ingenieur
wissenschaften
htw saar

Hochschule für
Technik und Wirtschaft
des Saarlandes
University of
Applied Sciences

STL SYSTEMTECHNIKLABOR

Prototypische Entwicklung einer KI-basierten Erste-Hilfe-App

Tobias Gottschalk

Technical Report – STL-TR-2022-01 – ISSN 2364-7167

stl.htwsaar.de

Technische Berichte des Systemtechniklabors (STL) der htw saar
Technical Reports of the System Technology Lab (STL) at htw saar
ISSN 2364-7167

Tobias Gottschalk: Prototypische Entwicklung einer KI-basierten Erste-Hilfe-App
Technical report id: STL-TR-2022-01

First published: March 2022
Last revision: November 2021
Internal review: Tobias Greff, André Miede

For the most recent version of this report see: https://stl.htwsaar.de/

Title image source: succo, https://pixabay.com/de/photos/erste-hilfe-rettung-opfer-retter-850485/

htw saar – Hochschule für Technik und Wirtschaft des Saarlandes (University of Applied Sciences)
Fakultät für Ingenieurwissenschaften (School of Engineering)
STL – Systemtechniklabor (System Technology Lab)
Prof. Dr.-Ing. André Miede (andre.miede@htwsaar.de)
Goebenstraße 40
66117 Saarbrücken, Germany
https://stl.htwsaar.de

ingenieur
wissenschaften
htw saar

Hochschule für
Technik und Wirtschaft
des Saarlandes
University of
Applied Sciences

Master-Thesis

zur Erlangung des akademischen Grades
Master of Science (M. Sc.)
an der Hochschule für Technik und Wirtschaft des Saarlandes
im Studiengang Praktische Informatik
der Fakultät für Ingenieurwissenschaften

Prototypische Entwicklung einer KI-basierten Erste-Hilfe-App

vorgelegt von
Tobias Gottschalk

betreut und begutachtet von
Prof. Dr.-Ing. André Miede
Tobias Greff

Saarbrücken, 30. November 2021

Selbständigkeitserklärung

Ich versichere, dass ich die vorliegende Arbeit (bei einer Gruppenarbeit: den entsprechend gekennzeichneten Anteil der Arbeit) selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Ich erkläre hiermit weiterhin, dass die vorgelegte Arbeit zuvor weder von mir noch von einer anderen Person an dieser oder einer anderen Hochschule eingereicht wurde.

Darüber hinaus ist mir bekannt, dass die Unrichtigkeit dieser Erklärung eine Benotung der Arbeit mit der Note „nicht ausreichend“zur Folge hat und einen Ausschluss von der Erbringung weiterer Prüfungsleistungen zur Folge haben kann.

Saarbrücken, 30. November 2021

Tobias Gottschalk

Zusammenfassung

Assistenzsysteme, die bei medizinischen Angelegenheiten helfen und unterstützen, dringen immer weiter in die verschiedenen medizinischen Bereiche vor. Bedingt durch den demografischen Wandel ist davon auszugehen, dass der Bedarf nach KI-gestützten Assistenzsystemen weiter steigen wird. Diese Arbeit beschäftigt sich mit der Frage, wie Anwender durch ein entsprechendes System in Erste-Hilfe-Situationen aktiv unterstützt werden können. Vorgegangen wird nach dem Design-Science Research Ansatz. Vorhandene Ansätze und Apps werden analysiert und die Anforderungen werden aufgestellt. Konzeptionell wurde eine App entworfen, die ein Empfehlungssystem benutzt und dem Anwender Handlungsempfehlungen für die Behandlung von vier verschiedenen Wundtypen gibt. Das Empfehlungssystem greift auf zwei Deep-Learning Modelle zurück, die eine Wundklassifizierung und -lokalisierung durchführen. Als Proof-of-Concept wurde eine prototypische Android-App mit Unity entwickelt und die Modelle wurden integriert. Abschließend wurde die App durch eine Einzelfallstudie evaluiert.

Mit dem ersten Glied ist die Kette geschmiedet. Wenn die erste Rede zensiert, der erste Gedanke verboten, die erste Freiheit verweigert wird, dann sind wir alle unwiderruflich gefesselt.

— Erik Satie

Danksagung

Ich möchte mich bei Prof. Dr. André Miede bedanken, der mich ausgezeichnet während der Master-Thesis betreut und unterstützt hat. Ebenfalls möchte ich mich bei Tobias, Kevin und Svea bedanken, die mich bei der Erstellung der Arbeit super unterstützt haben. Ebenfalls möchte ich mich bei meiner gesamten Familie bedanken, die mir während der Thesis geholfen und mich entlastet haben. Mein besonderer Dank gilt meiner Frau Jasmin. Du hast einen erheblichen Anteil dazu beigetragen, dass ich nicht nur die Master-Thesis, sondern auch das gesamte Studium erfolgreich bestreiten konnte.

