderfall	Das Modell wurde nicht gewechselt
Standardablauf	1. Nutzer betätigt den Radiobutton neben dem Namen des Modells 2. Der Radiobutton wird als selektiert markiert und das Modell wurde gewechselt
Sonderfälle	2a. Das Modell konnte nichte selektiert werden → Keine Sonderbehandlung dieses Falles notwendig

Tabelle A.2.: Anwendungsfall M2: Modell wechseln

Nummer	**AF-M3**
Anwendungsfall	**Gespeicherte Objekte im Modell betrachten**
Beschreibung	Alle im Modell abgespeicherten Objekte sollen besichtigt werden können
Akteur	Nutzer
Auslöser	Der Nutzer betätigt den Button welcher sich rechts neben dem Modellnamen befindet
Vorbedingung	Der Nutzer befindet sich in der Modellübersicht und ein Modell wurde bereits abgespeichert
Nachbedingung/Ziel	Ein Dialogfenster, welches zum Modell alle gespeicherten Objekte anzeigt wurde geöffnet
Nachbedingung im Sonderfall	Es wurden noch keine Objekte für das Modell abgespeichert und ein Hilfetext befindet sich im geöffneten Dialog
Standardablauf	1. Nutzer betätigt den Button, welcher sich rechts neben dem Modell befindet, dessen Objekte betrachtet werden sollen 2. Ein Dialogfenster öffnet sich, welches eine Listenansicht aller zum Modell gespeicherten Objekte enthält
Sonderfälle	2a. Es öffnet sich keine Listenansicht, da noch kein Objekt für das jeweilige Modell abgespeichert wurde $\rightarrow$ Der Nutzer wird über das Fehlen von Objekten durch einen Text in der Mitte des Dialogfensters informiert

Tabelle A.3.: Anwendungsfall M3: Gespeicherte Objekte im Modell anschauen

Nummer	**AF-M4**
Anwendungsfall	**Alle Modelle betrachten**
Beschreibung	Alle abgespeicherten Modelle sollen mitsamt wichtiger Informationen besichtigt werden können
Akteur	Nutzer
Auslöser	Das Modell soll eingefroren werden. Das bedeutet, dass der Latent-Replay Algorithmus bei Ende der Trainingssitzung nicht mehr ausgeführt wird.
Vorbedingung	Es wurden bereits ein oder mehrere Modelle gespeichert
Nachbedingung/Ziel	Der Nutzer befindet sich in der Modellübersicht, in welcher die gespeicherten Modelle betrachtet werden können
Nachbedingung im Sonderfall	Die Modellübersicht enthält einen Text welcher daraufhinweist, dass bisher noch kein Modell gespeichert wurde, um einen leeren Bildschirm zu vermeiden
Standardablauf	1. Der Nutzer betätigt den Button in der Navigationsleiste der zur Modellübersicht führt 2. Der Nutzer befindet sich in der Modellübersicht, welche eine Listenansicht der gespeicherten Objekte anzeigt, mitsamt der Anzahl bereits gespeicherter Objekte
Sonderfälle	2a. Es öffnet sich keine Listenansicht, da noch kein Modell abgespeichert wurde. $\rightarrow$ Der Nutzer wird über das Fehlen von Modellen durch einen Text in der Mitte des Bildschirms informiert

