HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

Innovation. Development. Namics.
A Merkle Company

beutler-coaching.ch

Videoanalyse für fliegende Rettungshunde

Bachelorarbeit

Abteilung Informatik
Hochschule für Technik Rapperswil

Frühjahrssemester 2019

Autor(en):	Dominik Kessler und Cyrill Hänni
Betreuer:	Prof. Dr. Farhad D. Mehta
Projektpartner:	Namics AG und Beutler Coaching
Experte:	Dr. Peter Dürr
Gegenleser:	Prof. Stefan F. Keller

Abstract

Unsere Projektpartner möchten Drohnen zur Unterstützung der alpinen Rettung einsetzen. Die Drohnen können bereits vordefinierte Routen autonom abfliegen und dabei Videos aufnehmen. In dieser Bachelorarbeit geht es darum herauszufinden, ob es möglich ist, Personen und Gegenstände in diesen Videos automatisch zu finden.

Damit diese Automatisierung hilfreich ist, muss sie schneller sein, als ein Mensch bräuchte um das Video zu durchsuchen. Dabei ist man durch die Infrastruktur und die örtlichen Gegebenheiten stark eingeschränkt. Konkret bedeutet das, dass man nur ein Smartphone oder Macbook mit schlechter oder gar keiner Internetverbindung zur Verfügung hat.

Der Fokus lag auf der Machbarkeitsanalyse und der Technologieevaluation, in denen bestehende Frameworks und Algorithmen geprüft und verglichen wurden.

Wir haben uns entschieden das Problem mithilfe des, auf Deep Learning basierenden, Object Detection Algorithmus YOLO zu lösen.

Nachdem das Resultat der Evaluation positiv ausfiel, haben wir eine macOS Applikation implementiert, welche Objekte in Videofiles finden kann.

Dafür haben wir Training- und Testdaten so realitätsnah wie möglich gesammelt und anschliessend mit Turi Create ein neuronales Netzwerk Modell für CoreML trainiert.

Das selbe Modell konnte auch in die iOS App, aus der Vorgängerarbeit, integriert werden, um den Livestream einer Drohne zu analysieren.

Zusätzlich wurden auch Konzepte für die Optimierung der Analyse erarbeitet. Ein Beispiel dafür ist, nur die geänderten Bereiche der Frames zu analysieren um die Rechenleistung gezielter einzusetzen.

Management Summary

Ausgangslage

In der alpinen Rettung kann es immer wieder zu Situationen kommen, in denen die Retter mit den herkömmlichen Mitteln wie Hunden oder Helikoptern Schwierigkeiten haben, eine vermisste Person zu suchen, da das Wetter oder das Gelände dies nicht zulässt.

In einem Vorgänger-Projekt dieser Bachelorarbeit wurde der Einsatz von Drohnen in der alpinen Rettung untersucht. Die Drohne fliegt eine Route ab und nimmt dabei ein Video auf. Das Video wird anschliessend von den Rettern angeschaut und nach Personen oder Objekten, welche Hinweise auf den Verbleib von vermissten Personen geben könnten, durchsucht. Wie man sich vorstellen kann, ist das Suchen innerhalb des Videos zeitaufwendig und anstrengend.

Bei dieser Bachelorarbeit geht es jetzt darum diesen manuellen Suchprozess zu automatisieren.

Vorgehen und Ergebnisse

Der Fokus der Arbeit war abzuklären, ob das Suchen von Objekten grundsätzlich automatisiert werden kann und was für Technologien diese Problemstellung lösen können.

Darum wurde in einem ersten Schritt eine Technologieevaluation und Machbarkeitsanalyse durchgeführt. Dabei wurden verschiedene Object Detection Algorithmen und Libraries untersucht und miteinander verglichen.

Die Machbarkeitsanalyse ist positiv ausgefallen und in der Technologieevaluation haben wir uns entschieden, Apples Vision Framework einzusetzen.

Aufgrund der Evaluation wurde anschliessend eine macOS Applikation implementiert, welche Videos nach Gegenständen bzw. Personen durchsucht und diese anzeigt.

Zusätzlich konnten wir auch das Fernziel Echtzeitanalyse umsetzen. Dafür haben wir die bestehende iOS Applikation aus der Vorgängerarbeit so angepasst, dass der Videostream der Drohne für die Analyse verwendet wird. Die gefundenen Objekte werden dem Benutzer in Echtzeit angezeigt.

Ausblick

Neben der Evaluation und den beiden Applikationen, wurden auch Konzepte für Erweiterungen und Optimierungen erarbeitet.

Ein Beispiel dafür ist eine Feedbackplattform, welche es ermöglichen soll, das Modell automatisch, mithilfe von Benutzerrückmeldungen, zu verbessern.

Ebenfalls wurde untersucht wie man das Resultat der Analyse verbessern könnte, indem man zum Beispiel nur die Bereiche eines Frames anschaut, welche nicht schon zuvor in einem anderen Frame analysiert wurden.

Danksagung

An dieser Stelle wollen wir uns noch bei allen bedanken, die uns während dieser Bachelorarbeit unterstützt haben.

Speziell wollen wir unseren beiden Projektpartnern Heinz Beutler und Beat Helfenberger danken, die uns mit ihren Visionen immer wieder begeistern konnten.

Auch bei Lukas Oberholzer und David Riederer, welche das Vorgängerprojekt durchgeführt haben, wollen wir uns für ihre Unterstützung und die Einführung in die bestehende Applikation bedanken.

Des Weiteren möchten wir noch uns noch bei unserem Betreuer Prof. Dr. Farhad Mehta bedanken, welcher uns während der Arbeit mit wertvollem Rat zur Seite stand.

Zusätzlich danken wir unserem Experten Dr. Peter Dürr und unserem Gegenleser Prof. Stefan F. Keller für ihre interessanten Anregungen.

Inhaltsverzeichnis

Kapitel 1

Projektplan

1.1 Projektübersicht

Die Bergrettung ist bei der Suche von vermissten Personen durch natürliche Umstände, wie Wetter und Gelände, teilweise stark eingegrenzt. Um dem entgegen zu wirken, werden die Retter mit Helikoptern und neuerdings auch von Drohnen von der Luft aus unterstützt.

Der Vorteil von Drohnen ist dabei, dass sie auch bei Wetter fliegen können, bei dem die Helikopter am Boden bleiben müssen.

Während des Fluges zeichnen die Rettungsdrohnen Videos auf, welche anschliessend von mehreren Bergrettern auf einem Bildschirm angeschaut werden, um Objekte wie beispielsweise Rucksäcke oder Personen zu finden. Dieser manuelle Prozess ist zeitaufwendig und soll nun mit einer automatisierten Videoanalysesoftware optimiert werden.

Die Software soll das Video schnell nach Personen oder Objekten durchsuchen, welche Hinweise auf den Verbleib von Personen liefern könnten und den Rettern die entsprechenden Frames für die manuelle Beurteilung anzeigen.

Das Ziel des Projektes ist es eine Machbarkeitsanalyse und Technologieevalution für die Videoanalyse durchzuführen. Dabei soll die Analyse nicht auf Videos von Drohnen limitiert sein. Man könnte sich beispielsweise auch vorstellen, Kameras unter einer Gondelbahn anzubringen und deren Videos für die Analyse zu verwenden.

1.2 Endprodukt

Damit die Auftraggeber Namics AG und Beutler Coaching die Arbeit weiterziehen können, werden ihnen am Ende der Arbeit folgende Dokumente und Resultate zur Verfügung gestellt.

- Dokumentation

  - Machbarkeitsanalyse
  - Technologieevaluation
- Video Analyse Tool (wenn Machbarkeitsanalyse zeigt es ist möglich)
- Quellcode

1.3 Rahmenbedingungen

1.3.1 Zeitliche Bedingungen

Das Projekt wird im Rahmen einer Bachelorarbeit durchgeführt und hat daher die zeitlichen Rahmenbedingungen der HSR.

Den Studenten werden bei einer erfolgreichen Durchführung 12 ETC-Punkte angerechnet. Ein ECTS-Punkt entspricht 30 Arbeitsstunden. Somit haben wir 360 Arbeitsstunden pro Person zur Verfügung. Diese Zeit wird auf 17 Wochen aufgeteilt, wobei in den letzten beiden Wochen in einem 100% Pensum gearbeitet werden kann.

Dauer	17 Wochen
Start	18.02.2019
Ende	14.06.2019
Anzahl Studenten	2
Zeit	720

Tabelle 1.1: Zeitliche Bedingungen

1.3.2 Technische Einschränkungen

Internet Während Rettungseinsätzen ist oft gar keine oder höchstens eine beschränkte Internetverbindung verfügbar. Daher sollte die Applikation während des Einsatzes nicht von Cloud Services abhängig sein.

Hardware Die entwickelte Applikation muss auf einem Rechner der Wahl der Ansprechpartner ausführbar sein.

Vorhandene Hardware: Macbook Pro (Radeon Pro 555), iPad Pro (2018, 12.9”, 256GB), iPhone, Android Smartphone. Die Hardwareanforderungen können mit dem Ansprechpartner jedoch abgesprochen werden.

Technologien Technologiewahl muss mit den Projektpartnern abgesprochen werden.

Falls möglich, sollten Technologien aus bisherigen Arbeiten, weiterverwendet werden. Bestehende Technologien für das Backend sind Java, Spring Boot und MySQL auf einem Linux Server. Daneben gibt es native iOS (Swift) und Android (Java) Apps.

1.4 Projekt Zeitplan

1.4.1 Phasen

Inception In der Inception Phase geht es darum einen Projektplan zu definieren und sowohl funktionale, als auch nichtfunktionale Anforderungen festzulegen.

Elaboration Die Elaboration Phase wird in diesem Projekt den grössten Teil beanspruchen. In dieser Phase werden wir verschiedene Libraries und Lösungsansätze miteinander vergleichen und uns schlussendlich für eine Technologie entscheiden mit der wir das Endprodukt implementieren werden.

Ein alternativer Ausgang dieser Phase wäre, dass sich herausstellt, dass die Anforderungen, mit den gegebenen Einschränkungen, technisch nicht erfüllbar wären.

Construction In der Construction Phase werden wir die in der Elaboration Phase gewählte Technologie, in eine benutzerfreundliche Applikation verpacken, die auf einem der Geräte des Auftraggebers lauffähig ist.

Für den Fall, dass die Elaboration Phase zeigt, dass es technisch nicht möglich ist, werden wir in dieser Phase ein alternatives Use Case implementieren. Siehe Sektion 1.8.

Transition In der Transition Phase werden wir das Projekt an den Auftraggeber übergeben und die Bachelorarbeit abgeben.

1.4.2 Meilensteine

Meilenstein	Datum
Ende der Inception-Phase	25.02.2019
Zwischenpräsentation	16.04.2019
Ende der Elaboration-Phase	28.04.2019
Ende der Construction-Phase	31.05.2019
Ende der Transition-Phase	05.06.2019
Reserve	07.06.2019
Abgabe Arbeit, Abstract und Poster	07.06.2019
Endpräsentation	14.06.2019

Tabelle 1.2: Meilensteine

1.5 Risikomanagement

1.5.1 Risiken

Nr.	Beschreibung	Ausmass	Wahrsch.
R1	Machbarkeitsanalyse zeigt, dass das Problem nicht lösbar ist	Hoch	Tief
R2	Videoanalyse ist zu langsam	Hoch	Mittel
R3	Zu viele False Positives bzw. False Negatives	Mittel	Mittel
R4	Komplexität für Studenten zu hoch	Hoch	Mittel
R5	Ausfälle von Teammitgliedern	Mittel	Tief
R6	Research / Elaboration dauert länger als erwartet	Mittel	Mittel

Tabelle 1.3: Risiken

1.5.2 Risikobewältigung

Nr.	Umgang & Massnahmen
R1	Dass das Problem nicht lösbar ist, ist ein valides Resultat der Bachelorarbeit. Zusätzliche Arbeiten sind in der Aufgabenstellung aufgelistet. Durch bestehende Technologien auf dem Markt, kann man jedoch davon ausgehen, dass das Risiko sehr tief ist.
R2	Hardwareanforderungen erhöhen, das Bild in tieferer Auflösung analysieren, nur jedes x-te Videoframe analysieren oder anderen Lösungsansatz versuchen.
R3	Threshold erhöhen bzw. verringern oder anderen Ansatz versuchen.
R4	Studenten belegten das Modul Statistical Machine Learning und besuchen während der Arbeit das Deep Learning Modul. Des Weiteren, wird auf bestehende Libraries gesetzt, welche komplizierte Details abstrahieren, statt Algorithmen von Grund auf zu implementieren.
R5	Es wurde eine Zeitreserve eingeplant.
R6	Aufgrund der Art des Projektes nimmt die Elaboration Phase mehr Zeit ein als üblich. Das ist bewusst so gewählt, da der Fokus des Projektes auf der Analyse liegt und das Endprodukt sekundär ist. Wir haben uns jedoch Meilensteine gesetzt, sodass wir den zeitlichen Verlauf des Projektes im Auge behalten können.

Tabelle 1.4: Risikobewältigungsmassnahmen

1.6 Infrastruktur

Version Control Ein Repository auf GitLab wird vom Auftraggeber zur Verfügung gestellt.

Hardware Die HSR stellt einen iMac und eine Workstation mit Grafikkarte zur Verfügung.

1.7 Qualitätsmassnahmen

1.7.1 Dokumentation

Für die Dokumentation verwenden wir LaTeX. Die LaTeX Dateien legen wir im gleichen Repository wie der Code ab. Für den Reviewprozess verwenden wir Merge Requests.

Quelldateien für Grafiken etc., welche wir in die Dokumentation einbeziehen, legen wir ebenfalls im Repository ab, damit diese veränderbar bleiben.

Da beide Partnerunternehmen deutschschweizerische Unternehmen sind, wird die Dokumentation in Deutsch geschrieben.

1.7.2 Projekt Management

Wir werden agil arbeiten, jedoch keine Sprints verwenden, sondern uns an Kanban orientieren. Wir entscheiden uns für diesen Ansatz, da Sprints besonders dann Sinn machen, wenn ein Produkt nach dem Minimum Viable Product (MVP) Ansatz entwickelt wird.

Unser Projekt ist aber noch nicht reif genug, um nach diesem Ansatz entwickelt zu werden. Zuerst muss viel Zeit in die Evaluation und den Vergleich von verschiedenen Ansätzen und Technologien investiert werden, was eine genaue Sprint Planung am Anfang sehr schwierig macht und uns keinen Mehrwert bietet.

Konkret werden wir mit GitLab Issues arbeiten. Jede Aufgabe wird als GitLab Issue erfasst. Im GitLab wird ein Backlog mit den verschiedenen Aufgaben geführt, welches nach Priorität sortiert ist. Sobald ein Gruppenmitglied Kapazität hat eine Aufgabe aufzunehmen, wird es sich die Höchstpriorisierte zuweisen und der Issue das Label „in Progress“ geben.

Für jede Aufgabe wird ein Feature Branch für die Implementation und Dokumentation erstellt. Sobald die Implementation beendet ist, wird ein Merge Request in den Master Branch erstellt, welcher vom anderen Teammitglied überprüft werden muss.

Wenn das Review abgeschlossen ist und der Merge Request gemerged wurde, wird die dazugehörende Issue geschlossen und das „in Progress“ Label entfernt. Anschliessend kann das Teammitglied die nächste Aufgabe aus dem Backlog übernehmen.

1.7.3 Sitzungen

Wir werden alle zwei Wochen eine Sitzung mit unseren Auftraggebern halten, um sie über den aktuellen Stand der Arbeit auf dem Laufenden zu halten, sowie um allfällige Fragen zu klären.

Mit unserem Betreuer Prof. Dr. Farhad Mehta werden wir anfangs wöchentliche Sitzungen durchführen. Wenn wir später sehen, dass das zu häufig ist, werden wir das auf eine Sitzung alle zwei Wochen reduzieren.

Für jede Sitzung werden wir im GitLab eine Wiki Seite erstellen. Vor der Sitzung werden wir darin die Agenda und offene Fragen erfassen und den Link allen Teilnehmern per Mail zustellen, damit sie sich vorbereiten können. Auf der gleichen Seite, wird dann das eigentliche Protokoll verfasst.

1.8 Alternative Use Cases

Falls die Elaboration Phase zeigt, dass die Videoanalyse mit den gegebenen Einschränkungen nicht möglich wäre, kann auch einer der folgenden alternativen Use Cases implementiert werden. Wir werden diese Use Cases hier nicht im Detail beschreiben. Die genauen Anforderungen würden wir erst im Fall, dass wir tatsächlich eine Alternative wählen müssten, ausarbeiten.

- 3D-Visualisierung der Einsätze
- 3D-Simulation der Drohnenflüge
- Drohne mit VR-Brille (FPV-Flüge)
- Cockpit Mobile App (Android/iOS)
- Mobile Tracker-Erweiterung Integration z.B. mit Smartwatch, GPS-Tracker
- Drohnen Sensorik (Radar/Wärmebild/etc.)
- Life Video Streaming ins Cockpit

Kapitel 2

Anforderungen

2.1 Funktionale Anforderungen

2.1.1 Eigenständige macOS Applikation

Das Endprodukt deckt den Use Case ab, dass ein Video von einer Quelle, wie einer Drohne oder eines Helikopters, nach vermissten Personen oder Objekten, welche Hinweise auf den Verbleib von Personen geben könnten, durchsucht wird.