Inhaltsverzeichnis

1 Einleitung

Als Ende des Jahres 2018 die Bertelsmann Stiftung ihre Studie #SmartHealthSystems [62] veröffentlicht hat, in der sie Digitalisierungsstrategien international vergleicht, ging ein Raunen durch die Republik. Deutschland belegt den vorletzten Platz unter den 17 verglichenen Ländern [62, S. 225]. Seit dem Jahr 2018 hat sich allerdings einiges getan. Es wurden viele Reformen auf Bundes- und Landesebene angestoßen und weiter voran getrieben. Hierbei ist beispielsweise das health innovation hub (hih) oder das Health.AI zu nennen. Das am 25. Oktober 2019 in Kraft getretene Krankenhauszukunftsgesetz (KHZG) sieht eine Förderung von über vier Milliarden Euro für die Modernisierung von Krankenhäusern in Bezug auf die Digitalisierung und auf die IT-Sicherheit vor [15]. Die tragenden Säulen der Digitalisierungsstrategie sind unter anderem die Einführung der elektronische Patientenakte (ePA), die Schaffung einer nationalen Infrastruktur zur Nutzung von digitalen Diensten sowie die Sicherstellung der Standardisierung und der Interoperabilität.

Die Coronakrise hat für einen weiteren Innovationsschub gesorgt und war gleichzeitig ein Stresstest für das Gesundheitssystem. Die große Aufgabe der Digitalisierung des Gesundheitswesens kann nicht ohne entsprechende erfahrene Experten bewerkstelligt werden. Ein Experte für die Digitalisierung ist das August-Wilhelm-Scheer-Institut für digitale Produkte und Prozesse gGmbH (AWSi), bei dem diese Masterthesis entstanden ist. Das AWSi ist eine gemeinnützige, unabhängige Forschungseinrichtung zur Erforschung gradueller und disruptiver Digitalisierung von Wirtschaft und Gesellschaft. Im Schwerpunkt werden neue Technologien und Konzepte der Mensch-Maschine-Interaktion mit Fokus auf zukünftige Arbeitsprozesse erforscht. Dabei entwickelt und evaluiert das Institut primär Produkt-Service-Systeme in den Bereichen virtuelle Assistenzdienste, Robotic-Process-Automation, maschinelles Lernen sowie künstliche Intelligenz (KI) und smarter Interfaces. Im Projekt KAMeri werden dazu bspw. Brain-Computer-Interfaces zum Arbeitsschutz in der Mensch-Robotik-Kollaboration umgesetzt (BMBF, KAMeri). Von Relevanz für die automatisierte Arbeitsdokumentation ist zudem das Projekt KIRPA, in dem kognitive Softwareroboter (RPA) genutzt werden, um Arbeitsprozesse zum Datenaustausch mit Büro-IT-Systemen zu automatisieren (BMBF, KiRPA).

1.1 Motivation

Die Digitalisierung gewinnt zunehmend an Bedeutung im Gesundheitswesen. Ein Treiber für die Digitalisierung ist die künstliche Intelligenz. KI-gestützte Assistenzsysteme werden heute bereits im Gesundheitsbereich eingesetzt, um medizinisches Personal bei komplexen Entscheidungen zu unterstützen. Die Nutzung von KI-basierten Assistenzsystemen ist jedoch ungleich verteilt. Während bei der Bildgebung, speziell bei der Radiologie, die Nutzung KI-basierter Assistenzsysteme weit fortgeschritten ist, ist dies in anderen Bereichen nicht der Fall. Bedingt durch den demografischen Wandel und durch den Engpass beim Pflegepersonal, ist die Weiterentwicklung dieser Assistenzsysteme essenziell. Ein weiteres Assistenzsystem ist im Bereich der Ersten Hilfe interessant und bietet Potenzial für medizinisches wie auch für nicht-medizinisches Personal. In vielen anderen Bereichen, wie in der Heimpflege durch pflegende Angehörige, im Rahmen der betrieblichen Ersthilfe durch Ersthelfer oder in verschiedenen Betreuungssettings, z. B.

durch Ehrenamtliche, sind noch keine etablierten Systeme vorzufinden, die den jeweiligen Personen eine KI-gestützte Assistenz bieten und damit die optimale Versorgung der Menschen gewährleisten. Im ländlichen Raum mit einer geringeren Krankenhaus- und Ärztedichte können Assistenzsysteme die medizinische Versorgung verbessern. Im Jahre 2015 gab es 3,15 Millionen Unfälle im häuslichen Bereich sowie 3,89 Millionen Unfälle bei Freizeitaktivitäten [4]. Durch die Coronakrise und durch den daraus resultierenden gestiegenen Bedarf an Homeoffice, ist anzunehmen, dass sich diese Zahlen weiter erhöhen. Eine schnelle Hilfe ist ebenso ausschlaggebend wie eine qualitativ hochwertige Hilfe. Mythen, die sich auf die Wundversorgung beziehen, halten sich hartnäckig. Ein solcher Mythos ist beispielsweise, dass Wunden am besten ‚an der Luft' heilen [1].