Tabelle A.4.: Anwendungsfall M4: Alle Modelle betrachten

Nummer	**AF-M5**
Anwendungsfall	**Modell einfrieren**
Beschreibung	Das Modell soll eingefroren werden. Das bedeutet, dass der Latent-Replay Algorithmus bei Ende der Trainingssitzung nicht mehr ausgeführt wird. Diese Kontrolle dem Nutzer zu übergeben, bietet den Vorteil, dass der Nutzer nicht durch langes Warten bei Ende der Trainingssitzung aufgrund des rechenintensiven Latent-Replay Algorithmus überrascht wird und die Möglichkeit hat das Modell fertigzustellen und nicht weiter zu trainieren.
Akteur	Nutzer
Auslöser	Der Nutzer verlässt die Trainingssitzung durch Wechseln der App mithilfe der Navigationsleiste
Vorbedingung	Nutzer hat in dieser Trainingssitzung ein neues Objekt hinzugefügt
Nachbedingung/Ziel	Das Modell wurde eingefroren
Nachbedingung im Sonderfall	Das Modell wurde nicht eingefroren und der Latent-Replay Algorithmus wurde ausgeführt, um das Phänomen des Katastrophalen Vergessens in der nächsten Trainingssitzung zu verhindern
Standardablauf	1. Nutzer wechselt den Kameratab durch Betätigen eines Reiters in der Navigationsleiste 2. Dialogfenster öffnet sich, welches den Nutzer darüber informiert, dass er nun die Möglichkeit hat sein Modell einzufrieren und welche Folgen dies hat. 3. Nutzer betätigt den Button, der das Modell einfriert 4. Es wird kein aufwendiger Algorithmus bei Ende der Trainingssitzung durchgeführt
Sonderfälle	3a. Nutzer betätigt den Button, der das Modell auf eine nächste Trainingssitzung vorbereitet 3b. Latent-Replay Algorithmus wird ausgeführt und der Nutzer wird durch einen Ladekreis dazu angehalten bis zum Ende der Ausführung mit der Fortsetzung der Nutzung der Applikation zu warten

Tabelle A.5.: Anwendungsfall M5: Modell einfrieren

Nummer	**AF-O1**
Anwendungsfall	**Objekt hinzufügen**
Beschreibung	Objekte sollen hinzugefügt werden können
Akteur	Nutzer
Auslöser	Der Nutzer betätigt den „+“-Button der sich in der Mitte der Navigationsleiste befindet
Vorbedingung	Der Nutzer befindet sich in der Kamera-Ansicht und hat bereits ein Modell hinzugefügt
Nachbedingung/Ziel	Ein Objekt konnte erfolgreich zum derzeitig ausgewählten Modell hinzugefügt werden und kann wiedererkannt werden
Nachbedingung im Sonderfall	Das Objekt wurde nicht abgespeichert
Standardablauf	1. Nutzer befindet sich in der Kamera-Ansicht und betätigt den „+“-Button in der Mitte der Navigationsleiste 2. Ein Dialogfenster öffnet sich, welches den Nutzer darüber informiert was nun zu tun ist. Er hat nun die Möglichkeit diese Meldung nicht mehr anzuzeigen durch das Klicken des Hakens neben dem Text „Nicht mehr anzeigen“ 3. Nutzer betätigt den „Fortfahren“-Button 4. Ein Dialogfenster öffnet sich, welches den Nutzer dazu auffordert dem Objekt Informationen hinzuzufügen 5. Nutzer gibt korrekte Daten ein und startet Training durch das Betätigen des „Training“-Buttons 6. Das Dialogfenster schließt sich und ein Countdown beginnt, welcher das bevorstehende Einlesen ankündigt 7. Das Training wurde gestartet und der Nutzer wird darüber durch Ändern des Textes am rechten oberen Bildschirmran in „Training“ und einem deterministischen Ladekreis im rechten oberen Bildschirmrand informiert 8. Das Training wurde abgeschlossen und der Text am rechten oberen Bildschildrand ändert sich wieder zu „Erkennen“
Sonderfälle	1a. Der Nutzer befindet sich in der Kamera-Ansicht → Die Ansicht wird durch Betätigen des Buttons zur Kamera-Ansicht gewechselt 5a. Der Name des Objekts ist bereits vergeben → Der Nutzer wird darüber informiert und das Training wird noch nicht gestartet 7a. Der Nutzer verlässt frühzeit die Kamera-Ansicht, obwohl das Training noch nicht abgeschlossen ist 7b. Alle Objektplätze des ausgewählten Modells sind bereits belegt → Der Nutzer wird über diesen Fehler informiert. Das Training wird abgebrochen und das Objekt gelöscht