Die Applikation nimmt als Input eine Videodatei entgegen, durchsucht das Video auf die oben erwähnten Gegenstände und Personen und zeigt dem Benutzer die interessanten Stellen im Video an, welche dieser dann überprüfen kann.

So müssen die Bergretter nicht mehr das gesamte Video anschauen, sondern können sich auf die, durch die Software erstellte Vorauswahl von Frames konzentrieren und so Zeit einsparen.

Dem Nutzer soll auch ersichtlich sein, an welcher GPS-Position sich die Drohne/der Helikopter befunden hat, als dieser Videoabschnitt aufgenommen wurde.

2.1.2 Realtime Analyse in der iOS Applikation

Wie in der Aufgabenstellung definiert, existiert das Fernziel, die Videoanalyse in Echtzeit durchzuführen.

Dabei wird die bestehende iOS Applikation so angepasst, dass sie den Kamerastream, welcher von der Drohne übertragen wird, in Echtzeit analysiert.

Dem Piloten sollte ein Feedback, wie zum Beispiel ein Audiosignal, gegeben werden, falls etwas Ungewöhnliches gefunden wurde. So hätte der Pilot die Möglichkeit einzugreifen und den Gegenstand genauer anzusehen.

2.2 Nicht Funktionale Anforderungen

2.2.1 Benutzbarkeit

Ein Benutzer sollte ohne grosse Erklärung in der Lage sein, die Videoanalyse zu starten und die Auswertung anzuschauen.

Es muss möglich sein, alle Konfigurationsparameter(Anzahl Frames, Threshold, CPU/GPU) zur Laufzeit zu ändern.

2.2.2 Performance

Die Applikation soll das Video schneller analysieren können, als es dauern würde, wenn ein Mensch das Video anschaut.

2.2.3 Sicherheit

Durch die Sensitivität der Videodateien, dürfen diese für die Analyse keinesfalls auf nicht kontrollierbaren Dritt-Systemen abgelegt werden. Kontrollierbar bedeutet, die Projektpartner können festlegen, welche Personen bzw. Applikationen auf die Daten zugreifen können.

2.2.4 Infrastruktur

Aufgrund lokaler Gegebenheiten, muss davon ausgegangen werden, dass während der Videoanalyse nur eine schlechte oder gar keine Internetverbindung vorhanden ist. Das bedeutet, dass die Software auch ohne Internetverbindung lauffähig sein soll.

Die Software muss auf einem, der vom Auftraggeber vorgegebenen Geräte, lauffähig sein. Siehe Abschnitt 1.3.2 für Auflistung.

Lauffähigkeit auf unterschiedlichen Plattformen wäre erstrebenswert, aber die Performance hat Vorrang.

2.2.5 Technologie

Die eingesetzten Technologien sollten mit den bestehenden Systemen übereinstimmen, siehe Abschnitt 1.3.2. Jedoch dürfen Abweichungen mit dem Auftraggeber abgesprochen werden.

Aufgrund der weiten Verbreitung von Python im Machine Learning Umfeld gehört die Programmiersprache ebenfalls zu den erlaubten Technologien.

Kapitel 3

Technologieevaluation und Machbarkeitsanalyse

Das Ziel der Machbarkeitsanalyse ist es herauszufinden, wie wir Objekte und Personen in Videodateien erkennen können und ob es möglich ist, dies auf einem Macbook schneller zu tun, als das ein Mensch manuell machen könnte.

Eng damit verbunden ist die Technologieevaluation, in der es darum geht, zu entscheiden, mit welcher Technologie wir die Applikation implementieren werden. Dies setzt natürlich ein positives Ergebnis der Machbarkeitsanalyse voraus.

3.1 Entscheidungen zum Lösungsansatz

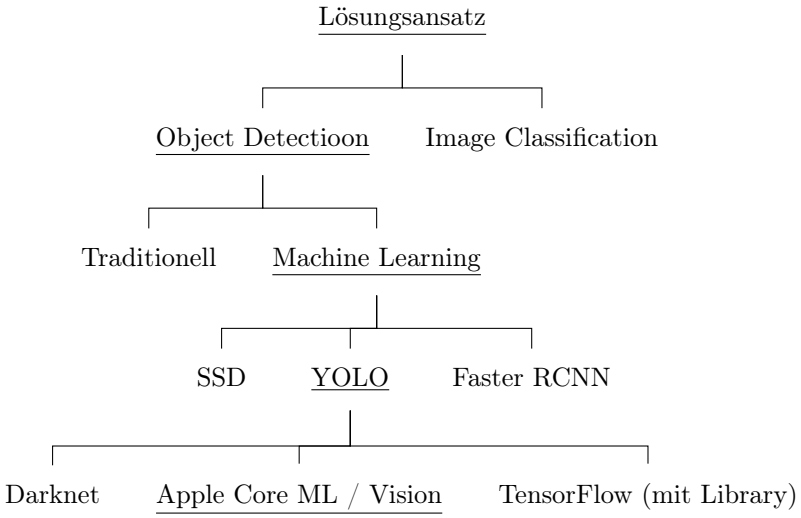

Abbildung 3.1: In diesem Baumdiagramm sieht man eine Übersicht über die Entscheidungen, auf verschiedenen Stufen, welchen Lösungsansatz wir verfolgen sollen. Der unterstrichene Ansatz ist jeweils der, den wir weiter verfolgt haben. In den folgenden Abschnitten, sind diese Entscheidungen genauer erläutert.

3.1.1 Object Detection oder Image Classification?

Die beiden verbreitetsten Methoden zur Analyse von Bildern sind Image Classification und Object Detection.

Bei der **Image Classification** geht es darum, einem Bild ein Label zuzuweisen[1, Seite xi]. In unserem Fall könnten das beispielsweise die beiden Labels „Enthält Gegenstände oder Personen“ und „Enthält keine Gegenstände oder Personen“ sein.

Der Nachteil daran ist, dass man nicht erfährt, wo im Bild sich diese Objekte befinden. Des Weiteren sind wir nicht sicher, ob die korrekte Klassifizierung des gesamten Bildes aufgrund eines Objektes, wie eines Rucksacks, in einem kleinen Ausschnitt des Bildes überhaupt möglich ist.

Besser ist der Ansatz von **Object Detection**. Dabei ist das Ziel nicht nur zu erkennen, ob das Bild bestimmte Objekte enthält, sondern auch wo im Bild sich die Objekte befinden[1, Seite xi].

Aufgrund dieses Vorteils werden wir uns im weiteren Verlauf der Elaboration auf Object Detection beschränken.

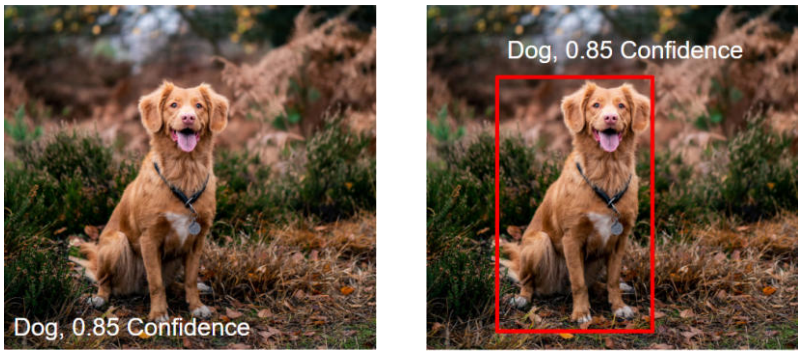

Abbildung 3.2: Unterschied zwischen Image Classification (Links) und Object Detection (Rechts). Bildquelle: [38]

3.1.2 Machine Learning oder traditioneller Ansatz?

Unter traditionellem Ansatz verstehen wir in diesem Kontext eine algorithmische Lösung ohne Machine Learning.

Für Object Detection gibt es einige Ansätze, die versuchen ohne Machine Learning Objekte in Bildern zu finden[35][45]. Diese Ansätze setzen allerdings oft voraus, dass die Erkennung in einem kontrollierten Rahmen passiert. Zum Beispiel könnte vorausgesetzt werden, dass die gesuchten Objekte genau bekannt sind oder der Hintergrund unifarben ist.

Die Machine Learning Ansätze für Object Detection[17][33][34] sind viel flexibler und erkennen Objekte auch auf komplexen Hintergründen. Ein weiterer Vorteil ist das einfache trainieren von Modellen aufgrund von Beispielbildern. Wir werden also den Machine Learning Ansatz weiterverfolgen.

3.1.3 Welcher Object Detection Algorithmus?

Bei der Entscheidung, welchen Object Detection Algorithmus man verwenden soll, muss stets zwischen der **Geschwindigkeit** und der **Genauigkeit** abgewogen werden.

Ist die Analyse zu langsam, wäre man schneller, wenn man das Video manuell anschauen würde. Ist die Analyse zwar schnell, liefert aber viele Fehleinschätzungen, kann man sich nicht auf die Resultate verlassen.

Die Applikation muss auf einem Macbook oder iOS Gerät lauffähig sein, wodurch keine grosse GPU Leistung zur Verfügung steht. Darum haben wir uns entschieden, uns in der folgenden Technologieevaluation auf YOLO zu konzentrieren, da es aktuell der schnellste bekannte Object Detection Algorithmus ist.[32]

Falls sich YOLO in unseren Tests als nicht schnell genug herausstellt, können wir daraus schliessen, dass die Videoanalyse in Echtzeit auf der verfügbaren Hardware nicht möglich ist.

Sollten wir jedoch feststellen, dass YOLO nicht genau genug ist, werden wir auf einen anderen Algorithmus ausweichen.

3.1.4 Welche Object Detection Library?

Wir möchten eine Library benutzen, welche YOLO implementiert und Tools zum trainieren von Modellen zur Verfügung stellt.

Dafür kommen verschiedene Libraries, basierend auf unterschiedlichen Technologien, in Frage. Schlussendlich haben wir uns für Apples Vision Framework[47] entschieden. Dem Vergleich der verfügbaren Frameworks und Libraries haben wir einen eigenen Abschnitt gewidmet, siehe 3.2. Dort kann auch nachgelesen werden, warum wir uns für das Vision Framework entschieden haben.

3.2 Object Detection Library

Nachdem wir uns für einen Algorithmus entschieden haben, müssen wir entscheiden, welche Library wir verwenden werden.

3.2.1 Ausschlusskriterien

Alle Libraries müssen folgende Ausschlusskriterien erfüllen, damit wir sie in die engere Auswahl aufnehmen:

YOLO Unterstüzung Da wir zum Schluss gekommen sind, den YOLO Algorithmus zu verwenden, muss die eingesetzte Library diesen auch unterstützen.

Modelle trainieren Wir müssen in der Lage sein, eigene Modelle zu trainieren bzw. zu verwenden.

Aktive Community Ein weiteres Kriterium ist, dass die Library immer noch aktiv in Entwicklung ist und auch einen gewissen Popularitätsgrad hat. Wir möchten keine Abhängigkeit haben, die in nächster Zeit wieder verschwindet bzw. nur als Experiment entwickelt wurde.

3.2.2 Nice to have

Folgende Features sind keine zwingenden Anforderungen, aber es wäre ideal, wenn die Library möglichst viele dieser Punkte erfüllen würde.

Analyse von Videodateien Die Analyse von Videodateien sollte unterstützt werden, falls das nicht der Fall ist, kann man immer noch mit einer weiteren Library die Frames der Videodateien extrahieren und diese einzeln analysieren.

Einfache API Die API zum Trainieren bzw. zum Analysieren sollte verständlich und gut dokumentiert sein.

Einfache Integration in bestehende Applikationen Die Library soll möglichst einfach in die bestehenden Apps integriert werden können.

GPU Support Die Möglichkeit GPUs für die Analyse beizuziehen, würde den Prozess beschleunigen.

Mobile Support Damit wir das Fernziel Echtzeitanalyse erreichen können, müsste die Library auch auf einem Smartphone oder Tablet eingesetzt werden können. Sollte das nicht gehen, könnte man immer noch das Modell konvertieren und eine weitere Library verwenden.

3.2.3 Darknet

Darknet[31] ist eine Deep Learning Library von Joseph Redmon, einem der Autoren des YOLO Papers[33]. Wie daher zu erwarten, wird YOLO ohne zusätzliche Library von Darknet unterstützt.

Darknet ist in C geschrieben und läuft auf Unix Systemen. Es gibt auch eine angepasste Versionen, welche auch auf Windows lauffähig ist[50].

Darknet unterstützt Nvidia Grafikprozessoren für eine schnelle Analyse. Die neuen Geräte von Apple haben jedoch AMD GPUs und Darknet kann diese somit nicht ansprechen.

Die Library bietet ein Command Line Interface zum Analysieren von Bild- und Videodateien, sowie zum Trainieren von Modellen. Es gibt allerdings keine dokumentierte API und man müsste den Source Code reverse engineeren, um eine taugliche Lösung anbieten zu können.

Die Installation für Videoanalyse mit GPU-Unterstützung ist dürftig dokumentiert und es waren mehrere Anläufe nötig, bis alle nötigen Abhängigkeiten in der richtigen Version kompiliert und installiert waren.

3.2.4 TensorFlow

TensorFlow[40] von Google ist die wohl bekannteste Deep Learning Library und unterstützt von Haus aus Object Detection. Allerdings wird kein Modell mit YOLO zur Verfügung gestellt, sondern nur SSD und verschiedene Versionen von R-CNN[41].

Es sind aber diverse, auf TensorFlow basierende, YOLO-Implementationen verfugbar. Auf diese werden wir separat eingehen.

TensorFlow unterstützt GPU Beschleunigung mit CUDA. Allerdings ist die Installation der Version mit GPU Unterstützung nicht ganz einfach, da viele Abhängigkeiten zuvor manuell installiert werden müssen. Besonders die Installation von CUDA stellte sich als Herausforderung heraus. Es muss dabei sehr genau beachtet werden, welche Version man installiert, damit sie mit der gewählten TensorFlow Version kompatibel ist.

3.2.5 Caffe2

Caffe2[3] ist ein Deep Learning Framework von Facebook. Genau wie TensorFlow bietet Caffe2 keine eigene Implementation von YOLO, aber es sind Versionen von anderen Entwicklern verfügbar. Im Unterschied zu den YOLO-Libraries basierend auf TensorFlow, scheinen diese, aufgrund der geringen Anzahl an GitHub Sternen und der kleinen Anzahl an Contributors, aber eher experimentell und nicht reif für den produktiven Einsatz.

Beispiele von Caffe2 YOLO Implementationen:

- `https://github.com/dongfangduoshou123/Caffe2-yolo-v3`
- `https://github.com/KleinYuan/caffe2-yolo`

3.2.6 PyTorch

PyTorch[29] ist ein weiteres populäres Machine Learning Framework. Wie TensorFlow und Caffe2 unterstützt es YOLO nicht von Haus aus, sondern es muss eine zusätzliche Library dafür verwendet werden. Ebenfalls wie bei Caffe2 haben wir keine Library gefunden, welche YOLO implementiert und die wir als ausgereift genug betrachten.

Beispiele von PyTorch YOLO Implementationen:

- `https://github.com/ayooshkathuria/pytorch-yolo-v3`
- `https://github.com/eriklindernoren/PyTorch-YOLOv3`

3.2.7 Darkflow

Darkflow[8] implementiert die Darknet Library[9] in TensorFlow. Der Vorteil dabei, im Vergleich zur originalen Darknet Library, ist, dass man eine Python API statt dem Command Line Interface zu Verfügung hat.

Die Library wurde allerdings seit fast einem Jahr nicht mehr weiterentwickelt und weil wir möchten, dass unsere Applikation wartbar bleibt, wollen wir keine Libraries verwenden, welche nicht mehr unterhalten werden. Darkflow werden wir deshalb nicht weiter berücksichtigen.

3.2.8 ImageAI

ImageAI[25] ist ein Python Computer Vision Projekt und basiert auf dem Machine Learning Framework TensorFlow. Die Softwarebibliothek bietet eine einfache Schnittstelle für Image Recoginition und Object Detection für Bilder und Videos an.

ImageAI ist auf Windows, Linux und macOS lauffähig, aber nicht auf Smartphones. Durch die Abhängigkeit auf TensorFlow unterstützt ImageAI die Ausführung auf Grafikprozessoren. Wie bei TensorFlow ist dafür jedoch die komplexe Installation von CUDA nötig.

Nebst der Unterstützung von GPU bietet ImageAI einige Parameter an, um die Performance der Analyse zu verändern. Man kann das Intervall der Frames, welche analysiert werden, verändern und so zum Beispiel nur jedes zweite Frame anschauen. Dies erhöht die Geschwindigkeit, aber reduziert die Genauigkeit. Einige Messergebnisse und Videos findet man im ImageAI Repository[22].

Es existiert eine verständliche API, um ein Bild oder Video zu analysieren. Als Output erhält man ein Bild bzw. Video, in welchem die Objekte markiert wurden. Während der Analyse kann man pro Frame, Sekunde oder Minute auch eine Liste von gefunden Objekte erhalten.