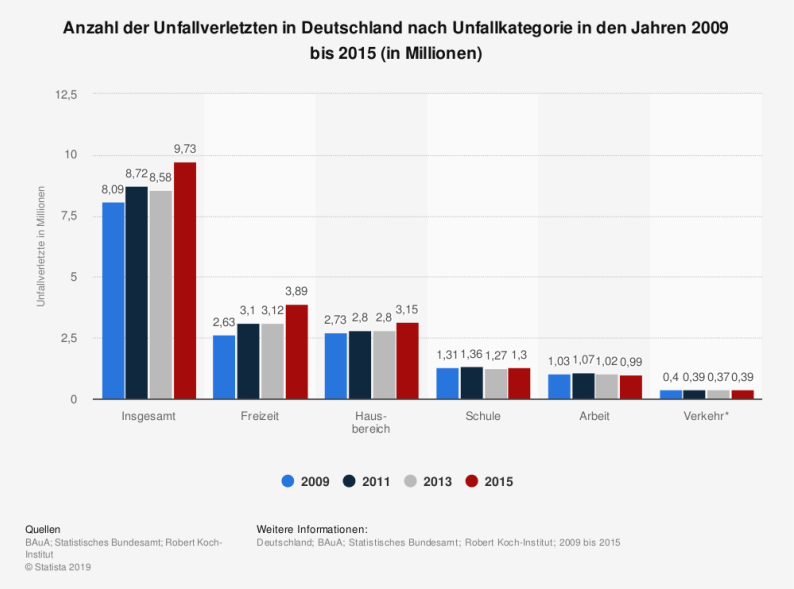


Abbildung 1.1: Anzahl der Unfallverletzten, 2009 bis 2015 [4]

Die korrekte Wundbehandlung wirkt sich direkt auf einen positiven Wundverlauf aus. Wundreinigung und Wunddesinfektion können durch Assistenzsysteme aktiv unterstützt werden. Dem Anwender kann z. B. aufgezeigt werden, welcher Verband zu nutzen ist und wie dieser angebracht werden kann oder zu welchem Zeitpunkt eine Selbstversorgung nicht mehr möglich ist und ein Arzt aufgesucht werden muss. Die Beobachtung des Wundverlaufs kann ebenfalls unterstützt werden. Der Wundverlauf kann sich über die Behandlung hinweg verändern. Ein klassisches Beispiel hierfür ist, dass sich die Wunde entzündet. Ein präventives Assistenzsystem kann den Anwender davor schützen.

1.2 Aufgabenstellung und Zielsetzung

Ziel der Arbeit ist die prototypische Entwicklung eines KI-Assistenzsystems in Form einer Erste-Hilfe-App, die Anwendern in Erste-Hilfe-Situationen bedarfsorientierte Handlungsempfehlungen geben soll. Dabei beschäftigt sich die Arbeit mit der Fragestellung, wie der Anwender in besonderen medizinischen Situationen und bei der Versorgung von Wunden unterstützt und angeleitet werden kann. Bei der Entwicklung sollen zwei verschiedene KI-Systeme zum Einsatz kommen. Das eine KI-System soll eine Lokalisierung durchführen, bei der erkannt werden soll, an welchem Körperteil sich die Wunde befindet. Das zweite KI-System soll eine Klassifizierung der Wunde durchführen. Die genauen Kriterien werden im Rahmen der Arbeit festgelegt. Denkbar ist eine Klassifizierung nach Wundtyp

(Schürf-, Schnitt- oder Platzwunden) oder nach Wundstatus (frisch oder neugebildetes Gewebe). Die Handlungsanweisung für den Anwender berücksichtigt den Ort der Wunde und die Klassifizierung. Sie ändert sich gegebenenfalls bei gleichem Wundtyp. Infektionen am Kopf sind gefährlicher als beispielsweise am Arm, da die Blut-Hirn-Schranke bereits überschritten ist.

Der Schutz der personenbezogenen Daten unterliegt den strengen Anforderungen der Datenschutz-Grundverordnung (DSGVO). Das gilt vor allem im Gesundheitswesen, wo sensible Patientendaten für die Nutzung von KI-Systemen benötigt werden. Dementsprechend ist die Schwelle für die Veröffentlichung von Datensätzen weitaus höher als bei anderen Branchen. Es gibt daher tendenziell weniger öffentliche Datensätze. Im Rahmen der Arbeit muss daher auch evaluiert werden, ob bereits genügend Daten vorhanden sind, um die Modelle zu trainieren. Für den Fall, dass kein bestehender Datensatz benutzt werden kann, muss ein eigener Datensatz erstellt werden.