Nummer	**AF-O2**
Anwendungsfall	**Objekte wiedererkennen**
Beschreibung	Bereits eingespeicherte Objekte sollen durch Nutzung der Kamera wiedererkannt werden, wenn sich das Objekt innerhalb der Fokusbox befindet
Akteur	Nutzer
Auslöser	Der Nutzer kann das wiederzuerkennende Objekt in der Kamera-Ansicht sehen, da die Kamera des Mobilgeräts auf dieses Objekt ausgerichtet ist
Vorbedingung	Das Objekt wurde bereits eingespeichert und der Nutzer befindet sich in der Kamera-Ansicht
Nachbedingung/Ziel	Das Objekt wird korrekt erkannt und die Informationen über das Objekt werden am oberen Bildschirmrand angezeigt
Nachbedingung im Sonderfall	Das Objekt wurde nicht erkannt
Standardablauf	1. Nutzer zeigt mit der Kamera auf das Objekt 2. Das Objekt wird erkannt und eingespeicherte Informationen über das Objekt werden für den Nutzer angezeigt
Sonderfälle	2a. Das Objekt wurde nicht erkannt $\rightarrow$ Nutzer hat die Möglichkeit weitere Eindrücke über das Objekt zu sammeln, oder gewisse Einstellungen an der Applikation vorzunehmen, um das Problem selbstständig beheben zu können

Tabelle A.7.: Anwendungsfall O2: Objekt wiedererkennen

Nummer	**AF-O3**
Anwendungsfall	**Objekt bearbeiten**
Beschreibung	Die Attribute, die für das Objekt gespeichert wurden, sollen geändert werden können
Akteur	Nutzer
Auslöser	Der Nutzer betätigt den „Bearbeiten“-Button in der Objektübersicht
Vorbedingung	Der Nutzer befindet sich in der Objektübersicht und es sind bereits Objekte eingspeichert
Nachbedingung/Ziel	Der Nutzer kann durch das Öffnen eines Dialogfensters die bereits gespeicherten Informationen eines Objektes abändern
Nachbedingung im Sonderfall	Es sind keine Objekte eingespeichert und der Nutzer hat dementsprechend keine Möglichkeit ein Objekt abzuändern
Standardablauf	1. Der Nutzer betätigt den „Bearbeiten“-Button in der Objektübersicht 2. Ein Dialogfenster öffnet sich, welches den Nutzer dazu auffordert das Objekt zu bearbeiten 3. Der Nutzer bearbeitet die Informationen korrekt 4. Der Nutzer betätigt den „Bestätigen“-Button und kehrt zur Objektübersicht zurück
Sonderfälle	3a. Der Nutzer gibt einen neuen Objektnamen ein, der bereits existiert → Nutzer wird darauf hingewiesen und die Änderungen werden nicht gespeichert 4a. Der Nutzer betätigt den Abbrechen Button → Die Änderungen werden nicht gespeichert

Tabelle A.8.: Anwendungsfall O3: Objekt bearbeiten

Nummer	**AF-O4**
Anwendungsfall	**Weitere Trainingsdaten für Objekt erfassen**
Beschreibung	Es sollen zu einem bestehenden Objekt weitere Trainingsdaten erfasst werden, um die Erkennungsgenauigkeit für das gewählte Objekt erhöhen zu können
Akteur	Nutzer
Auslöser	Der Nutzer betätigt den „+ “-Button der sich unterhalb des Vorschaubildes des Objekts in der Listenansicht befindet
Vorbedingung	Es wurde bereits ein Objekt abgespeichert und der Nutzer befindet sich in der Objektübersicht
Nachbedingung/Ziel	Es wurde eine neue Trainingssitzung gestartet und dem Objekt wurde weitere Trainingsdaten hinzugefügt
Nachbedingung im Sonderfall	Das Training wurde nicht gestarte
Standardablauf	1. Nutzer betätigt den „+“-Button 2. Die Applikation wechselt in die Kamera-Ansicht und das Training wird nach Ablauf des Countdowns gestartet
Sonderfälle	2a. Das Training konnte nicht gestartet werden → Der Nutzer wird darüber mit einer Fehlermeldung informiert

Tabelle A.9.: Anwendungsfall O4: Weitere Trainingsdaten für Objekt erfassen

Nummer	**AF-O5**
Anwendungsfall	**Objekt löschen**
Beschreibung	Ein Objekt soll gelöscht werden können, sodass dieses nicht mehr von der Applikation erkannt wird
Akteur	Nutzer
Auslöser	Der Nutzer betätigt den „Löschen“-Button
Vorbedingung	Der Nutzer befindet sich in der Objektübersicht und hat bereits ein Objekt eingespeichert
Nachbedingung/Ziel	Das zu löschende Objekt wurde erfolgreich entfernt
Nachbedingung im Sonderfall	Das zu löschende Objekt wurde nicht erfolgreich entfernt
Standardablauf	1. Der Nutzer betätigt den „Löschen“-Button 2. Ein Dialogfenster öffnet sich, welches den Nutzer fragt, ob er sich sicher sei das Objekt zu löschen 3. Der Nutzer bestätigt seine Sicherheit durch Betätigen des Buttons und das Objekt wird gelöscht
Sonderfälle	3a. Der Nutzer betätigt den „Abbrechen“-Button $\rightarrow$ Das Dialogfenster wird geschlossen und das Objekt wird nicht gelöscht