Zwar ist ImageAI ein Open Source Projekt, jedoch ist der Hauptentwickler zur Zeit mit einem anderen Projekt beschäftigt und es gibt teilweise offene Issues und Pull Requests, welche nicht bearbeitet werden. Des Weiteren ist die Dokumentation teilweise lückenhaft. Ein Beispiel dafür ist, dass keine Anleitung für das Erstellen von eigene Modelle existiert.

Es ist grundsätzlich möglich Modelle, welche in Darknet erstellt wurden, zu konvertieren. Jedoch kann man nur umständlich eigene Klassen definieren, da ImageAI auf die Kategorien vom COCO Dataset[5] ausgelegt wurde und der Quellcode angepasst werden müsste um eigene Kategorien zu verwenden.

3.2.9 Apple Vision Framework

Das Vision Framework[47] von Apple unterstützt diverse Computer Vision Algorithmen. Für uns relevant ist dabei Object Detection und vielleicht noch Object Tracking.

Intern benutzt das Vision Framework Apples Machine Learning Library Core ML[2]. Core ML wurde für die lokale Ausführung von Machine Learning Modellen auf macOS und iOS Geräten entwickelt. Auf macOS wird die Ausführung auf GPUs unterstützt und auf iOS kann von der schnellen Neural Engine der A11 und A12 Chips Gebrauch gemacht werden.

Das Vision Framework gehörte zu den schnellsten Libraries, welche wir getestet haben, vermutlich lässt sich das auf die gute Hardwareoptimierung zurückführen.

Die Core ML Modelle[51], aufgrund deren das Vision Framework Objekte erkennt, können fertig heruntergeladen oder selber trainiert werden. Zusätzlich können auch Modelle von anderen Libraries wie Keras oder Caffe zu Core ML Modellen konvertiert werden[7].

Zum Trainieren eigener Modelle stellt Apple das Python Utility Turi Create[44] zur Verfügung. Turi Create läuft nicht nur auf macOS sondern auch auf Linux und Windows 10 (mit WSL). Leider bietet WSL noch keinen Support für CUDA, wodurch man auf Windows nicht mit GPU Unterstützung trainieren kann. Turi Create unterstützt Transfer Learning, mehr dazu im Abschnitt 4.2.

Core ML und das Vision Framework gehören zu der Standardbibliothek für macOS und iOS und werden darum mit der Xcode Installation ausgeliefert. Von allen Libraries, welche wir evaluiert haben, war die Installation somit mit Abstand am einfachsten. Sogar der Einsatz der GPU funktioniert bei Core ML ohne zusätzliche Konfiguration. Dank der Tatsache, dass Core ML und das Vision Framework Teil der Standardbibliothek sind, gestaltet sich auch die Paketierung in einer fertigen App äusserst simpel.

Selbstverständlich gibt es auch einige Negativpunkte. Der Grösste davon ist wohl, dass man damit an Apple Hardware gebunden ist. Allerdings betrifft uns das nicht, da, mit Ausnahme des Android Smartphones, alle in der Aufgabenstellung gelisteten Geräte von Apple produziert werden und somit das Vision Framework unterstützen. Im Gegensatz zu allen anderen erwähnten Libraries, sind Vision und Core ML nicht Open Source.

3.2.10 Fazit

Nachdem wir alle Frameworks genauer angeschaut haben, haben wir uns entschieden, das Apple Vision Framework zu verwenden.

Trotz der Limitierung auf Applegeräte, ist Vision immer noch das Framework mit der einfachsten Installation und gehört gleichzeitig zu den schnellsten, die wir getestet haben. Zusätzlich ist es sehr einfach, die Modelle in eine macOS oder iOS App zu integrieren.

Bei vielen der anderen Frameworks haben wir das Gefühl, dass diese nicht ausgereift bzw. flexibel genug sind, um sie einsetzten zu können. Ebenfalls sind die meisten Libraries für eine Serverseitige Analyse ausgelegt und nicht für die Object Detection auf Geräten von Endbenutzern gedacht.

3.3 Fazit Machbarkeitsstudie

Unabhängig von den angeschauten Libraries, haben wir auch überprüft, ob eine Videoanalyse schnell und zuverlässig genug durchgeführt werden kann, sodass die Bergretter entlastet werden können.

Wir vertreten die Meinung, dass das grundsätzlich möglich ist, aber man muss sich über einige Punkte im Klaren sein:

Einerseits ist das Resultat einer Deep Learning Applikation sehr stark von den Trainingsdaten abhängig. Falls nicht genügend Daten vorhanden sind, kann das Modell nicht lernen die gesuchten Objekte zu erkennen oder wird Overfitted auf die Trainingsdaten und kennt nur Objekte in Bildern, mit denen es trainiert wurde.

Die Performance kann auf verschiedene Arten optimiert werden, allerdings könnten diese Optimierungen das Resultat negativ beeinflussen. Ein Beispiel dafür ist es, nicht jedes Frame zu analysieren. Dadurch würde die Applikation definitiv schneller laufen, aber es könnte zufälligerweise ein Objekt nur in einem der nicht angeschauten Frames sichtbar sein.

Ebenfalls könnte es sein, dass die Applikation nichts findet. Dann muss man sich überlegen, ob man auf das Resultat vertrauen will oder ob Menschen das Video noch genauer anschauen sollten. Dieser Fall könnte auch bei einer nicht automatisierten Analyse auftreten.

Trotz dieser Einschränkungen, fällt das Ergebnis unserer Machbarkeitsanalyse positiv aus. Die umständliche Analyse wird von einer Applikation unterstützt und teilweise abgenommen. Zusätzlich könnte man die Position der Kamera (Koordinaten der Drohne) anzeigen damit die Rettern wissen, wo ein erkanntes Objekt lokalisiert ist.

Kapitel 4

Modelle trainieren

4.1 Vortrainierte Modelle

Für viele Algorithmen bzw. Frameworks gibt es bereits trainierte Modelle. Diese Modelle können viele unterschiedliche Objekte erkennen und sind mit einer immensen Anzahl Bildern trainiert worden.

Oft können diese Modelle hunderte Objektarten unterscheiden. Momentan sind wir allerdings nur an wenigen Unterscheidungen interessiert. Je mehr Klassen von einem Modell unterschieden werden können, desto komplexer und dementsprechend langsamer wird es.

Ein weiterer Nachteil bei der Verwendung von fertigen Modellen ist, dass diese meistens nicht auf die Erkennung von Objekten aus der Vogelperspektive optimiert sind, sondern mehrheitlich mit Bildern, welche aus Augenhöhe aufgenommenen wurden, trainiert wurden.

4.2 Transfer Learning

Mit Transfer Learning[28] kann man die Vorteile von vortrainierten Modellen und eigenen Modellen verbinden. Dabei nimmt man ein existierendes Modell als Basis für das Trainieren des eigenen Modelles.

Durch das Entfernen des Output Layers im neuronalen Netzwerk, muss das Basisodell und das eigene Modell nicht die gleiche Anzahl Kategorien unterstützen.

Als Basismodell nimmt man beispielsweise eines, welches mit dem COCO[5] Datensatz trainiert wurde. Dieser Datensatz besteht aus über 200'000 Bildern mit je bis zu fünf Labeln in fast 200 Kategorien. Für viele Libraries wie Darknet oder Apples Turi Create[44], werden solche Basismodelle von den Entwicklern zur Verfügung gestellt.

Dank dieser guten Basis, können eigene Modelle mit viel weniger Trainingsdaten und dementsprechend kleinerem Rechenaufwand trainiert werden. Laut Apple können, dank Transfer Learning, brauchbare Object Detection Modelle in Turi Create bereits mit nur 30 Bildern pro Klasse trainiert werden.[42]

Transfer Learning bietet also eine gute Kombination der Vorteile vortrainierter und eigener Modelle.

4.3 Trainingsdaten

Bevor eigene Modelle erstellt werden können, muss definiert werden, welche Problemstellung gelöst werden soll. In unserem Fall also das finden von Personen und Objekte, welche Hinweise auf den Verbleib von Personen geben könnten, in alpinem Gelände.

Bei den Objekten beschränken wir uns in dieser Arbeit auf Taschen, insbesondere Rucksäcke, und Kleidungsstücke.

Damit man ein vernünftiges Modell erstellen kann, benötigt man genügend Trainingsdaten, welche so realitätsnah wie möglich sind. Bevor wir mit dem Datensammeln angefangen haben, haben wir darum Szenarien aufgelistet, welche unsere Daten abdecken sollten:

Gegenstände bzw. Personen

- Person stehend
- Person liegend
- Person mit Rucksack
- Person ohne Rucksack
- Gegenstände am Boden

Anforderungen

- Verschiedene Untergründe (Schnee, Wiese etc.)
- Unterschiedliche Lichtverhältnisse (Schatten, Sonne, bewölkt, Neben etc.)
- Unterschiedliche Winkel
- Unterschiedliche Flughöhen
- Gegenstände teilweise verdeckt (beispielsweise von Schnee)

Anschliessend haben wir ein geeignetes Gebiet aufgesucht und Personen, Rucksäcke und Kleidungsstücke darin platziert. Die Drohne hat das Gebiet mehrfach abgeflogen und dabei Videos aufgenommen.

Aus zeitlicher Gründen und Wetterabhängigkeit, konnten wir nicht alle gewünschten Fälle abdecken. Beispielsweise haben wir keine Testdaten im Geröll

Abbildung 4.1: Beispielbild mit markierten Objekten

aufgenommen, da die meisten Geröllfelder während der Zeit der Arbeit noch eingeschneit waren.

Zusätzlich zu den eigenen Aufnahmen haben wir auch Videos von Videoplattformen gesammelt, um einige der Szenarien besser abdecken zu können.

So konnten wir eine solide Grundlage an Daten sammeln und hatten am Ende der Arbeit ein Trainingsset mit ca. 600 Bildern und fast 2000 markierten Objekten.

4.4 Kennzeichnung

Nachdem man die Bilder gesammelt hat, kann man diese für das Lernen vorbereiten. Das heisst, in den einzelnen Frames die gesuchten Objekte zu lokalisieren und zu kategorisieren.

Am besten verwendet man für diesen Schritt ein Tool. Wir haben dafür das Visual Object Tagging Tool (VoTT)[48] von Microsoft verwendet.

4.5 Data Augmentation

Oft ist das Sammeln von Testdaten zeitaufwendig und teuer. Daher ist man eingeschränkt, genügend Daten zu erhalten. Data Augmentation bietet die Mög-

lichkeit, aus bestehenden Trainingsdaten weitere Trainingsdaten zu generieren, ohne zeitaufwendige Messungen durchzuführen. Bei Bildern ist das zum Beispiel durch skalieren, rotieren oder durch das Hinzufügen von Rauschen möglich.

Diese veränderten Trainingsdaten können das Modell zum Beispiel so beeinflussen, dass Objekte nicht nur in einer bestimmten Ausrichtung oder Grösse erkannt werden und dadurch flexibler wird.

Verschiedene Libraries, wie zum Beispiel das von uns eingesetzte Trainingstool Turi Create, führen von sich aus Data Augmentation aus[16].

Man kann auch zusätzliche Veränderungen an den Bildern durchführen und zum Beispiel einzelne Gegenstände anders einfärben.

Da wir andere Aufgaben höher priorisiert haben, werden wir an dieser Stelle nicht weiter auf diese Möglichkeit eingehen, sie könnte aber in einer Folgearbeit weiterverfolgt werden.

4.6 Trainings- und Testdaten

Die gesammelten Daten haben wir in ein Testset und ein Trainingsset aufgeteilt. Das Testset besteht aus ca. 65 Bildern und das Trainingsset aus rund 600 Bildern.

Bei der Aufteilung haben wir darauf geachtet, dass wir keine Frames aus dem gleichen Video im Testset und im Trainingsset verwenden, sondern alle Frames aus einem Video entweder für das Eine oder das Andere verwendet werden.

Der Grund dafür ist, dass Frames aus dem gleichen Video oft ähnlich aussehen, zwei aufeinanderfolgende Frames sehen beispielsweise fast identisch aus. Somit hätte das einen ähnlichen Effekt, wie das selbe Frame im Trainingsset und im Testset zu verwenden.

4.7 Turi Create Performance

Um einschätzen zu können, wie lange das Training mit Turi Create bei unterschiedlicher Anzahl Trainingsbildern dauert, haben wir das Modell mit unterschiedlicher Anzahl Trainingsdaten trainiert und die Zeit gemessen, welche dafür benötigt wurde. Damit die Messungen vergleichbar sind, wurden alle auf der gleichen Linux Workstation, mit einer Nvidia GTX 1060 Grafikkarte mit 6GB Grafikspeicher, ausgeführt.

Bei den Messungen haben wir Turi Create die beiden Parameter `batch_size` und `max_iterations` selbst wählen lassen. Die Messresultate sind im Anhang B aufgelistet.

4.7.1 Vergleich Anzahl Trainingsbilder / Trainingsdauer

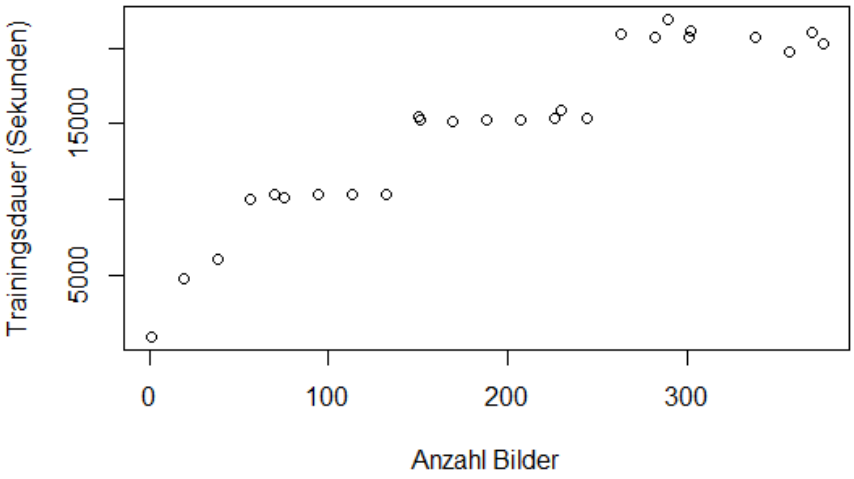

Abbildung 4.2: Vergleich zwischen Anzahl Trainingsbildern und resultierender Trainingsdauer mit den Daten aus Tabelle B.1

Um den Zusammenhang zwischen der Anzahl Objekte und der Trainingsdauer darzustellen, haben wir es in der Abbildung 4.2 geplottet. Dabei ist klar zu erkennen, dass die Trainingsdauer sprunghaft ansteigt.

4.7.2 Vergleich Anzahl Objekte / Trainingsdauer

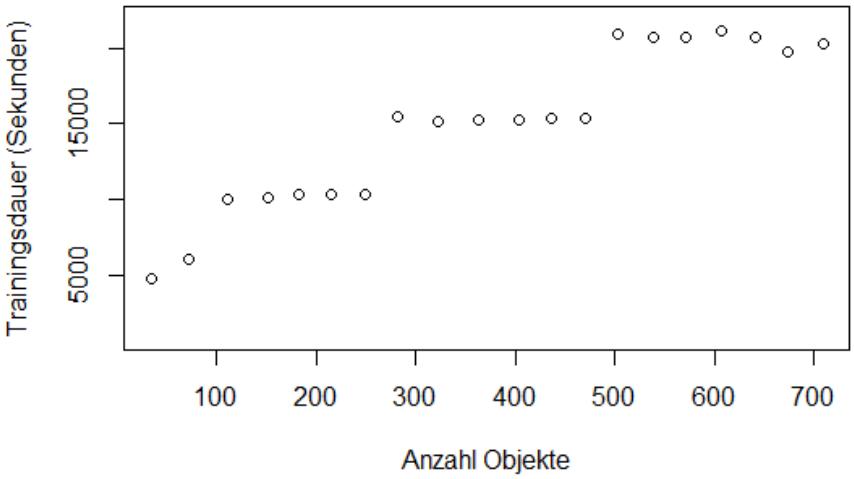

Abbildung 4.3: Vergleich zwischen Anzahl Objekten in den Bilder und daraus resultierender Trainingsdauer mit den Daten aus Tabelle B.1

Um zu erkennen ob die Anzahl Bilder oder die Anzahl markierter Objekte in den Bildern ausschlaggebender für die Trainingsdauer ist, haben wir in Abbildung 4.3 den Zusammenhang zwischen der Anzahl Objekten und der Trainingsdauer geplottet.

Dabei fällt auf, dass es weniger Ausreisser gibt als in der Abbildung 4.2. Warum das so ist, wird im Abschnitt 4.7.4 erklärt.