1.3 Aufbau der Arbeit

Im ersten Kapitel wird auf die Motivation und auf die Aufgabenstellung inklusive des Ziels eingegangen. Im Kapitel 2 werden alle erforderlichen Grundlagen für die Entwicklung der Erste-Hilfe-App aufgezeigt. Danach erfolgt im Kapitel 3 die Analyse. Es wird der Stand der Technik dargelegt und es wird ein Blick auf bereits vorhandene Erste-Hilfe-Apps geworfen. Das grundlegende Problem wird ebenfalls analysiert und die Lücke zwischen dem Stand der Technik und dieser Arbeit wird aufgezeigt. Basierend auf den in Kapitel 3 erfassten Anforderungen wird im vierten Kapitel die App entworfen. Es wird dabei auf die Architekturen der verwendet Deep-Learning-Modelle eingegangen, Mockups für die GUI werden entworfen und die Handlungsempfehlungen werden definiert. In Kapitel 5 wird die praktische Implementierung erläutert. Diese umfasst drei Punkte: die Entwicklung der beiden KI-Systeme für die Lokalisierung und für die Klassifizierung der Wunde sowie für das Betreiben der für die App notwendigen Komponenten. Im vorletzten Kapitel wird eine erste Evaluierung der App vorgenommen. In Kapitel 7 werden die Ergebnisse dieser Arbeit zusammengefasst und bewertet. Ebenfalls wird ein Ausblick auf mögliche Verbesserungen und weitere Fragestellungen, die mit der Arbeit entstanden sind, gegeben.

1.4 Vorgehen nach Design-Science-Research

In der Informatik existieren zahlreiche Vorgehen, die für eine wissenschaftliche Arbeit herangezogen werden können. Eine anerkannte Methodologie ist Design Science Research (DSR) nach Hevner [31], an dessen strukturiertem Vorgehen sich diese Arbeit orientiert.

Ziel des DSR ist es, neue Ergebnisse mithilfe eines praxisnahen Forschungsansatzes zu entwickeln. Die Ergebnisse werden Artefakte genannt. Ein Artefakt kann beispielsweise eine neue Software, neue Methoden, neue Modelle oder ein neues Konzept sein. Der Ergebnisbeitrag und nicht der Erkenntnisbeitrag steht dabei im Vordergrund. DSR fokussiert sich darauf, ein praktisches Problem zu lösen. Die Verzahnung und die tiefe Integration der Wissenschaft und der Praxis führen zu kürzeren Feedbackzyklen. Diese erlauben dann wieder einen schnelleren und frühen Forschungsertrag. Das Feedback durch die Praxis ermöglicht auch eine schnellere Einarbeitung in die Problemdomäne. [vgl. 13, S. 67 bis 68]

DSR besteht aus drei eng verbundenen Zyklen, die sich gegeneinander beeinflussen. Diese drei Zyklen sind der Relevanz-, der Design- und der Rigor-Zyklus. Abbildung 1.2 illustriert DSR. Der Relevanz-Zyklus verbindet die Umwelt mit dem DSR. Die Umwelt kann z. B. eine Firma, ein Mensch, ein technisches System oder ein Problem sein. Die Umwelt bildet die Anwendungsdomäne. Der Relevanz-Zyklus ist der erste Schritt in der Arbeit. Er dient dazu, die Chancen und die Probleme zu identifizieren und darzustellen. Er liefert die Anforderungen, die an das Artefakt gestellt werden und die Kriterien, die erfüllt sein müssen [vgl. 32, S. 3]. Für den Fall, dass nach der Entwicklung das Artefakt nicht wie gefordert funktioniert, muss wieder in den Relevanz-Zyklus zurückgekehrt werden. Chancen und Probleme müssen dann erneut geprüft werden.

Der Rigor-Zyklus stellt die Brücke zwischen der Wissensbasis und dem DSR dar. Die Grundlage bilden die gesamte bestehende Fachliteratur sowie die wissenschaftlichen Theorien und Methoden. Die Wissensbasis enthält außerdem zwei verschiedene Typen. Diese sind zum einen die Erfahrungen und das Fachwissen, das den Stand der Forschung bildet, und zum anderen die bestehenden Artefakte und Prozesse, die in der Anwendungsdomäne zu finden sind. Der Rigor-Zyklus stellt sicher, dass die Artefakte neu und innovativ sind.[vgl. 32, S. 4 bis 5] Hier unterscheidet sich DSR von normalen Softwareprojekten, denn diese adressieren ebenfalls ein Problem, entwerfen eine Lösung und testen, ob die Lösung funktioniert. Mit DSR soll allerdings durch den Prozess das Gelernte abstrahiert werden und Prinzipien oder theoretische Schlüsse sollen gezogen werden. Der Rigor-Zyklus ist aber auch für den Rückfluss von DSR zur Wissensbasis verantwortlich. So muss das finalisierte Artefakt zurück in die Wissensbasis fließen. Dies stellt einen Beitrag zur Forschung dar.