Tabelle A.10.: Anwendungsfall O5: Objekt löschen

Nummer	**AF-O6**
Anwendungsfall	**Alle Objekte betrachten**
Beschreibung	Alle eingespeicherten Objekte sollen mitsamt aller Attribute des Objektes betrachtet werden können
Akteur	Nutzer
Auslöser	Der Nutzer wechselt in die Objektübersicht
Vorbedingung	Der Nutzer bereits Objekte eingespeichert
Nachbedingung/Ziel	Der Nutzer kann nun eine Listenansicht über alle abgespeicherten Objekte betrachten
Nachbedingung im Sonderfall	Es wurden noch keine Objekte abgespeichert. Dementsprechend wurde keine Listenansicht erzeugt
Standardablauf	1. Nutzer wechselt durch Betätigen des jeweiligen Buttons in der Navigationsleiste in die Objektübersicht 2. Nutzer betrachtet eine Liste, welche alle Objekte mit den zugehörig abgespeicherten Daten beinhaltet
Sonderfälle	2a. Es wurde noch kein Objekt abgespeichert $\rightarrow$ Es wird in der Mitte des Bildschirm ein Informationstext angezeigt, um eine komplett leere Ansicht zu vermeiden

Tabelle A.11.: Anwendungsfall O6: Alle Objekte betrachten

Nummer	**AF-E1**
Anwendungsfall	**Einstellen der Anzahl der hinzuzufügenden Daten pro Training**
Beschreibung	Die Anzahl der Trainingsdaten die pro Objekttraining erfasst werden, soll konfigurierbar sein, um Kontrolle über die Dauer und die Genauigkeit der Erkennung zu erhalten
Akteur	Nutzer
Auslöser	Der Nutzer betätigt den jeweiligen Schieberegler in den App-Einstellungen
Vorbedingung	Der Nutzer befindet sich in den App-Einstellungen
Nachbedingung/Ziel	Die Anzahl der hinzuzufügenden Daten pro Training wurde auf den neuen Wert angepasst
Nachbedingung im Sonderfall	Änderung konnte nicht vollzogen werden
Standardablauf	1. Der Nutzer stellt den Schieberegler auf einen neuen Wert 2. Die Anzahl der Trainingsdaten pro Objekttraining wird angepasst
Sonderfälle	2a. Anzahl der Trainingsdaten pro Objekttraining kann nicht angepasst werden → Der Nutzer wird darüber durch eine Fehlermeldung informiert

Tabelle A.12.: Anwendungsfall E1: Einstellen der Anzahl der hinzuzufügenden Daten pro Training

Nummer	**AF-E2**
Anwendungsfall	**Einstellen der Dauer des Countdowns vor dem Training**
Beschreibung	Der unmittelbar vor dem Training laufende Countdown, soll konfigurierbar sein, um unnötige Wartezeiten, oder notwendige Zeiten für die korrekte Ausrichtung der Kamera zu ermöglichen
Akteur	Nutzer
Auslöser	Der Nutzer betätigt den jeweiligen Schieberegler in den App-Einstellungen
Vorbedingung	Der Nutzer befindet sich in den App-Einstellungen
Nachbedingung/Ziel	Die Länge des Countdowns wurde auf den neuen Wert gesetzt
Nachbedingung im Sonderfall	Die Länge des Countdowns konnte nicht auf den neuen Wert gesetzt werden
Standardablauf	1. Der Nutzer stellt den Schieberegler auf einen neuen Wert 2. Die Länge des Countdowns wird angepasst
Sonderfälle	2a. Die Länge des Countdowns konnte nicht angepasst werden → Der Nutzer wird darüber durch eine Fehlermeldung informiert