4.7.3 Vergleich Anzahl Iterationen / Trainingsdauer

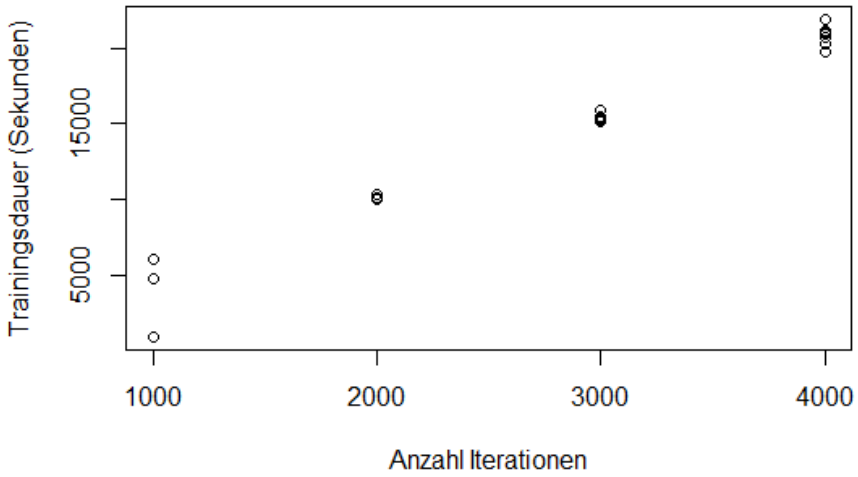

Abbildung 4.4: Vergleich zwischen Anzahl Iterationen und daraus resultierender Trainingsdauer mit den Daten aus Tabelle B.1

Zum Schluss haben wir auf Abbildung 4.4 noch den Zusammenhang zwischen der Anzahl an Iterationen und der Trainingsdauer verglichen. Dabei fällt eine hohe Korrelation von 0.99 auf. Alle Trainingsdurchläufe mit ähnlicher Anzahl Iterationen haben sehr ähnliche Trainingsdauern. Im Folgenden werden wir darum analysieren, wie Turi Create die Anzahl Iterationen berechnet.

4.7.4 Anzahl Iterationen in Turi Create

Um herauszufinden, wie Turi Create berechnet, wie viele Iterationen durchgeführt werden sollen, haben wir uns den Source Code angeschaut.

In der Python Datei object_detector.py[1] haben wir folgendes Code Snippet gefunden:

```
# Set number of iterations through a heuristic
num_iterations_raw = 5000 * _np.sqrt(num_instances) / batch_size
num_iterations = 1000 * max(1, int(round(num_iterations_raw / 1000)))
```

Die Anzahl Iterationen hängt also von den beiden Parametern `num_instances` und `batch_size` ab.

Die Variable `num_instances` entspricht dabei der Anzahl von Boxen oder Labels in den Trainingsdaten. Sie ist also nicht direkt von der Anzahl Bildern abhän-

[1] `https://github.com/apple/turicreate/blob/master/src/unity/python/turicreate/toolkits/object_detector/object_detector.py`

gig, sondern von der Anzahl Boxen in allen Bildern. Das erklärt auch warum Abbildung 4.3 weniger Ausreisser hat als 4.2.

Die `batch_size` Variable steht für die Anzahl Bilder, mit denen während einer Iteration trainiert wird. Sie kann vom Benutzer über einen Parameter konfiguriert werden, wird von Turicreate aber limitiert, falls sie zu gross wäre für den vorhandenen Grafikkartenspeicher. Standardmässig wird `batch_size` auf 32 gesetzt, wenn mehr als 4GB Grafikkartenspeicher vorhanden sind, wie das bei uns der Fall ist.

Im folgenden ist der Python Code von Oben noch als Formel dargestellt:

$$num\_iterations\_raw = 5000 * \frac{\sqrt{num\_instances}}{batch\_size}$$

$$num\_iterations = 1000 * max(1, round(\frac{1}{1000} * num\_iterations\_raw)) \quad \Longleftrightarrow$$
$$num\_iterations = 1000 * max(1, round(\frac{1}{1000} * 5000 * \frac{\sqrt{num\_instances}}{batch\_size})) \quad \Longleftrightarrow$$
$$num\_iterations = 1000 * max(1, round(5 * \frac{\sqrt{num\_instances}}{batch\_size}))$$

Durch die $round()$ Funktion ist der Anstieg an Iterationen nicht stetig, sondern sprunghaft. Die $max()$ Funktion bewirkt, dass mindestens 1000 Iterationen durchgeführt werden und die Multiplikation mit 1000 am Anfang erreicht, dass die sprunghafte Erhöhung jeweils in 1000er Schritten durchgeführt wird.

Diese sprunghafte Erhöhung ist auch klar in den Plots aus den vorhergehenden Abschnitten ersichtlich.

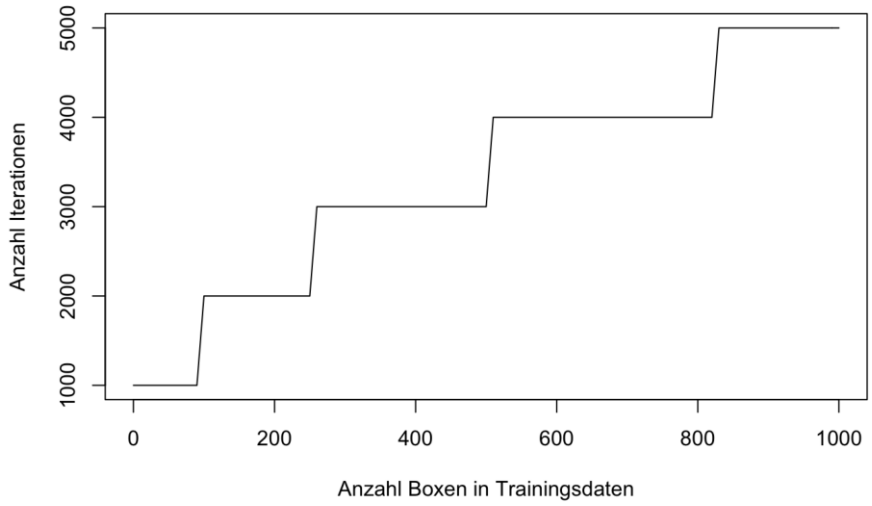

Abbildung 4.5: Plot für Formel für $num\_iterations$ mit fixem Wert für $batch\_size$ von 32 und Werten von 0 bis 1000 für $num\_instances$.

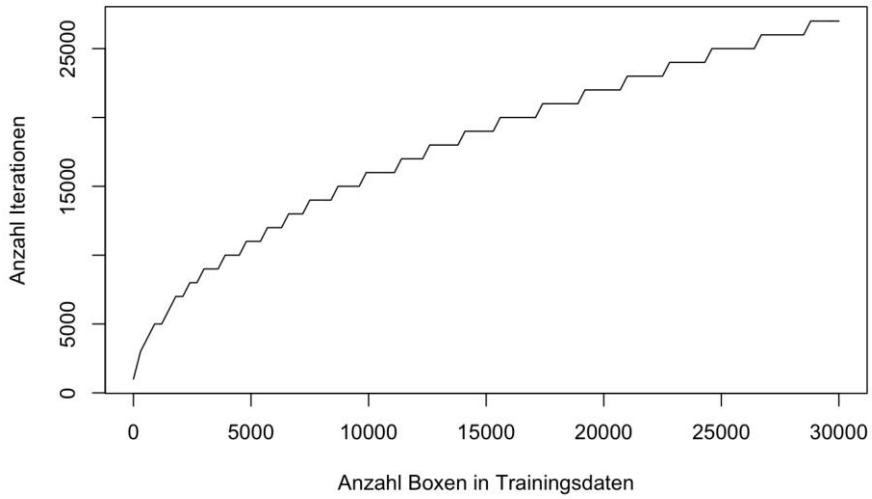

Abbildung 4.6: Plot für Formel für *num_iterations* mit fixem Wert für *batch_size* von 32 und Werten von 0 bis 30000 für *num_instances*. Abbildung 4.5 ist ein kleiner Ausschnitt aus diesem Plot.

Besonders aus Abbildung 4.6 wird ersichtlich, dass sich die Anzahl Iterationen aufgrund der Anzahl an Boxen in den Trainingsdaten wie die Kurve einer Quadratwurzel entwickelt.

Für uns ist dieses Resultat positiv, denn es zeigt, dass die Skallierbarkeit besser ist, als das bei einer linearen oder gar exponentiellen Entwicklung der Fall wäre.

4.7.5 Einfluss von Batch Size auf die Anzahl Iterationen

Bis jetzt haben wir den Parameter `batch_size` auf dem Standardwert von 32 belassen. Hier wollen wir jetzt untersuchen, wie sich die Veränderung von `batch_size` auf die Performance auswirkt.

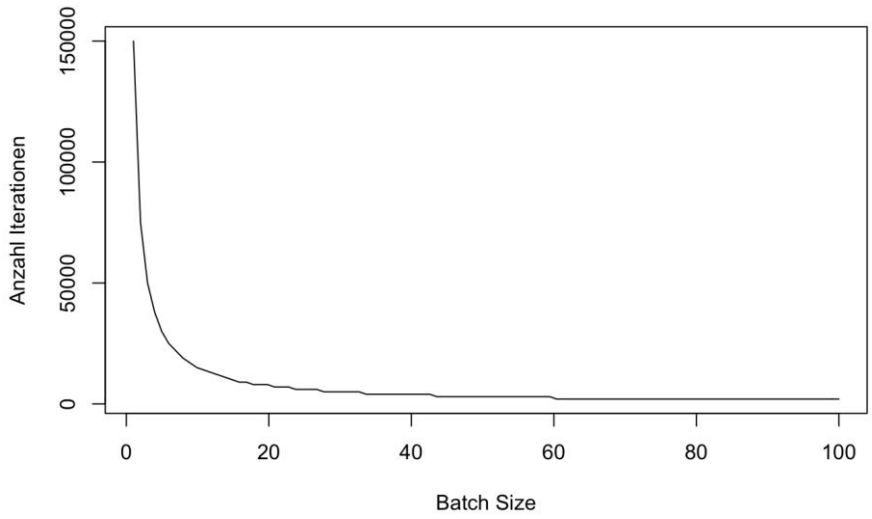

Abbildung 4.7: Plot für Formel für *num_iterations* mit fixem Wert für *num_instances* und Werten von 0 bis 1000 für *batch_size*.

Wie aus Abbildung 4.7 ersichtlich, reduziert sich die Anzahl Iterationen bis zu einer *batch_size* von ungefähr 20 sehr stark, flacht nachher allerdings schnell ab. Der Standardwert von 32 liegt bereits im relativ flachen Bereich der Funktion. Somit hätte eine weitere Erhöhung der *batch_size* nur noch einen kleinen Einfluss auf die Anzahl Iterationen.

4.8 Modell evaluieren

Um die Qualität des Modells zu bewerten, wird ein Testset mit Bildern verwendet.

Bei der Auswertung der Resultate müssen verschiedene Kriterien beachtet werden. Es reicht nicht, wenn die Kategorie einer Beobachtung stimmt, sondern sie muss auch am richtigen Ort im Bild platziert werden.

4.8.1 Intersection over Union (IOU)

Dass die Box, welche das Object Detection Modell zurückgibt, genau am gleichen Ort ist, wie die Markierung im Bild aus dem Testset, ist sehr unwahrscheinlich. Darum braucht es etwas Toleranz bei der Bewertung, ob zwei Boxen äquivalent sind. Dafür wird oft ein variabler Grenzwert für die sogenannte „Intersection over Union“, kurz IOU, verwendet.[27]

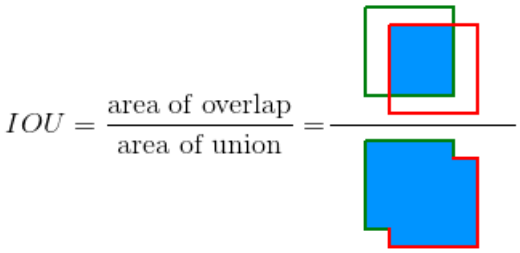

Abbildung 4.8: „Intersection over Union“, kurz IOU. IOU wird verwendet, um zu erkennen, ob zwei Boxen, das gleiche Objekt markieren. Bildquelle: [27]

Eine Beobachtung wird nur als richtig bewertet, wenn sowohl die Klassifizierung mit der Erwarteten übereinstimmt, als auch die IOU, der erwarteten und der beobachteten Box, den bestimmten Grenzwert nicht überschreitet.

Der Grenzwert für die IOU ist variabel und kann Werte zwischen 0 und 1 einnehmen. Wenn es sehr wichtig ist, dass die Erkennungen genau mit den Markierungen im Testset übereinstimmen, wählt man einen Wert näher bei 1 und wenn es keine grosse Rolle spielt, dass das Objekt exakt von der Box abgedeckt wird, wählt man einen Wert näher bei 0.

4.8.2 Fehlertypen

Falls nicht beide, der im vorherigen Abschnitt beschriebenen, Kriterien erfüllt sind, wird die Beobachtung als Falsch eingestuft. Dabei kann aber noch weiter zwischen unterschiedlichen Fehlertypen unterschieden werden:

False Negative Unter einem False Negative, verstehen wir den Fall, wenn im Testbild ein Objekt vorhanden ist, welches vom Modell aber nicht erkannt wurde.

False Positive False Positive ist das Gegenteil vom False Negative. Dabei wird vom Modell ein Objekt an einer Position erkannt, an der im Testbild nichts ist.

Falsche Klassifizierung Eine falsche Klassifizierung beschreibt den Fall, dass ein Objekt zwar korrekt vom Modell erkannt wurde, dem Objekt aber die falsche Kategorie zugeordnet wurde.

Abbildung 4.9: **Links**: Beispiel für False Positive. Hund wurde als Person und als Tasche erkannt, hätte aber nicht erkannt werden sollen, da keine Kategorie für Hund existiert. Bildquelle: Heinz Beutler **Mitte**: Beispiel für False Negative. Jacke wurde nicht gefunden. **Rechts**: Beispiel für falsche Klassifizierung. Jacke wurde als Tasche markiert.

4.8.3 Turi Create

Turi Create hat eine integrierte Funktion, um Modelle zu evaluieren, allerdings unterscheidet diese nicht zwischen den oben gelisteten Fehlertypen. Als Resultat erhält man verschiedene Variationen von `average_precision`.[43]

4.8.4 Eigene Modellbewertung

Für viele Anwendungen reicht die Evaluation von Turi Create bestimmt aus, in unserem Fall führen aber nicht alle Typen von Fehlern zu gleich grossen Konsequenzen.

Wenn beispielsweise ein Stein als Person erkannt wird, ist das viel weniger schlimm, als wenn eine Person nicht erkannt würde. Im ersten Fall würde der Retter sehr schnell erkennen, dass es sich dabei nicht um eine Person handelt. Im zweiten Fall könnte es aber sein, dass die Person, durch dieses False Negative, nicht gefunden wird.

Ähnlich verhält es sich dabei mit falschen Klassifizierungen. Wenn zum Beispiel eine Person als Kleidungsstück markiert werden würde, würden die Retter entweder schon beim genaueren anschauen des Videos erkennen, dass es sich dabei um eine Person handelt, oder spätestens dann, wenn man sich das angebliche Kleidungsstück aus der Nähe genauer anschauen würde.

Ebenfalls ist ein Objekt, welches nicht perfekt von der Markierungsbox abgedeckt wird, besser als das Objekt gar nicht zu sehen. Deshalb haben wir uns dafür entschieden, den IOU Threshold auf einen tiefen Wert von 0.2 zu setzen.

Für unsere Auswertung nach Fehlerkategorien, haben wir eine Python Funktion entwickelt, welche das Testset mit den erwarteten Objekten und die vom Modell

erkannten Objekte als Parameter entgegennimmt und zählt wie viele False Positives, False Negatives, falsche Klassifizierungen und korrekte Klassifizierungen vom Modell generiert wurden.

Die Anzahl der False Negatives zusammen mit der Anzahl an falschen und richtigen Klassifizierungen ist gleich gross wie die Anzahl erwarteter Objekte. Die Anzahl an False Positives, kann unabhängig davon variieren.

$$
\begin{aligned}
n_{FN} &= \text{Anzahl False Negatives} \\
n_{FK} &= \text{Anzahl falscher Klassifizierungen} \\
n_{KK} &= \text{Anzahl korrekter Klassifizierungen} \\
n &= \text{Anzahl erwarteter Objekte (aus Testset)} \\
n &= n_{FN} + n_{FK} + n_{KK}
\end{aligned}
$$

4.9 Vergleich Anzahl Kategorien

Um herauszufinden ob mehr Objekte erkannt werden, wenn man die Trainingsdaten in mehr Kategorien unterteilt oder ob man besser nur eine einzelne Kategorie für alles macht, haben wir drei verschiedene Modelle mit unterschiedlicher Anzahl Kategorien trainiert und anschliessend die Ergebnisse verglichen. Alle Modelle haben wir mit dem gleichen Trainingsset trainiert und mit dem gleichen Testset evaluiert. Die Ergebnisse dieser Evaluation sind in der Abbildung 4.10 ersichtlich.

Drei Kategorien Für das Modell mit drei Kategorien klassifizieren wir die Objekte als Person (person), Tasche (bag) oder Kleidungsstück (clothing).

Zwei Kategorien Für das Modell mit zwei Kategorien haben wir Tasche und Kleidungsstück zur neuen Kategorie Ausrüstungsgegenstand (gear) zusammengefasst. Für Personen haben wir die separate Kategorie beibehalten.

Eine Kategorie Für das Modell mit einer Kategorie unterscheiden wir nicht mehr zwischen Personen und Ausrüstungsgegenständen und haben alles einfach als Objekt gekennzeichnet.