Der Design-Zyklus ist der Kern von DSR. Der Zyklus stellt die Beziehung zwischen der Entwicklung und der Evaluierung dar. Er wird iterativ ausgeführt. Wie in Abbildung 1.2 dargestellt, beinhaltet der innere Zyklus die Beziehung zwischen der Artefakterstellung und der Evaluation. Es ist auf eine Balance zwischen dem Aufwand für die Erstellung und der Bewertung des Artefakts[vgl. 32, S. 6]. Nach der Entwicklung des Artefakts, wird mithilfe der Kriterien, die im Relevanz-Zyklus entwickelt wurden, das Artefakt bewertet. Für die Evaluation können verschiedene Methoden, wie Fallstudie, Einzelfallstudie, Experimente oder Simulation, genutzt werden.

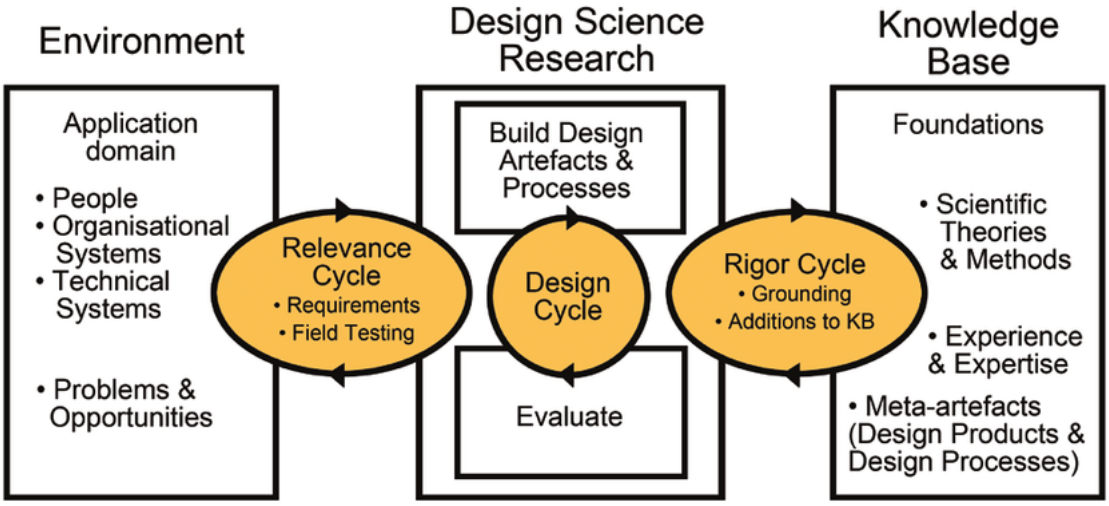


Abbildung 1.2: Design Science Research Zyklen [32, S. 3]

2 Grundlagen

In diesem Kapitel werden grundlegende Technologien, die für diese Arbeit relevant sind, erörtert. Zuerst wird Unity als zentrales Entwicklungswerkzeug vorgestellt. Danach werden die notwendigen Frameworks und Deep-Learning-Modelle eingeführt. Zum Schluss wird auf die Programmbibliothek Open Source Computer Vision (OpenCV) sowie auf die Unity-Integration OpenCV for Unity verwiesen, mithilfe derer die Deep-Learning-Modelle in die App eingebunden werden.

2.1 Unity

Unity ist eine flexible und mächtige Echtzeit-Entwicklungsumgebung. Sie ist zusammen mit Unreal de facto der Standard für Spiele-Engines. Neben der klassischen Spieleentwicklung können auch Augmented Reality (AR) und Virtual Reality (VR) Spiele entwickelt werden. AR ist die Erweiterung der Realität mit digitalen Informationen. Der Anwender nimmt also noch seine Umwelt physisch war. Ein prominentes Beispiel für eine AR-Anwendung ist Pokémon GO. Die Hardwareplattform ist meistens ein Smartphone oder eine spezielle Brille mit einem Display und mit verschiedenen Sensoren, wie mit Bewegungssensoren oder mit einer Gestensteuerung. Im Gegensatz zu AR wird in VR die Umgebung komplett mithilfe des Computers generiert. Der Benutzer der VR-Anwendung taucht dabei völlig in die virtuelle Welt ein und nimmt seine Umwelt nicht mehr wahr. Für VR-Anwendungen sind VR-Brillen notwendig, die nur in Verbindung mit einem Computer funktionieren. Die Bewegungssteuerung innerhalb der virtuellen Umgebung wird mittels eines Controllers durchgeführt. Beispiele für VR-Brillen sind Oculus Quest 2, HTC Vive Pro 2 oder Pimax Vision 8K. Zielplattformen für Unity sind der PC, Spielekonsolen, Android oder iOS.