Tabelle A.13.: Anwendungsfall E2: Einstellen der Dauer des Countdowns vor dem Training

Nummer	**AF-E3**
Anwendungsfall	**Einstellen der Auflösung der hinzuzufügenden Trainingsdaten**
Beschreibung	Die Größe der Fokusbox im Kamerahauptbildschirm soll konfigurierbar sein, um nahe beziehungs Objekte in der Ferne zu erfassen, ohne zu viel Hintergrund einzufangen
Akteur	Nutzer
Auslöser	Der Nutzer betätigt den jeweiligen Schieberegler in den App-Einstellungen
Vorbedingung	Der Nutzer befindet sich in den App-Einstellungen
Nachbedingung/Ziel	Die Auflösung der hinzuzufügenden Trainingsdaten wird angepasst
Nachbedingung im Sonderfall	Die Auflösung der hinzuzufügenden Trainingsdaten konnte nicht angepasst werden
Standardablauf	1. Der Nutzer stellt den Schieberegler auf einen neuen Wert 2. Die Auflösung der hinzuzufügenden Trainingsdaten wird angepasst
Sonderfälle	2a. Die Auflösung der hinzuzufügenden Trainingsdaten konnte nicht angepasst werden → Der Nutzer wird darüber durch eine Fehlermeldung informiert

Tabelle A.14.: Anwendungsfall E3: Einstellen der Auflösung der hinzuzufügenden Trainingsdaten

Nummer	**AF-E4**
Anwendungsfall	**Einstellen des Konfidenzschwellwerts bei Inferenz**
Beschreibung	Der Konfidenzschwellwert soll konfigurierbar sein. Dies bedeutet, dass ein Objekt erst im Overlay angezeigt wird, wenn das Modell mit einer bestimmen Erkennungssicherheit (Konfidenz) das Objekt richtig erkannt hat
Akteur	Nutzer
Auslöser	Der Nutzer betätigt den jeweiligen Schieberegler in den App-Einstellungen
Vorbedingung	Der Nutzer befindet sich in den App-Einstellungen
Nachbedingung/Ziel	Der Konfidenzschwellwert wurde auf den neuen Wert gesetzt
Nachbedingung im Sonderfall	Der Konfidenzschwellwert konnte nicht auf den neuen Wert gesetzt werden
Standardablauf	1. Der Nutzer stellt den Schieberegler auf einen neuen Wert 2. Der Konfidenzschwellwert wird angepasst
Sonderfälle	2a. Der Konfidenzschwellwert wird nicht angepasst $\rightarrow$ Der Nutzer wird darüber durch eine Fehlermeldung informiert

Tabelle A.15.: Anwendungsfall E4: Einstellen des Konfidenzschwellwerts bei Inferenz

Nummer	**AF-E5**
Anwendungsfall	**Applikation zurücksetzen**
Beschreibung	Alle persistent gespeicherten Daten sollen zurückgesetzt werden können, um eine Alternative zur Neuinstallation der Applikation zu schaffen
Akteur	Nutzer
Auslöser	Der Nutzer betätigt das Feld in den App-Einstellungen, welches für das Zurücksetzen der Applikation zuständig ist
Vorbedingung	Der Nutzer befindet sich in den App-Einstellungen
Nachbedingung/Ziel	Alle Modelle, Objekte und sonstige Konfigurationen wurden auf Werkseinstellungen zurückgesetzt
Nachbedingung im Sonderfall	Das Zurücksetzen war nicht erfolgreich
Standardablauf	1. Der Nutzer betätigt das Feld 2. Ein Dialogfenster erscheint, welches den Nutzer darauf hinweist, dass das Zurücksetzen der Applikation nicht rückgängig gemacht werden kann und dass alle eingespeicherten Daten gelöscht werden 3. Der Nutzer bestätigt das Zurücksetzen 4. Die Applikation wird zurückgesetzt und neu gestartet
Sonderfälle	3a. Der Nutzer betätigt den „Abbrechen “-Button und das Dialogfenster schließt sich. Der Vorgang wird abgebrochen 4a. Die Applikation konnte nicht zurückgesetzt werden → Der Nutzer wird darüber durch eine Fehlermeldung informiert und der Vorgang wird abgebrochen

Tabelle A.16.: Anwendungsfall E5: Applikation zurücksetzen

B. Anhang B