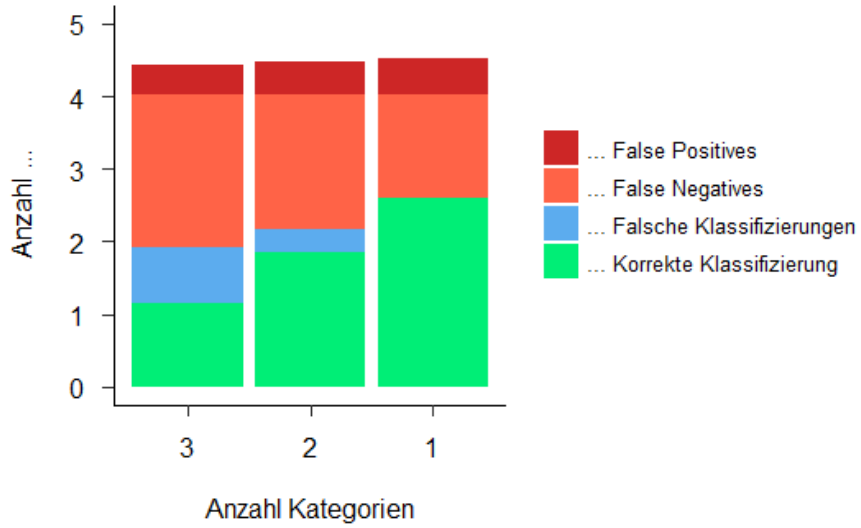

Abbildung 4.10: Vergleich verschiedener Fehlertypen, mit Modellen mit unterschiedlicher Anzahl an Kategorien. Die Y-Achse zeigt, wie viele der markierten Objekte, im Schnitt, pro Trainingsbild, welcher Kategorie zugeordnet werden können.

Wie man aus der Grafik erkennen kann, sind pro Bild aus dem Testset im Schnitt ca. vier Objekte markiert.

Natürlich sind beim Modell mit nur einer Kategorie keine falschen Klassifizierungen möglich. Aber auch wenn man die falschen Klassifizierungen der anderen Modelle ignoriert, werden beim Modell mit einer Kategorie immer noch am meisten Objekte erkannt. Allerdings geht das auf Kosten einer kleinen Erhöhung an False Positives.

Auffallend ist auch, wie viel mehr falsche Klassifizierungen das Modell mit drei Kategorien produziert, als das mit nur zwei. Wir vermuten, dass das daran liegt, dass Kleidungsstücke und Taschen auf diese Distanz zu schwierig zu unterscheiden sind.

Für die Suche wäre das Modell mit einer Klasse also am geeignetsten. Für die Abgabe und die Präsentation werden wir allerdings das Modell mit zwei Klassen verwenden, da man damit demonstrieren kann, dass zwischen verschiedenen Kategorien unterschieden werden kann.

Für den Einsatz während einer echten Suche wird dann voraussichtlich das Modell mit nur einer Kategorie verwendet.

4.10 Analyse in höherer Auflösung

Die Gegenstände auf den Testbilder sind teilweise sehr klein und dass kleine Objekte schwerer zu erkennen sind, ist eine bekannte Limitierung von YO-

LO[33].

Um das zu überprüfen, wurde die gleiche Messung, wie in Abbildung 4.10 noch einmal mit der doppelten Auflösung, der Testbilder durchgeführt. Dafür wurden die Testbilder alle halbiert und die beiden Hälften einzeln als Input für das Modell verwendet. Somit brauchte die Analyse doppelt so viele Modelldurchläufe. Das Resultat davon ist in Abbildung 4.11 zu sehen.

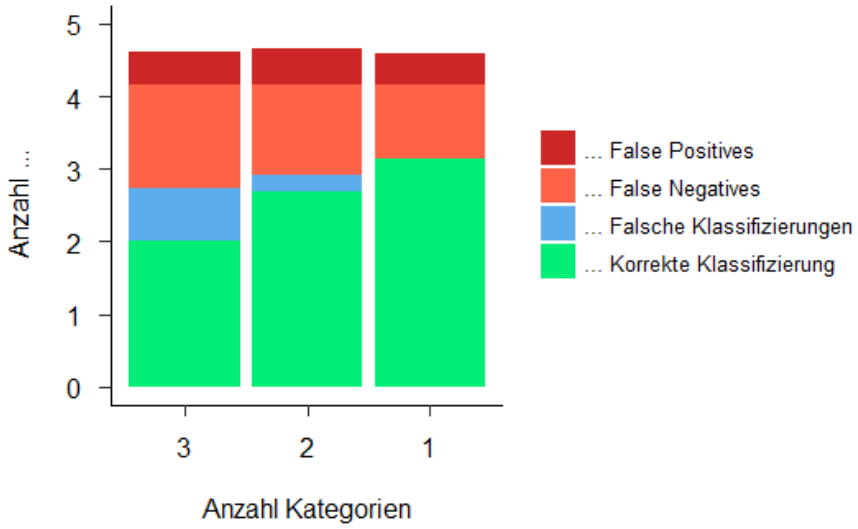

Abbildung 4.11: Wie Abbildung 4.10, aber Testbilder wurden in doppelter Auflösung analysiert.

Wie man leicht erkennen kann, wird mit der Analyse in doppelter Auflösung ein besseres Resultat erzielt.

Kapitel 5

Planung macOS App

5.1 Qualitätsmassnahmen Swift

Ergänzend zu den Qualitätsmassnahmen im Projektplan, führen wir hier noch die Massnahmen für den Swift Code auf.

5.1.1 Unit Tests

Zum Schreiben von Unit Tests verwenden wir das bewährte XCTest Framework[52]. Das Framework ist gut in XCode integriert und es wird eine Code Coverage Ansicht angeboten.

Wir werden keine Unit Tests für das Benutzerinterface erstellen, da sich die Oberfläche zu häufig ändert und wir bei jeder neuen Änderung die Tests neu schreiben müssten. Für den restlichen Code streben wir eine Test Coverage von mindestens 80% an.

5.1.2 Manuelle Tests

Da wir die Benutzeroberfläche nicht automatisch testen, decken wir das mit manuellen Tests ab. Dafür haben wir Testprotokolle definiert, welche vor jedem Release der Applikation durchgeführt werden sollen. Die Testprotokolle und Testergebnisse sind im Anhang A abgelegt.

5.1.3 Statische Code Analyse Tools

Neben den Unit Tests werden wir auch statische Code Analyse Tools verwenden, welche die Qualität des Codes verbessern und den Code vereinheitlichen sollen.

SwiftLint

SwiftLint ist ein Linter für Swift, welcher lose auf dem GitHub Style Guide beruht[30]. Dabei können wir die Regeln sehr gut auf unsere Bedürfnisse anpassen und zum Beispiel bestimmte kürzere Variablennamen zulassen.

SwiftLint werden wir in den Build Prozess integrieren, so dass mögliche Warnungen und Fehlermeldungen bei jedem Build angezeigt werden.

XCode Analyze

Zusätzlich zu SwiftLint nutzen wir die XCode Analyse Funktion, welche übliche Programmierfehler findet und dem Entwickler meldet.

5.2 User Interface Design

Die Applikation ist ein erweiterter Videoplayer, welcher nicht nur Videos abspielen kann, sondern gleich noch Objekte im Video erkennen und dem Benutzer anzeigen kann. Die gefunden Objekte werden umrahmt dargestellt.

Bei gewöhnlichen Videoplayern ist der Fortschrittsbalken dafür da, zu zeigen an welchem Zeitpunkt im Video man gerade steht, sowie zum Vor- und Zurückspulen des Videos.

In unserer Applikation wird der Fortschrittsbalken so erweitern, dass er zusätzlich als Indikator dafür verwendet werden kann, welche Frames des Videos Objekte enthalten, welche keine Objekte enthalten und welche noch nicht analysiert wurden.

Konkret wird diese Indikation mit unterschiedlichen Farben für die verschiedenen Kategorien von Frames umgesetzt. Jeder Kategorie wird eine Farbe zugeordnet, beispielsweise grau für Frames, welche noch nicht analysiert wurden, grün für Frames, welche analysiert wurden und Objekte enthalten und rot für Frames, welche zwar analysiert wurden, aber keine Objekte enthalten.

Natürlich werden Benutzer die Möglichkeit haben mittels Buttons das Abspielen des Videos zu pausieren bzw. fortzusetzen. Die Analyse wird unabhängig davon weiterlaufen.

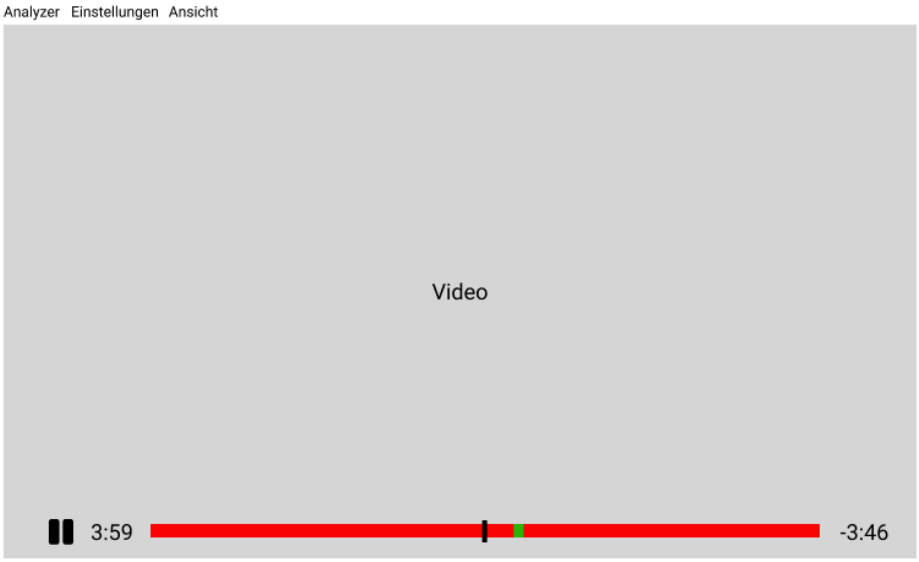

Abbildung 5.1: User Interface Design für Videoanalyse, ohne Kartenansicht. Zu beachten ist dabei der grüne Ausschnitt im Fortschrittsbalken, welcher Frames, welche Objekte enthalten markieren.

Über die Menuleiste kann ein Video für die Analyse ausgewählt werden, sowie das Einstellungsfenster geöffnet werden, in welchem man die Konfigurationsparameter ändern kann.

Alle Parameter, bei denen wir uns nicht sicher sind, welchen Wert wir dafür setzen sollen, werden wir konfigurierbar machen. Dadurch muss die Applikation beim Testen nicht neu kompiliert werden. Falls wir in Zukunft mehr Erfahrungswerte haben, können wir die Standardwerte auf diese setzen.

Falls für das Video eine Untertitel Datei im SRT Format existiert, welche die Koordinaten der Flugroute enthält, wird neben dem Video eine Karte eingeblendet, auf der die Route und die Position der Drohne an der aktuellen Abspielposition im Video angezeigt wird.

Es ist möglich, in der Kartenansicht zu navigieren und zu zoomen.

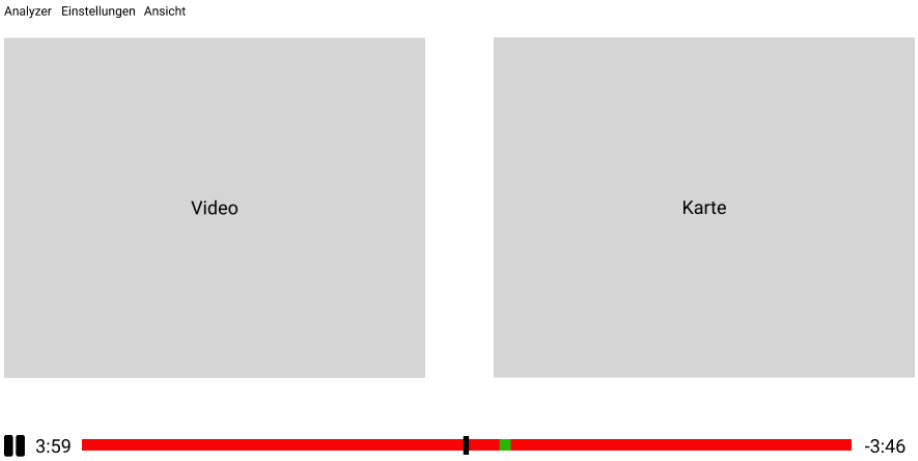

Abbildung 5.2: User Interface Design für Videoanalyse, mit Kartenansicht. Trotz der Karte, verläuft der Fortschrittsbalken über die ganze Breite des Fensters.

Kapitel 6

Integration in iOS App

Zusätzlich zur standalone macOS Applikation, haben wir die Object Detection auch in die bestehende App „TRS Rescuer“ integriert. In der Aufgabenstellung war diese Integration noch als Fernziel gelistet, aber dank dem raschen Vorwärtskommen, sind wir bereits während unserer Arbeit dazu gekommen.

6.1 Beschreibung TRS Rescuer iOS App

Die TRS Rescuer App wurde als Teil der Bachelorarbeit von David Riederer und Lukas Oberholzer im Frühlingssemester 2018 entwickelt. Die Applikation ermöglicht die Verbindung zwischen den Rettern im Feld und der Cockpit Applikation. Retter können über die App, die Ihnen zugewiesenen Suchaufträge anschauen und bearbeiten. Die App sendet die Position der Retter in regelmässigen Abständen an die Cockpit Applikation, um der Rettungsleitung einen Überblick über die laufenden Rettungsarbeiten zu geben.[10]

Ebenfalls Teil dieser Applikation ist die Steuerung der Drohne, einschliesslich automatisiertem Abfliegen von Flugrouten, welche zuvor in der Cockpit Web Applikation erfasst wurden. Dafür wurde das DJI Mobile SDK[12] eingesetzt, welches die Kommunikation zwischen App und Drohne ermöglicht. Damit die Steuerung funktioniert muss das iOS Gerät per Kabel mit der Fernsteuerung verbunden sein, welche wiederum kabellos mit der Drohne verbunden ist. [10]

Abbildung 6.1: Screenshot von Drohnen-Steuerungsansicht aus existierender TRS Rescue App.

6.2 Erweiterung mit Object Detection

Während dem Flug wird in der, vom DJI SDK zur Verfügung gestellten Ansicht, ein live Videostream von der Drohne angezeigt. Dieser Stream soll jetzt in Echtzeit nach Gegenständen durchsucht werden, welche Hinweise auf das Verbleiben einer vermissten Person geben könnte.

Da die Zeit, welche zur Analyse eines einzelnen Frames gebraucht ist, höher ist, als die Zeit in der ein einzelnes Frame angezeigt wird, kann nicht jedes Frame analysiert werden.

Es soll so implementiert werden, dass immer nur ein einzelnes Frame auf einmal analysiert wird. Sobald die Analyse beendet ist, wird das Resultat an die Benutzeroberfläche geschickt und das nächste Frame kann analysiert werden. Als nächstes Frame zur Analyse, soll immer das neuste Frame aus dem Stream gewählt werden. So können Verzögerung in der Anzeige möglichst klein gehalten werden.

Wie in der macOS Applikation, sollen gefundene Objekte mit Rechtecken umrahmt werden und das dazugehörige Label, sowie die Confidence angezeigt werden. Zusätzlich soll der Pilot mit einem Alarmton benachrichtigt werden, sobald ein Objekt erkannt wird.

6.3 Qualitätsmassnahmen

Soweit wie möglich, möchten wir die Qualitätsmassnahmen für Swift, welche wir für die macOS App im Abschnitt 5.1 beschrieben haben, auch für die iOS App umsetzen. Dabei sind wir allerdings ein wenig eingeschränkt, da wir an einer existierenden Applikation weiterarbeiten und nicht auf einer grünen Wiese beginnen. Unit Tests existieren beispielsweise nur wenige und keine für den Bereich, welcher für die Steuerung der Drohne zuständig ist.

Unser Ansatz, die Massnahmen trotzdem umsetzen zu können, ist, Funktionen, welche sowohl von der iOS App, als auch von der macOS Applikation benutzt werden können, in eine Core Library auszulagern, welche dann von beiden Anwendungen referenziert wird.

In der iOS App selbst, müssen wir so nur noch die Frames extrahieren, zur Analyse an eine Core Funktion weitergeben und das Resultat darstellen.

Diese Funktionalität werden wir durch Manuelle Tests abdecken. Siehe Abschnitt A.1 im Anhang.

Kapitel 7

Architektur

7.1 Komponenten

Wie bereits in den letzten beiden Kapiteln angetönt, gibt es drei Komponenten. Eine macOS App, eine iOS App und eine gemeinsame Core Library von welcher die Apps für die beiden Plattformen abhängig sind.

Das mit Turi Create trainierte Modell ist Teil der Core Library.

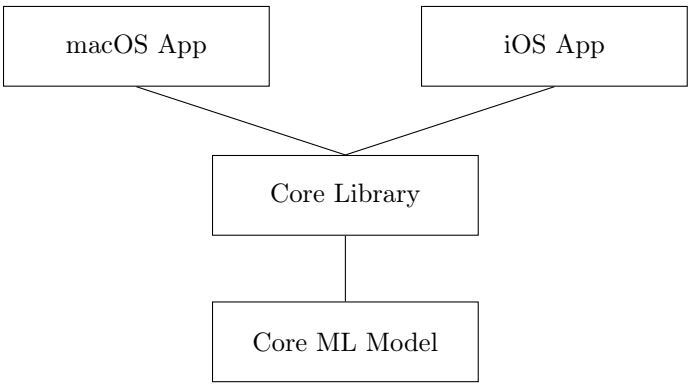

Abbildung 7.1: Komponentendiagramm für macOS App, iOS App und gemeinsamer Core Library, welche das Core ML Model enthält.