Unity ist aber nicht nur für die Spieleentwicklung interessant. Es können auch andere Bereiche adressiert werden. Aktuelle Anwendungen im Gesundheitswesen sind z. B. folgende:

- Training und Ausbildung von medizinischem Personal
- Diagnostische Medizin
- Patientenbetreuung und Nachsorge
- Psychotherapie

Die in Unity verwendete Programmiersprache ist C# und diese ermöglicht einen schnellen und einfachen Einstieg in die Entwicklung. Unity ist in verschiedene Fenster unterteilt, die jeweils unterschiedliche Aufgaben übernehmen und dynamisch angepasst werden können. Abbildung 2.1 zeigt ein typisches Layout, das auch während der Entwicklung benutzt wird. Im Hierarchiefenster, gekennzeichnet mit einer 1 in Abbildung 2.1, werden alle GameObjects (siehe Abschnitt 2.1.1), wie Kameras, Modelle oder C#-Skripte dargestellt. Mithilfe des Hierarchiefensters werden die Eltern-Kind-Beziehungen festgelegt. Die GameObjects können gruppiert und verschachtelt werden. Die Hierarchie regelt auch, in

welcher Reihenfolge die Elemente angezeigt werden. Elemente, die sich oben in der Hierarchie befinden, werden von Elementen verdeckt, die sich weiter unten in der Hierarchie befinden.

Im Szenenfenster, gekennzeichnet mit einer 2, wird die Umgebung innerhalb des Spieles oder der App dargestellt. Szenen sind die Grundbausteine, nach denen ein Spiel oder eine App aufgebaut ist. Sie können mit einem Spielelevel oder mit einer Page einer Android App verglichen werden. Es können die Sichten *Scene* und *Game* ausgewählt werden. Die Scene-Ansicht wird während der Entwicklung benutzt. Die GameObjects können in Echtzeit hinzugefügt und verändert werden.

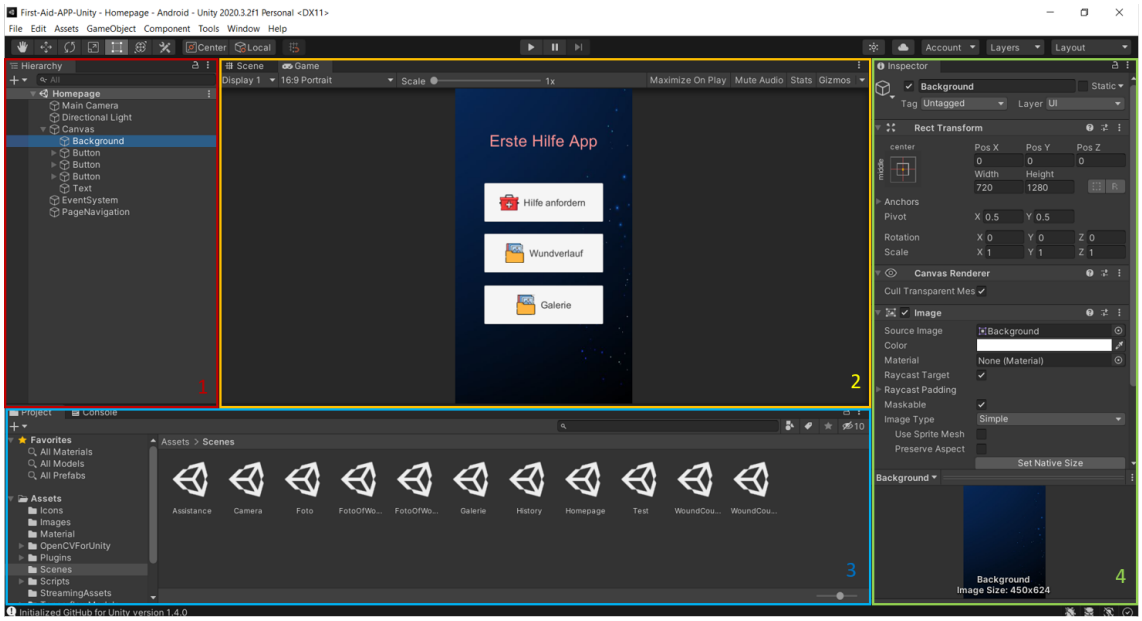


Abbildung 2.1: Unity Entwicklungsumgebung

Die *Game-Ansicht* ist die Vorschau. Das Spiel oder die App wird aus der Anwendersicht dargestellt. Kontroll- und Steuerelemente, die für die Entwicklung relevant sind, werden ausgeblendet und GameObjects können nicht manipuliert werden. Zusätzlich kann die Anwendung gestartet werden, um eine Echtzeitvorschau zu bekommen. Sobald der Play-Mode aktiviert ist, verhält sich die App bzw. das Spiel genauso, wie es später auf den Geräten bereitgestellt wird. Dabei kann der Anwender mit den GameObjects interagieren.

Das mit 4 gekennzeichnete Fenster ist das Projektfenster und zeigt alle zum Unity-Projekt gehörenden Dateien und Assets an. Mit dem Projektfenster kann die Struktur der Unity-Anwendung verwaltet werden. Es bietet die Möglichkeit, Dateien in das Projekt per Drag-and-drop zu laden, eine Suche durchzuführen sowie C#-Skripte und Szenen zu erstellen.

Auf der rechten Seite von Unity wird das Inspektorfenster angezeigt. Das Inspektorfenster ändert sich in Abhängigkeit vom gewählten GameObject, von der selektierten Unity-Komponente oder vom Asset. Mit ihm können die Einstellungen und Eigenschaften verändert werden. „Ein Unity-Asset ist ein Objekt, das man im Spiel oder Projekt verwenden kann. Ein Asset kann von einer Datei stammen, die außerhalb von Unity erstellt wurde, beispielsweise ein 3D-Modell, eine Audio-Datei, ein Bild oder jede andere Art von Datei, die Unity unterstützt. Es gibt auch Assets, die innerhalb von Unity erstellt werden können, wie z. B. ein Animator-Controller, ein Audio-Mixer oder eine Render-Textur.“ [14] Unity stellt auch einen Asset-Store bereit, aus dem sowohl kostenlose als auch kostenpflichtige Inhalte heruntergeladen und in das Projekt importiert werden können.

Der Asset-Store kann auch direkt auf Unity geöffnet werden.

2.1.1 GameObjects

GameObjects sind zentrale Elemente in Unity. Sie fungieren als Container und können mit anderen GameObjects verschachtelt werden. Der Begriff ‚GameObject' wird aber auch an anderen Stellen, z. B. bei C#-Skripten, verwendet, so dass die deutsche Übersetzung ‚Spieleobjekt' an diesen Stellen benutzt wird, um die Begriffe klar zu trennen. Alle Objekte einer Szene sind GameObjects. Spieleobjekte sind nicht in anderen Szenen wiederverwendbar und werden individuell in jeder Szene einzeln erstellt. *Prefabs* bieten die Möglichkeit, Spieleobjekte wiederzuverwenden [54]. Zu jedem GameObject können Komponenten über den Inspektor hinzugefügt werden. Komponenten steuern und beeinflussen das Verhalten von GameObjects. Jedem GameObject ist die Komponente *Transform* zugeordnet. Diese speichert unter anderem Position, Rotation und Skalierung des Objects. Weitere Komponenten sind z. B.:

- C#-Skripte
- User Interface (UI)-Elemente wie Buttons, Bilder oder Text
- Layout-Elemente wie Canvas

Ein GameObject kann über das Hierarchiefenster erstellt werden. Es ist ebenfalls möglich, dass GameObjects während der Laufzeit über ein Skript generiert und instanziiert werden. Zusätzlich kann ein gesamtes GameObject inklusive aller Komponenten kopiert werden. Es kann dabei ein völlig leeres GameObject ohne weitere Komponenten oder ein bestimmtes GameObject mit der entsprechenden Komponente erstellt werden. Beispielsweise kann ein Button generiert werden, der die Komponenten *Image* und Click-Listener automatisch besitzt.

Ein GameObject kann den Status ‚Aktiv 'und ‚Inaktiv' haben. Beim inaktiven Zustand werden keine zum Spieleobjekt hinzugefügten Skripte ausgeführt. Jedem Spieleobjekt kann ein bestimmter Tag über den Inspektor zugewiesen werden. Ein Tag kann beliebig vielen GameObjects zugewiesen werden und hilft dabei, Spieleobjekte zu gruppieren. Ein prominentes Beispiel für die Verwendung ist, dass alle GameObjects neu erstellt werden müssen, nachdem ein Spieler in einem Level gestorben ist. Dazu können alle relevanten GameObjects mit dem Tag *Respawn* versehen werden. Ein Skript erstellt daraufhin alle Spieleobjekte mit diesem Tag.

2.1.2 C#-Skripte

Unity stellt Funktionen und Werkzeuge für viele Probleme, die bei der Entwicklung von Spielen und Apps entstehen, out of the box bereit. Für den Fall, dass der Werkzeugkasten von Unity nicht ausreichend ist, können die Fähigkeiten von Unity mit C#-Skripten erweitert werden. [vgl. 6, S. 327]

Um ein Skript zu erstellen, kann das Projektfenster oder der Inspektor eines GameObjects benutzt werden. Es wird empfohlen, Pascal-Case für die Namensgebung der Skripte zu verwenden [vgl. 6, S. 331]. Wenn das Skript über den Inspektor erstellt wurde, ist es direkt mit dem Spieleobjekt verbunden. Für die Entwicklung des Skripts wird ein externer Editor verwendet. Mit der Installation von Unity kann Visual Studio direkt mit installiert werden. Es ist aber auch möglich, jeden anderen C# Editor zu benutzen. Unity erstellt automatisch eine Klasse mit zwei Methoden. Die Klasse erbt automatisch von der Klasse *MonoBehaviour*. *MonoBehaviour* stellt die Basisklasse von Unity dar. Sie ermöglicht es, dass

ein Skript mit einem Spieleobjekt verbunden werden kann. Der Lebenszyklus des Skriptes bzw. der Klasse wird komplett durch Unity verwaltet. Der gesamte Lebenszyklus ist in Anlage A dargestellt.