Kapitel 8

Implementation

In diesem Kapitel wird die Implementation beschrieben. Dabei beschränken wir uns auf die Beschreibung der Entwicklung der macOS App, sowie der iOS App. Das Trainieren des Modells ist im Kapitel 4 beschrieben.

Dazu gehören eine Auflistung der 3rd Party Libraries und Tools, welche wir verwendet haben, sowie eine Beschreibung zweier Herausforderungen, welche wir während der Implementation zu bewältigen hatten.

8.1 Dependency Management

Als Dependency Manager haben wir CocoaPods[6] verwendet. Alternativen dazu wären der Swift Package Manager[39] oder Carthage[4], aber da bei der existierende iOS App bereits CocoaPods eingesetzt wird, haben wir uns entschieden, das ebenfalls zu verwenden.

Die beiden anderen Package Managers haben wir kurz ausprobiert, aber sie scheinen noch nicht gleich reif zu sein wie CocoaPods. Carthage lädt Dependencies nur herunter, aber sie müssen dann trotzdem noch manuell zum Projekt hinzugefügt werden und Swift Package Manager unterstützt iOS und macOS Applikationen noch nicht offiziell, es ist eher für Swift Anwendungen für Linux konzipiert.

8.2 Libraries

Für die iOS und macOS Entwicklung stehen diverse 3rd Party Open Source Libraries zur Verfügung. Im folgenden möchten wir darauf eingehen, welche wir verwendet haben und warum. Die meisten Libraries haben wir sowohl in der iOS App, als auch in der macOS App eingesetzt. Für iOS kommen allerdings noch die Libraries vom DJI SDK hinzu.

8.2.1 In iOS und macOS App verwendet

SnapKit SnapKit[37] vereinfacht die Verwendung vom programmatischen Constraint Layout, indem es eine simple und leserliche DSL dafür anbietet.

RxSwift und RxCocoa RxSwift[36] ist die Swift Version von ReactiveX. In unserer Applikation liess sich viel als Reactive Stream abbilden. Zum Beispiel haben wir das Video in der macOS Applikation als Reactive Stream von Bildern dargestellt. RxCocoa stellt die Möglichkeit zur Verfügung, UI Events wie Button Klicks, direkt als Reactive Stream zu erhalten, statt wie üblich in einer Callback Methode zu verarbeiten.

Mapbox Mapbox[23] wurde bereits in der bestehenden iOS App zur Darstellung von Karten verwendet. Der Vorteil dabei gegenüber Apple Maps ist, dass die Library auch topografische Karten mit Höhenlinien anzeigen kann. In der macOS App haben wir ebenfalls Mapbox für die Darstellung der Flugroute der Drohne verwendet.

8.2.2 Nur in iOS App verwendet

DJI SDK iOS Das DJI SDK[12] für iOS bietet die Grundfunktionalitäten zur Kontrolle von DJI Drohnen in eigenen iOS Apps. Die bestehende iOS App hat diese Library bereits verwendet.

DJI UXSDK iOS Das DJI UXSDK[14] für iOS bietet fertige Views, welche zur Kontrolle von DJI Drohnen verwendet werden können. Beispielsweise stellt es eine View zur Verfügung, welche fast gleich aussieht wie die Kontrolloberfläche aus der DJI Go App. Diese Ansicht wird in etwas modifizierter Form in der bestehenden App bereits angezeigt.

DJIWidget Neu hinzugefügt haben wir eine Abhängigkeit zu der DJIWidget Library[15], welche den VideoPreviewer zur Verfügung stellt, den wir verwenden um einzelne Frames aus dem Videostream zu extrahieren.

8.3 Herausforderungen

Im folgenden sind zwei interessante Herausforderungen aufgelistet, welche wir während der Implementation zu bewältigen hatten. Damit möchten wir einen Eindruck vermitteln, was uns während dieser Phase besonders beschäftigt hat.

8.3.1 Bildartefakte von Videopreviewer

Als wir den Videopreviewer zum extrahieren von Frames aus dem Videostream der Drohne, eingebunden haben, hat er immer Bilder mit seltsamen Bildartefakten zurückgeliefert. Ein Beispiel dafür ist auf Abbildung 8.1 zu sehen.

Abbildung 8.1: Beispielbild mit Bildartefakten von Videopreviewer

Anfangs wussten wir noch gar nicht, dass die Frames diese Artefakte enthalten und die Object Detection deshalb nichts Sinnvolles finden konnte. Zuerst vermuteten wir, dass das Vision Framework, Bilder im YUV Format, welches vom Previewer verwendet wird, nicht richtig analysieren kann. Das haben wir dann allerdings in einem Swift Playground getestet und es hat problemlos funktioniert. Diesen Grund konnten wir also ausschliessen.

Um zu testen, ob es grundsätzlich möglich ist, Frames aus einem live Videostream zu analysieren, haben wir eine Ansicht erstellt, welche wir „Objekterkennung Demo“ nennen. In diesem Demomodus wird statt dem Stream von der Drohne, ein Videostream der iPhone oder iPad Kamera als Input für die Analyse verwendet. Mit dem Stream der Kamera, hat die Analyse problemlos funktioniert und uns so gezeigt, dass es möglich ist einen Stream in Echtzeit auf einem iPhone zu analysieren. Die Analyse ist sogar schneller als auf einem Macbook Pro (2014). Den Demomodus haben wir in der App belassen, er kann via Einstellungen aktiviert werden und anschliessend vom Menu aus gestartet werden.

Erst als wir die Frames, welche vom Previewer geliefert wurden, in die Camera Roll exportiert haben und so genauer anschauen konnten, stellten wir fest, dass sie diese Bildartefakte enthalten. Das seltsame dabei war, dass die Videovorschau, welche in der App angezeigt wurde, absolut fehlerfrei war. Es konnte sich also nicht um einen Verbindungsfehler zwischen der Drohne und dem iOS Gerät handeln.

Die Lösung brachte schliesslich ein Update des DJI SDKs von Version 4.5 auf Version 4.10, welche während unserer Arbeit erschienen ist. Nach dem Update wurden die Bilder ohne die Artefakte zurückgegeben, wir können also vermuten, dass es sich dabei um einen Fehler im DJI SDK gehandelt hat.

8.3.2 Ungeglättete Flugroute

Neben dem Videofile, erstellen DJI Drohnen während dem Flug auch noch ein dazugehörendes Untertitelfile im SRT Format. Unter anderem enthält dieses File, die Koordinate der Drohne zu verschiedenen Timestamps im Video. Wir haben die Koordinaten aus dem File extrahiert und als Pfad in einer Karte in der macOS App dargestellt.

Den Auftraggebern ist dann allerdings aufgefallen, dass diese Karte viel eckiger ist, als wenn sie das gleiche SRT File in der Web App unter `https://tailorandwayne.com/dji-srt-viewer/` anschauen, welche ebenfalls die Darstellung von DJI SRT Files erlaubt.

Standardmässig wird die Route in der Web App geglättet. Wenn man den Glättungsslider ganz nach links bewegt, sieht die Route auch dort gleich aus wie in unserer Applikation.

Glücklicherweise ist die Web Applikation von Taylor & Wayne ein Open Source Projekt und wir konnten nachschauen, wie sie die Glättung implementiert haben. Sie wählten dafür einen gleitenden Mittelwert, welchen wir anschliessend auch Implementiert haben. [13]

In der neuen Version der macOS App lässt sich der Grad der Glättung nun in den Einstellungen konfigurieren.

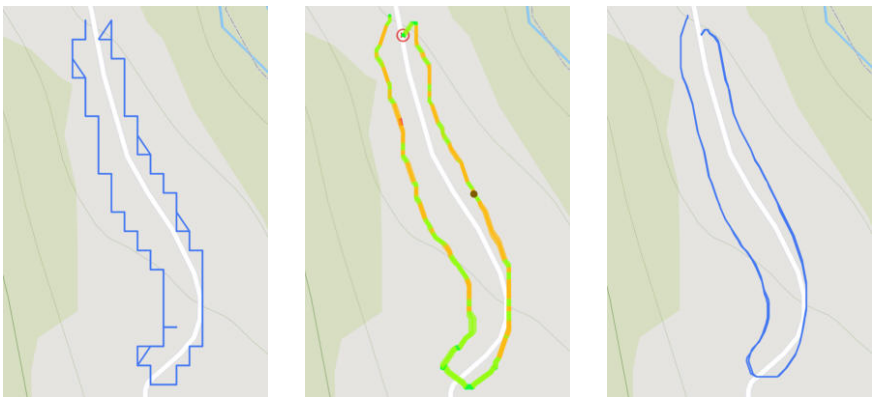

Abbildung 8.2: **Links**: Screenshot aus unserer macOS App, ohne Glättung. **Mitte**: Screenshot aus Web App von Taylor & Wayne, mit Glättung. **Rechts**: Screenshot aus unserer macOS App, mit Glättung

Kapitel 9

Mögliche Erweiterungen

In diesem Kapitel gehen wir nun auf mögliche Erweiterung ein, welche noch implementiert bzw. noch genauer erarbeitet werden könnten. Man könnte diese auch in einer weiteren Studien- oder Bachelorarbeit realisieren.

9.1 Konzept Feedback Loop

Während der Bachelorarbeit konnte, aus zeitlichen Gründen, nur eine limitierte Anzahl von Testdaten gesammelt und annotiert werden.

Optimal wäre, wenn die Trainingsdaten nicht ausschliesslich von den Entwicklern selbst gesammelt werden müssten, sondern dass die Benutzer der App, nach der Analyse, Feedback zum Resultat abgeben könnten.

Dabei sollten False Positives, False Negatives und falsche Klassifizierungen markiert und korrigiert werden können. Siehe Abschnitt 4.8.2 für eine Beschreibung der Fehlertypen.

True Positives, also richtig erkannte Objekte, sollten beibehalten werden, da sie das Training negativ beeinflussen würden, wenn sie nicht markiert wären.

Die Frames, mit den durch die Benutzer korrigierten Annotationen, sollten dann auf einer zentralen Plattform mit den Entwicklern geteilt werden.

So könnten die Bilder dann beim nächsten Trainingsdurchgang als Trainings- oder Testdaten verwendet werden.

Durch diese Feedbackfunktion wäre es uns möglich, sowohl quantitativ mehr, als auch realitätsnähere und dadurch qualitativ hochwertigere Trainingsdaten zu sammeln.

9.1.1 Feedback Plattform

Die Feedbacks, welche von den Benutzern zur Verfügung gestellt werden, werden auf einer zentralen Plattform gesammelt.

Die Sensitivität der Daten wurde bereits mehrmals angesprochen und müsste auch bei der Feedback Plattform beachtet werden. Konkret bedeutet das, dass die Bilder sicher vor Zugriffen Dritter geschützt werden müssen.

Entweder könnte diese Funktionalität in die bereits existierende Cockpit Web Applikation integriert werden oder man könnte eine separate Web Applikation dafür entwickeln.

Um zu vermeiden, dass das Model mit falschen Daten trainiert wird, sollen die Daten aus den Feedbacks erst zum Trainieren benutzt werden, nachdem sie von jemanden überprüft wurden.

9.1.2 Integration in Mac Applikation

Die Feedbackfunktion könnte in die neue Mac Applikationen integriert werden. Nach abgeschlossener Analyse soll ein Button angezeigt werden, um den Feedbackprozess zu starten.

Für das Feedback muss der Benutzer zuerst die Frames auswählen, für die er Korrekturen erstellen will. Anschliessend können Bounding Boxes von den Frames gelöscht werden, um False Positives zu korrigieren und neue Bounding Boxes gezeichnet werden, um False Negatives zu markieren.

Aufgrund der Sensitivität der Daten, muss der Benutzer vor Abgabe des Feedbacks bestätigen, dass er damit einverstanden ist, die Daten mit uns zu teilen.

Damit zugeordnet werden kann, welcher Benutzer, welches Bild zur Verfügung gestellt hat, sollte die Feedbackfunktion nur für eingeloggte Benutzer zugänglich sein.

9.1.3 Integration in Mobile Apps

Da die Videoanalyse in den Mobilen Apps in Echtzeit von einem Videostream gemacht wird und dieser Videostream nicht zwischengespeichert wird, müsste man während dem Flug die Möglichkeit haben Frames zu speichern, dass diese nach dem Flug hochgeladen werden könnten. Dies könnte man beispielsweise lösen, indem man einen Button auf dem Screen anzeigt, welcher dann bei einem Klick das aktuelle Frame zwischenspeichert und zur späteren Korrektur vormerkt.

Dass der Pilot die Feedback Funktion während eines richtigen Einsatzes verwendet ist eher unrealistisch, da dieser sich dabei auf die Rettung konzentrieren muss. Während eines Trainingseinsatzes ist es allerdings durchaus realistisch, dass Frames zur Korrektur vorgemerkt werden.

Sobald der Flug beendet ist und eine Internetverbindung vorhanden ist, kann man bei den vorgemerkten Frames die Annotationen korrigieren und die Frames hochladen. Das Einzeichnen der Boxen ist auf dem Mac jedoch genauer möglich als auf einem Touchscreen.

9.1.4 Integration in Feedback Plattform

Als Alternative zum Feedback über eine der nativen Apps, könnte man das markieren von Objekten in den Frames auch direkt in die Feedback Plattform integrieren. So wäre die Feedback Funktion plattformunabhängig und die Benutzer bräuchten nicht mehr zwingend einen Mac.

Dabei können wir uns zwei unterschiedliche Implementationsvarianten vorstellen. In beiden Varianten würden Benutzer zuerst ein Videofile oder Bilder auf die Plattform hochladen.

Variante 1 Bei der ersten Variante könnten Benutzer jetzt direkt mit dem Einzeichnen von Objekten beginnen. Im Gegensatz zu der Variante, welche für die Mac Applikation beschrieben ist, sehen die Benutzer in dieser Variante allerdings nicht, was von der Object Detection bereits erkannt worden wäre. Sie müssen also alle Objekte selber markieren.

Variante 2 Die zweite Variante löst genau dieses Problem und zwar indem, wie bei der Mac Applikation, zuerst die Object Detection ausgeführt wird, und die Benutzer anschliessend die Ergebnisse korrigiert. Das machen sie durch das Löschen falscher Markierungen, dem Einzeichnen fehlender Markierungen, sowie dem Korrigieren falscher Klassifizierungen.

Die Schwierigkeit in der zweiten Variante liegt darin, dass das Ausführen von Machine Learning Modellen innerhalb des Browsers wohl zu langsam sein wird. Es muss also auf dem Server erledigt werden, was bei einer grossen Anzahl paralleler Benutzern zu sehr grossen Hardwareanforderungen führen kann.

Durch eine Beschränkung der Object Detection auf die von den Benutzern ausgewählten Frames, könnte man die Anzahl nötigen Durchlaufe durch das neuronale Netzwerk extrem verringern. Wenn ein Benutzer in einem zehnminütigen Video mit 24 Bildern pro Sekunde, beispielsweise 50 Frames zum korrigieren auswählt, müssten nur 50 Frames statt $24\frac{\text{Frames}}{s} * 600s = 1440$ Frames durchsucht werden. Das ist eine enorme Einsparung.

9.1.5 Training

Mit den ausgewählten Daten kann nun ein Training durchgeführt werden. Das Training wurde wurde bereits im Abschnitt 4 beschrieben.

Es ist jedoch anzumerken, dass Turi Create zur Zeit kein Möglichkeit bietet, das Training basierend auf einem bestehenden Model auszuführen. Somit müsste das Model also jedes Mal von Grund auf neu trainiert werden, was eine längere Trainingszeit bedeutet.

Alternativ dazu könnte man das Modell auch mit Darknet oder TensorFlow trainieren und die Modelle anschliessend zu einem Core ML Modell für die Ausführung auf Apple Geräten konvertieren.

TensorFlow hätte dabei den Vorteil, dass es dafür ausgerichtet ist, auf mehreren Servern verteilt zu trainieren, was die Trainigszeit reduzieren könnte, allerdings würde das auch erhöhte Infrastrukturkosten mit sich bringen. [11]

Eine weitere Möglichkeit zum trainieren von Modellen wäre die Verwendung einer bestehenden Cloud Lösung. IBM bietet mit dem IBM Watson Visual Recognition Service [21] beispielsweise eine fertige Lösung, um Core ML Modelle in der Cloud zu trainieren, inklusive Anpassung des Modelles aufgrund von Feedbacks[19].

Momentan ist Object Detection für IBM Watson Visual Recognition noch in der Beta Phase[20], wird aber vielleicht zur Zeit der Implementation des hier beschriebenen Konzeptes verfügbar sein.

9.1.6 Ausgewogenheit der Daten

Damit ein Modell optimal trainiert werden kann, müssen die Kategorien eine ausgewogen Anzahl von Bildern haben, sodass keine Kategorie das Trainingsset dominiert.[24]

Wenn eine neue Kategorie hinzugefügt wird, bedeutet das also, dass idealerweise gleich viele Bilder der neuen Kategorie hinzugefügt werden müssen, wie von den bisherigen jeweils vorhanden sind. Ist das nicht möglich könnte man entweder die Bilder der anderen Kategorien reduzieren (Undersampling) oder man nimmt die Bilder der neuen Kategorie mehrfach ins Set auf (Oversampling). Hier gilt abzuwägen, ob man Oversampling oder Undersampling anwenden will oder noch warten kann bis genügend Bilder für die Kategorie verfügbar sind.

Ebenfalls benötigt man auch genügend Bilder, welche im Testset verwendet werden können.

9.1.7 Ressourcen Anforderung

Aufgrund des hohen Ressourcenverbrauchs sollte nicht jedes für jedes Feedback ein neuer Trainingsprozess gestartet werden. Sinnvoller ist es, die Feedbacks zu sammeln und den Prozess erst dann zu starten, wenn eine bestimmte Anzahl neuer Feedbacks erreicht ist.

9.1.8 Update

Über ein Updateprozess müssen die Nutzer nun in der Lage sein, das Verbesserte Modell herunterzuladen.

In unserem Fall ist das Modell in der iOS bzw. Mac Anwendung verpackt und kann dadurch über den Updateprozess dieser Plattformen verfügbar gemacht werden.

9.2 Nur Veränderungen analysieren

Da sich von einem Frame zum nächsten oft nur ein Teil des Bildes ändert, könnte man die Rechenpower während der Analyse auf diese Bereiche konzentrieren.

9.2.1 Bewegung zwischen Frames erkennen

In unserem Technologiestack könnte man mit einem VNTranslationalImageRegistrationRequest[49] Veränderungen (Translationen, Skalierungen und Rotationen) erkennen.

Bei voller Auflösung dauerte dieser Schritt ca. $100ms$[1]. Durch herunterskallieren des Bildes vor der Analyse der Veränderung, kann man diese Operation zwar enorm Beschleunigen, allerdings dauert das Herunterskallieren alleine schon länger als 100 Millisekunden, daher hilft uns das nicht weiter.

9.2.2 Performance

Ein Durchgang durch das neuronale Netzwerk für die Object Detection dauert immer noch gleich lange, aber man hat mehr Details in den Bildern und Objekte können daher genauer erkannt werden. Ebenfalls können mehr Objekte, die nahe bei einander sind, erkannt werden.

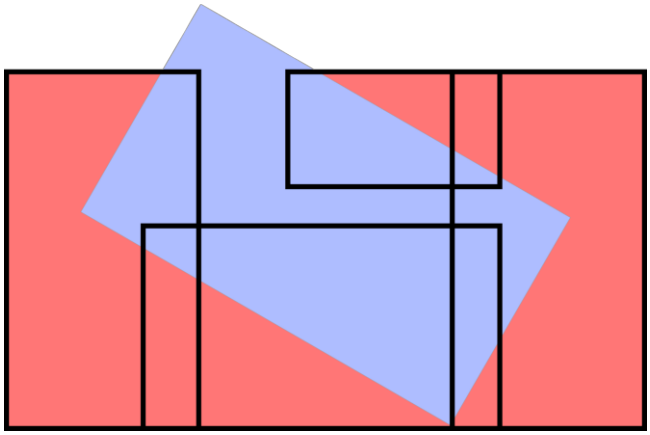

Abbildung 9.1: Beispiel für extreme Veränderung zwischen zwei Frames. Das blaue Rechteck steht für das vorheriges Frame, und das Rote für das neue Frame. Alle roten Bereiche, welche nicht vom alten, blauen Rechteck bereits analysiert wurden, müssen analysiert werden. Die schwarzen Rahmen zeigen, welche Regionen von der Object Detection angeschaut würden.

Ein Beispiel für eine sehr starke Veränderung zwischen zwei Frames, ist auf Abbildung 9.1 ersichtlich. Um in diesem Beispiel alle roten Regionen anzuschauen,

[1] Alle Messungen sind vom gleichen Macbook Pro aus 2014

welche nicht bereits im vorherigen Frame angeschaut wurden, müssen wir also vier Regionen anschauen (schwarz eingerahmt). Zusätzlich brauchen die Regionen, welche man anschauen möchte etwas Überlappung, damit Objekte an Regionsgrenzen ebenfalls erkannt werden können.

9.2.3 Worst Case

Wir wissen, dass ein Durchlauf durch das Object Detection Netzwerk etwa $150ms$ Dauert. Wenn man jetzt alles zusammenrechnet bräuchte man für die Erkennung jetzt also ungefähr $4 * 150ms + 100ms$ für die Erkennung der Verschiebung. Alles in allem braucht man im schlimmsten Fall also ca. $700ms$ pro Frame statt nur $150ms$.

9.2.4 Best Case

Im Besten Fall könnte man die gesamte Fläche, welche im vorherigen Frame nicht angeschaut wurde mit einem Rechteck abdecken. Man bräuchte so nur einen einzelnen Durchlauf durch das Object Detection Netzwerk. Dies resultiert in einer ungefähren Dauer für die Analyse eines Frames von $100ms + 150ms = 250ms$.

9.2.5 Durchschnitt

Wenn wir davon ausgehen, dass der Worst Case und der Best Case eher selten eintreten und die meisten Frames 2 bis 3 (Schnitt 2.5) Rechtecke brauchen. um alle Regionen abzudecken, dann dauert die Analyse pro Frame im Schnitt ca. $2.5 * 150ms + 100ms = 400ms$. Im Durchschnitt wären wir so also $250ms$ langsamer pro Frame, dafür könnten wir die Bilder in höherer Auflösung analysieren. Somit könnten statt ~ 6 Frames pro Sekunde, nur noch ~ 2.5 Frames pro Sekunde analysiert werden.

9.2.6 Risiken

- Berechnete Verschiebung stimmt nicht und gewisse Bildausschnitte werden nicht angeschaut
- Wenn zu viel Zeit zwischen zwei Frames verstrichen ist, gibt es keine Überlappung mehr und Frames müssen trotzdem Vollständig analysiert werden.
- Da mehr Zeit pro Frame gebraucht wird, müssen mehr Frames übersprungen werden und es könnten relevante Objekte übersehen werden.
- Falls sich etwas in einem bereits analysierten Bereich bewegt, würde dieser nicht mehr angeschaut und es würde nichts erkannt. Beispielsweise könnte eine Person hinter einem Stein sitzen und aufstehen sobald sie die Drohne hört, da die Drohne die Region aber schon analysiert hat bevor die Person aufgestanden ist, würde die Person nicht erkannt.

9.2.7 Alternative

Eine Alternative zu der beschrieben Optimierung wäre die Bilder in verschiedene Segmente, inklusive Überlappungen aufzuteilen und diese separat zu analysieren. Da die Analyse linear ist, geht diese Optimierung dementsprechend länger, aber dafür ist die Auflösung höher und die Objekte können besser erkennt werden.

Diese Variante liesse sich auch einfach als Konfigurationsmöglichkeit einbauen. Zum Beispiel könnte man Optionen für Teilen in 1, 2, 4 oder 8 Segmente anbieten.

9.2.8 Entscheidungskriterium

Für eine Entscheidung ob es Sinn ergibt dieses Konzept anzuwenden, muss geklärt werden, welche Methode weniger Objekte übersieht und dabei noch in einem legitimen Zeitrahmen durchgeführt werden kann:

1. Mehr Frames in kleinerer Auflösung
2. Weniger Frames in höherer Auflösung

Nur die nicht analysierten Teile eines Frames zu betrachten ist dabei nur eine Optimierung von Punkt 2. Diese Frage kann also geklärt werden, ohne diese Optimierung bereits zu implementieren.

Ein kurzer Vergleich zwischen Analysen in unterschiedlichen Auflösungen ist in Abschnitt 4.10 beschrieben.

9.3 Object Detection in der Android App

Während der Bachelorarbeit haben wir die iOS Applikation so erweitert, dass eine Echtzeitanalyse während des Fluges möglich ist. In einer vorherigen Arbeit wurde aber auch eine Android Applikation erstellt. Will man nun die selbe Funktionalität auch in dieser implementieren, wäre es wünschenswert, wenn das Modell nicht zwei mal trainiert werden muss.

Daher sollte es möglich sein, das bestehende CoreML Modell zu TensorFlow zu konvertieren, sodass es auf Android oder auch auf einem Server eingesetzt werden könnte.

9.3.1 Open Neural Network Exchange (ONNX)

ONNX ist ein Format für den Austausch von Modellen zwischen verschieden Libraries bzw. Frameworks [26]. Konkret können wir es einsetzen, um das CoreML Modell in ein TensorFlow Modell zu konvertieren oder auch umgekehrt, falls man das Training mit TensorFlow durchführen will.

Leider sind wir während dem Konvertieren auf ein Problem gestossen, dass ein Layer unseres Modelles noch nicht im ONNX Format unterstützt wird. Das

Problem ist den Entwicklern bekannt und sie haben uns versichert, dass es in nächster Zeit behoben werden wird.[46]

9.3.2 Alternativen zu ONNX

Sollte ONNX doch nicht funktionieren, gibt es im Moment keine weiteren Tools, um ein CoreML Modell zu TensorFlow zu konvertieren. Apple selber bietet mit dem Python Package coremltool[7] die umgekehrte Möglichkeit andere Formate zu CoreML zu konvertieren. Man könnte also mit einem anderen Tool ein Training durchführen und das erhaltene Modell in die gewünschten Formate konvertieren.

Es könnte jedoch sein, dass das Input oder Output Format nicht den Formaten von TuriCreate entspricht und die iOS und Mac Applikation angepasst werden müssten.

Kapitel 10

Schlussfolgerung

In den letzten 15 Wochen erarbeiteten wir das automatische Durchsuchen von Videos nach vermissten Personen und Gegenständen, welche Hinweise auf den Verbleib von Personen liefern könnten. Das Ziel war es zu überprüfen, ob die Analyse, trotz der vorgegebenen Restriktionen, schnell genug durchgeführt werden kann, um den bisherigen Prozess zu optimieren.

Dabei haben wir verschiedene Libraries und Frameworks getestet und miteinander verglichen. Schlussendlich haben wir uns für den Deep Learning basierten Object Detection Algorithmus YOLO entschieden und nutzen die Implementation des CoreML Frameworks von Apple, da dieses unsere Anforderungen am besten decken konnte.

Als nächsten Schritt haben wir dann eine macOS Applikation entwickelt, welche ein Videofile analysiert und während dem Abspielen die gefundenen Objekte markiert. Im Fortschrittsbalken wird hervorgehoben, wo Objekte gefunden wurden. Dadurch haben wir schon eine erste Verbesserung für den Rettungsprozess. Zusätzlich haben wir die verschiedenen Performance-Parameter konfigurierbar gemacht, sodass man einfach optimale Standardwerte finden kann.

Danach wurde am Fernziele Echtzeitanalyse gearbeitet. Dafür wurde die bestehende Rescuer App so angepasst, dass während dem Flug eine Analyse des Videostream der Drohne durchgeführt werden kann. Dank des optimierten Apple Chips war die Analyse schneller als auf einem Mac. Jedoch haben wir momentan noch das Problem, dass die Boxen an einer alten Position gezeichnet werden. Dieses Problem muss noch genauer analysiert werden, bevor man sagen kann, was der Grund für dieses Verhalten ist. Noch Anzumerken ist, dass dieses Problem nur mit dem Stream von DJI Drohnen auftritt und nicht mit dem der iPhone Kamera.

Gleichzeitig haben wir auch am Trainingsprozess für das neuronale Netzwerk Modell gearbeitet. Wir haben definiert, welche Daten nötig sind und diese dann gezielt gesammelt. Dafür haben wir zusammen mit den Projektpartnern geeignete Gebiete aufgesucht und Gegenstände und Personen darin verteilt. Anschliessend haben wir das Gebiet mit der Drohne, welche alles aufzeichnete, abgeflogen. Mit den gesammelten Daten haben wir dann verschiedene Trainings

durchgeführt und die Resultate miteinander verglichen. Dabei kamen wir zum Schluss, dass ein Modell mit einer oder zwei Klassen (Person und Gear) zu einem besseren Ergebnis führt, als ein Modell mit drei Klassen (Person, Clothing und Bag).

Im folgenden sind noch Screenshots der fertigen Apps, sowie Eindrücke vom Aufnehmen von Trainingsvideos abgebildet:

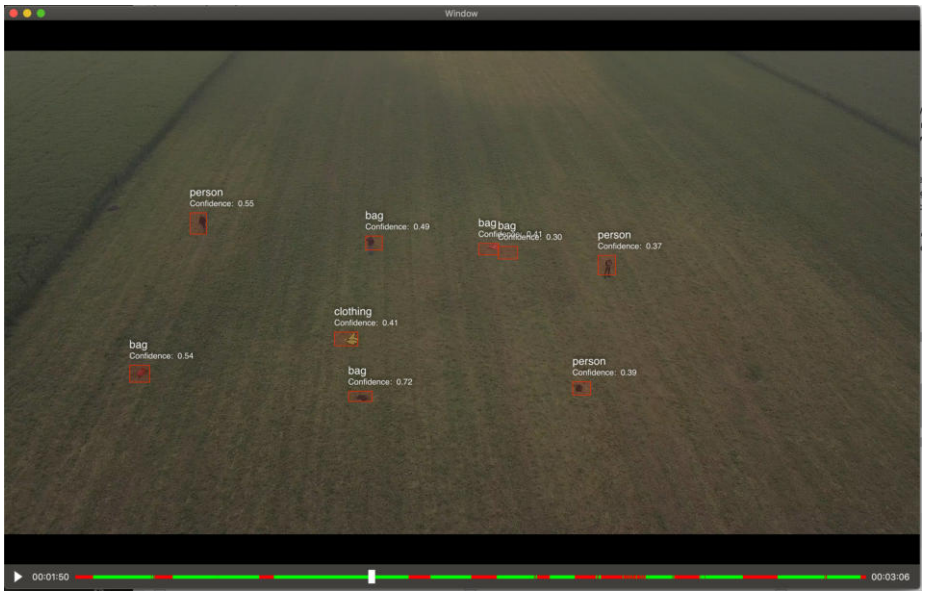

Abbildung 10.1: Screenshot aus macOS Applikation mit diversen erkannten Gegenständen.

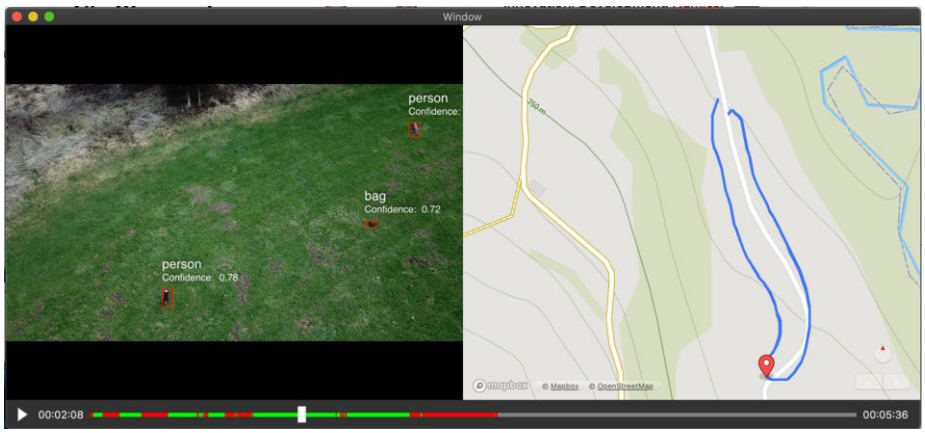

Abbildung 10.2: Screenshot aus macOS Applikation mit Kartenansicht der Flugroute.

Abbildung 10.3: Screenshot aus iOS App mit zwei erkannten Personen im Videostream.

Abbildung 10.4: Beim Aufnehmen von Trainingsvideos auf der Schwägalp. Bildquelle: Heinz Beutler

Abbildung 10.5: Drohne am Aufnehmen von Trainingsdaten auf der Schwägalp.

Kapitel 11

Ausblick

Nebst den beschrieben Resultaten haben wir auch mögliche Erweiterungen und Optimierungen konzeptionell ausgearbeitet, welche von den Projektpartnern oder auch von anderen Studenten im Rahmen weiterer Studien- oder Bachelorarbeit implementiert werden können.

Einerseits haben wir mögliche Optimierungen definiert, wie zum Beispiel das Analysieren von nur geänderten Bildabschnitten, sodass die Rechenleistung gezielter eingesetzt werden kann. Dabei muss abgewogen werden, ob man, im Gegenzug für eine höhere Genauigkeit, an Geschwindigkeit einbüssen möchte.

Andererseits haben wir auch eine mögliche Feedbackplattform konzipiert, welche es Drohnenpiloten erlaubt, Bilder oder Videos mit fehlerhaften Erkennungen zu korrigieren, sodass das Modell verbessert werden kann. Dadurch wird der Aufwand für das Sammeln von Testdaten an die Benutzer ausgelagert.

Zusätzlich gibt es noch eine Android Applikation, welche während unserer Arbeit nicht berücksichtigt wurde. Hierfür müsste das trainierte CoreML Modell zu TensorFlow konvertiert werden, damit es in der Applikation eingesetzt werden könnte.

Die Projektpartner haben bereits weitere Ideen, wie sie die gleichen Konzepte auf andere Anwendungsgebiete übertragen könnten.

Anhang A

Test Protokolle

In den folgenden Abschnitten werden alle manuellen Testfälle aufgelistet und auch deren Durchführungsresultate.

A.1 Testfälle Mac

A.1.1 Karte/Route wird nicht angezeigt wenn SRT fehlt

Testschritte	Erwartetet
1. Applikation starten 2. Video File auswählen, für das im selben Ordner kein SRT File mit gleichem Namen existiert	Karte mit Route wird nicht angezeigt

A.1.2 Karte/Route angezeigt wenn SRT vorhanden

Testschritte	Erwartetet
1. Applikation starten 2. Video File auswählen, für das im selben Ordner ein SRT File mit gleichem Namen existiert	Karte mit Route wird angezeigt

A.1.3 Position auf Karte verändert sich

Testschritte	Erwartetet
1. Applikation starten 2. Video File auswählen, für das im selben Ordner ein SRT File mit gleichem Namen existiert 3. Video abspielen	Position des „Pin“ verschiebt sich gemäss Flugroute

A.1.4 Slider verändert Position

Testschritte	Erwartetet
1. Applikation starten 2. Video File auswählen, für das im selben Ordner ein SRT File mit gleichem Namen existiert 3. Video abspielen 4. Slider verschieben	Karte wird angezeigt und „Pin“ verschiebt sich gemäss Flugroute

A.1.5 Analyse wird durchgeführt

Testschritte	Erwartetet
1. Applikation starten 2. Video File auswählen, welches Objekte beinhaltet 3. Video abspielen	Fortschrittsbalken wird eingefärbt. Wenn Abspielposition auf grünem Bereich, dann wird ein Objekt hervorgehoben

A.1.6 Weniger Frames anschauen ist schneller

Testschritte	Erwartetet
1. Applikation starten 2. In den Einstellungen: Nth Frame auf 1 setzen 3. Video File auswählen 4. 20 Sekunden warten 5. Applikation neu starten 6. In den Einstellungen: Nth Frame auf 20 setzen 7. Selbes Video File auswählen 8. 20 Sekunden warten	Analyse, welche nur jedes 20te Frame anschaut ist schneller

A.1.7 Analyse auf CPU bzw. GPU ist nicht gleich schnell

Testschritte	Erwartetet
1. Mac mit GPU verwenden 2. Applikation starten 3. In den Einstellungen: Analyse auf CPU 4. Video File auswählen 5. 20 Sekunden warten 6. Applikation neu starten 7. In den Einstellungen: Analyse auf GPU 8. Selbes Video File auswählen 9. 20 Sekunden warten	Analyse sollte unterschiedlich lange dauern

A.1.8 Bei hohem Treshold, werden weniger Objekte angezeigt

Testschritte	Erwartetet
1. Applikation starten 2. In den Einstellungen: Threshold auf 0 3. Video File auswählen 4. Warten bis erste Objekte erkannt 5. In den Einstellungen: Threshold auf 1	Es sollten keine Objekte mehr im ganzen Video angezeigt werden.

A.1.9 Einstellung Smoothness glättet Strecke

Testschritte	Erwartetet
1. Applikation starten 2. Einstellungen öffnen 3. Smoothness auf 1 setzen 4. Video File auswählen, für das im selben Ordner ein SRT File mit gleichem Namen existiert 5. Einstellungen öffnen 6. Smoothness auf 10 erhöhen	Route wird geglättet

A.2 Testfälle iOS

Wie aus der Aufgabenstellung zu entnehmen ist, existiert bereits eine iOS Applikation, welche wir anpassen können. Daher werden hier nur Testfälle aufgelistet, welche unseren Projektteil abdecken bzw. falls wir bestehende Funktionen ändern, werden auch dafür Tests definiert.

A.2.1 Objekte werden erkannt

Testschritte	Erwartetet
1. Auftrag erstellen 2. Objekte auf der Flugroute platzieren 3. Flug mit Drohne durchführen	Objekte werden erkannt (Ton & Box)

A.2.2 Analyse kann ausgeschaltet werden

Testschritte	Erwartetet
1. Auftrag erstellen 2. Objekte auf der Flugroute platzieren 3. In den Einstellungen der App Object Detection ausschalten. 4. Flug mit Drohne durchführen	Es werden keine gefunden Objekte angezeigt

A.3 Test 30.04.2019

Test	Status	Abweichung
A.1.1	OK	
A.1.2	OK	
A.1.3	OK	
A.1.4	OK	
A.1.5	OK	
A.1.6	OK	
A.1.7	OK	
A.1.8	OK	
A.1.9	OK	

A.4 Test 08.05.2019

Test	Status	Abweichung
A.1.1	OK	
A.1.2	OK	
A.1.3	OK	
A.1.4	OK	
A.1.5	OK	
A.1.6	OK	
A.1.7	OK	
A.1.8	OK	
A.1.9	OK	

A.5 Test 10.05.2019

Test	Status	Abweichung
A.2.1	Fehler	Erkennt Objekte nicht, Erkennt anderes als Objekte
A.2.2	OK	

A.6 Test 10.05.2019

Test	Status	Abweichung
A.2.1	Fehler	Erkennt Objekte, aber Position ist falsch
A.2.2	OK	

A.7 Test 21.05.2019

Test	Status	Abweichung
A.1.1	OK	
A.1.2	OK	
A.1.3	OK	
A.1.4	OK	
A.1.5	OK	
A.1.6	OK	
A.1.7	OK	
A.1.8	OK	
A.1.9	Fehler	Beim verändern der Smoothness Einstellung, ist die Route nicht mehr sichtbar, wenn man MapKit verwendet. Mit Apple Maps tritt das Problem nicht auf.

A.8 Test 21.05.2019

Test	Status	Abweichung
A.1.1	OK	
A.1.2	OK	
A.1.3	OK	
A.1.4	OK	
A.1.5	OK	
A.1.6	OK	
A.1.7	OK	
A.1.8	OK	
A.1.9	OK	

Anhang B

Messresultate Trainingsdauer

Anzahl Trainingsbilder	Anzahl Objekte	Iterationen	Dauer
1	-	1000	902s
19	34	1000	4754s
38	72	1000	6008s
56	111	2000	9949s
70	-	2000	10300s
75	152	2000	10127s
94	183	2000	10328s
113	215	2000	10278s
132	249	2000	10261s
150	281	3000	15456s
151	-	3000	15200s
169	322	3000	15177s
188	363	3000	15267s
207	403	3000	15293s

Anzahl Trainingsbilder	Anzahl Objekte	Iterationen	Dauer
226	436	3000	15317s
230	-	3000	15840s
244	470	3000	15312s
263	503	4000	20924s
282	539	4000	20739s
290	-	4000	21932s
301	572	4000	20671s
320	607	4000	21150s
338	642	4000	20766s
357	674	4000	19708s
370	-	4000	21000s
376	709	4000	20322s

Tabelle B.1: Messresultate für Trainingsdauer aufgrund unterschiedlicher Anzahl an Trainingsbildern

Anhang C

Glossar

Batch Ein Subset der Trainingsdaten, welches in der selben Trainingsiteration verwendet wird. 77

CUDA Steht für Compute Unified Device Architecture. CUDA ist NVIDIAs Programmiermodell um Applikationen für NVIDIA GPUs zu entwickeln. 23, 25

Deep Learning Erlaubt es einem Computer basierend auf Erfahrungen zu lernen. Ein komplexes Konzept wird auf viele simple Konzepte aufgeteilt, welche voneinander abhängig sind. Dies kann als tiefen (Deep) Graph dargestellt werden[18]. 2, 14, 23, 24, 26, 63, 77

DSL Steht für Domain Specific Language.. 52, 81

Frame Ein Standbild / Einzelbild aus einem Video. 2, 4, 8, 10, 17, 18, 22, 25, 26, 30, 31, 44, 45, 48, 49, 52, 53, 55–57, 59–61, 70

Iteration Abarbeiten eines Batches während des Trainingsprozesses. Basierend auf dem Resultat einer Iteration wird das Modell mittels Backpropagation verbessert. 34–37, 77

Layer Neuronen in einem neuronalen Netzwerk, welche auf der selben Ebene sind bilden einen Layer. 28, 61

Modell Enthält die Gewichtungen und die Architektur eines trainierten neuronalen Netzwerks. 2, 4, 21–23, 25, 26, 28, 29, 31, 37–42, 50, 51, 57, 58, 61–64, 67, 77, 78

Neuronale Netzwerke Ansatz von Deep Learning. Das Konzept basiert auf den vernetzten Neuronen im Gehirn[18]. 2, 28, 57, 59, 63, 77

Overfit Beim Trainieren wurden die Trainingsdaten auswendig gelernt und das Modell generalisiert nun schlechter. Das führt zwar zu sehr kleinen Fehlern, wenn mit dem Trainingsset getestet wird, führt aber zu grossen Fehlern beim Testen mit dem Testset[18]. 26

Testset Daten welche am Ende des Trainingsprozesses verwendet werden, um das Modell zu evaluieren. Die Daten sollten nicht in den Trainingsdaten vorkommen. 2, 29–31, 37–41, 55, 58, 67, 78

Trainingsset Daten mit welchen das Modell lernen und versucht zu generalisieren. 2, 26, 29–31, 34, 36, 40, 55, 58, 66, 77, 78

WSL Windows Subsystem for Linux `https://docs.microsoft.com/en-us/windows/wsl/install-win10`. 25

Anhang D

Bibliography

[1] Yali Amit. *2d Object Detection and Recognition: Models, Algorithms and Networks*. MIT Press, 2002. ISBN: 9780262011945.

[2] *Apple Core ML*. https://developer.apple.com/documentation/coreml. Accessed: 2019-03-26.

[3] *Caffe2*. https://caffe2.ai. Accessed: 2019-03-25.

[4] *Carthage Github Repository*. https://github.com/Carthage/Carthage. Accessed: 2019-05-27.

[5] *COCO Dataset*. http://cocodataset.org/. Accessed: 2019-03-25.

[6] *CocoaPods*. https://cocoapods.org. Accessed: 2019-05-27.

[7] *Converters - coremltools 2.0 documentation*. https://apple.github.io/coremltools/coremltools.converters.html. Accessed: 2019-03-26.

[8] *Darkflow*. https://github.com/thtrieu/darkflow. Accessed: 2019-03-25.

[9] *Darknet YOLO*. https://pjreddie.com/darknet/yolo/. Accessed: 2019-03-25.

[10] Lukas Oberholzer und David Riederer. "Einsatz von Drohnen zur Unterstützung von alpinen Rettungsaktionen". Bachelorarbeit. HSR Hochschule für Technik Rapperswil, 2018.

[11] *Distributed Training in TensorFlow | TensorFlow Core | TensorFlow*. https://www.tensorflow.org/guide/distribute_strategy. Accessed: 2019-05-21.

[12] *DJI Mobile SDK*. https://developer.dji.com/mobile-sdk/. Accessed: 2019-05-27.

[13] *DJI SRT Parser Github Repository*. https://github.com/JuanIrache/DJI_SRT_Parser/blob/master/index.js. Accessed: 2019-05-28.

[14] *DJI UX SDK*. https://developer.dji.com/mobile-sdk/documentation/introduction/ux_sdk_introduction.html. Accessed: 2019-05-27.

[15] *DJIWidget Library*. https://github.com/dji-sdk/DJIWidget. Accessed: 2019-05-27.

[16] *Does Turi Create resize or crop images*. https://github.com/apple/turicreate/issues/336. Accessed: 2019-04-19.

[17] Ross Girshick u. a. "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation". In: *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*. CVPR '14. Washington, DC, USA: IEEE Computer Society, 2014, S. 580–587. ISBN: 978-1-4799-5118-5. DOI: 10.1109/CVPR.2014.81. URL: https://doi.org/10.1109/CVPR.2014.81.

[18] Ian Goodfellow, Yoshua Bengio und Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[19] *IBM Watson - Apple Developer*. https://developer.apple.com/ibm/. Accessed: 2019-05-21.

[20] *IBM Watson - Custom Object Detection (Beta)*. https://cloud.ibm.com/docs/services/visual-recognition?topic=visual-recognition-object-detection-overview. Accessed: 2019-05-21.

[21] *IBM Watson Visual Recognition*. https://www.ibm.com/watson/services/visual-recognition/index.html. Accessed: 2019-05-21.

[22] *ImageAI : Video Object Detection, Tracking and Analysis*. https://github.com/OlafenwaMoses/ImageAI/blob/master/imageai/Detection/VIDEO.md. Accessed: 2019-03-05.

[23] *Maps | Mapbox*. https://www.mapbox.com/maps/. Accessed: 2019-05-27.

[24] David Masko und Paulina Hensman. "The Impact of Imbalanced Training Data for Convolutional Neural Networks". Independent thesis Basic level. KTH, School of Computer Science und Communication (CSC), 2015.

[25] *Official English Documentation for ImageAI*. https://imageai.readthedocs.io/en/latest/. Accessed: 2019-02-27.

[26] *Open Neural Network Exchange https://onnx.ai/*. https://github.com/onnx/onnx. Accessed: 2019-05-21.

[27] Rafael Padilla. *Metrics for Object Detection*. https://github.com/rafaelpadilla/Object-Detection-Metrics. Accessed: 2019-05-25.

[28] S. J. Pan und Q. Yang. "A Survey on Transfer Learning". In: *IEEE Transactions on Knowledge and Data Engineering* 22.10 (Okt. 2010), S. 1345–1359. ISSN: 1041-4347. DOI: 10.1109/TKDE.2009.191.

[29] *PyTorch*. https://pytorch.org. Accessed: 2019-03-26.

[30] *realm/SwiftLint: A tool to enforce Swift style and conventions*. https://github.com/realm/SwiftLint. Accessed: 2019-04-02.

[31] Joseph Redmon. *Darknet: Open Source Neural Networks in C*. https://pjreddie.com/darknet/. Accessed: 2019-03-26.

[32] Joseph Redmon und Ali Farhadi. “YOLO9000: Better, Faster, Stronger”. In: *arXiv preprint arXiv:1612.08242* (2016).

[33] Joseph Redmon u. a. “You Only Look Once: Unified, Real-Time Object Detection”. In: Juni 2016, S. 779–788. DOI: 10.1109/CVPR.2016.91.

[34] Shaoqing Ren u. a. “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39 (2015), S. 1137–1149.

[35] Fred Rothganger u. a. “3D Object Modeling and Recognition Using Local Affine-Invariant Image Descriptors and Multi-View Spatial Constraints”. In: *International Journal of Computer Vision* 66.3 (März 2006), S. 231–259. ISSN: 1573-1405. DOI: 10.1007/s11263-005-3674-1. URL: %5Curl%7Bhttps://doi.org/10.1007/s11263-005-3674-1%7D.

[36] *RxSwift: ReactiveX for Swift*. https://github.com/ReactiveX/RxSwift. Accessed: 2019-05-27.

[37] *SnapKit: An Auto Layout DSL for iOS & OSX*. https://github.com/SnapKit/SnapKit. Accessed: 2019-05-27.

[38] Jamie Street. *Image of a Dog*. https://unsplash.com/photos/me0hi86szok?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText. Accessed: 2019-06-04.

[39] *Swift Package Manager*. https://swift.org/package-manager/. Accessed: 2019-05-27.

[40] *TensorFlow: An open source machine learning framework for everyone*. https://www.tensorflow.org/. Accessed: 2019-02-27.

[41] *Tensorflow Object Detection API*. https://github.com/tensorflow/models/tree/master/research/object_detection. Accessed: 2019-03-26.

[42] *Turi Create Object Detection: How it works*. https://github.com/apple/turicreate. Accessed: 2019-02-26.

[43] *Turi Create: Object Detector evaluate*. https://apple.github.io/turicreate/docs/api/generated/turicreate.object_detector.ObjectDetector.evaluate.html. Accessed: 2019-05-26.

[44] *turicreate GitHub Repository*. https://github.com/apple/turicreate. Accessed: 2019-02-26.

[45] Markus Ulrich, C Wiedemann und C Steger. “Cad-based recognition of 3d objuriects in monocular images”. In: *International Conference on Robotics and Automation* (Jan. 2009), S. 1191–1198.

[46] *Unsupported shape calculation for operator nonMaximumSuppression*. https://github.com/onnx/onnxmltools/issues/300. Accessed: 2019-05-25.

[47] *Vision | Apple Developer Documentation*. https://developer.apple.com/documentation/vision. Accessed: 2019-02-26.

[48] *Visual Object Tagging Tool: An electron app for building end to end Object Detection Models from Images and Videos.* https://github.com/Microsoft/VoTT. Accessed: 2019-04-23.

[49] *VNTranslationalImageRegistrationRequest - Vision | Apple Developer Documentation.* https://developer.apple.com/documentation/vision/vntranslationalimageregistrationrequest. Accessed: 2019-05-21.

[50] *Windows and Linux version of Darknet YOLO v3 and v2 Neural Networks for object detection.* https://github.com/AlexeyAB/darknet. Accessed: 2019-03-26.

[51] *Working with Core ML Models.* https://developer.apple.com/machine-learning/build-run-models/. Accessed: 2019-02-26.

[52] *XCTest: Create and run unit tests, performance tests, and UI tests for your Xcode project.* https://developer.apple.com/documentation/xctest. Accessed: 2019-04-02.