Das Unity-Event-System ist für das Verarbeiten von Events verantwortlich. Es ist einfach und flexibel gestaltet. Einen Teil des Event-Systems stellen die *Event Functions* dar. Diese sind Methoden, die von der Klasse *MonoBehaviour* geerbt werden. Beispiele hierfür sind:

Start() Die Methode dient zur Initialisierung der Klasse. Die Methode wird von Unity aufgerufen, sobald das Spieleobjekt erstellt wurde und die Szene geladen wurde.

Awake() Im Gegensatz zur *Start-Methode* wird *Awake()* beim Laden der Szene einmalig aufgerufen oder dann, wenn ein GameObject von inaktiv auf den aktiven Zustand wechselt. *Awake()* wird unmittelbar nach dem Initialisieren der Spieleobjekte aufgerufen. Das stellt sicher, dass Spieleobjekte mittels einer Find-Funktion, die eventuell in der *Awake()* benutzt wird, funktionieren. Zu beachten ist, dass die Aufrufe durch Unity nicht deterministisch sind [53].

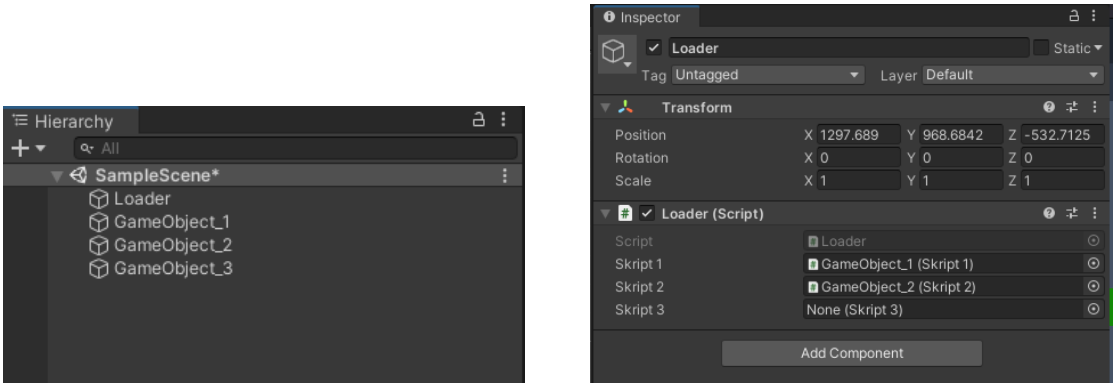
Update() Die Update-Methode ist eine der meist genutzten Methoden. Sie wird für jedes Frame ausgeführt, wenn das GameObject aktiv ist. Dies wird durch die eingestellten Frames per second (FPS) geregelt. Bei 60 FPS wird die Methode genau 60 mal pro Sekunde ausgeführt.

OnDestroy() Die *OnDestroy()-Methode* wird ausgeführt, sobald ein GameObject zerstört wird. Dies kann erfolgen, wenn eine neue Szene geladen wird. Die Methode kann verwendet werden, um die Webcam wieder freizugeben, wenn die Szene verlassen wurde.

Um die Entwicklung eines Spiels und von Apps zu vereinfachen, bietet Unity die Möglichkeit an, Skripte im Unity-Editor anzupassen. Der Spieleentwickler kann die öffentlichen Felder eines Skripts festlegen. Die Felder sind die normalen Datentypen von C#. Der Vorteil ist, dass der Entwickler nicht den Quellcode ändern muss. Dazu ist es notwendig, dass ein Feld public ist. Unity zeigt automatisch das Feld im Inspektor unter dem entsprechenden Skript an. Die Darstellung verändert sich mit dem Datentyp des Feldes. Bei einem booleschen Feld wird beispielsweise eine Checkbox angezeigt.

2.1.3 Verdrahtung von GameObjects

Im Inspektor können aber nicht nur Felder der Datentypen von C# verändert werden. Er dient auch zum Verdrahten der mit den GameObjects verknüpften Skripte. Die Verdrahtung ist notwendig, wenn es Abhängigkeiten zwischen den Spieleobjekten gibt. Das Konzept verfolgt einen Dependency Injection (DI) Ansatz. Unity übernimmt die Verwaltung und Instanziierung aller Klassen, die von der Klasse *MonoBehaviour* erben. Es sollte davon abgesehen werden, diese Klassen selbst zu instanziieren. In Abbildung 2.2.a sind vier GameObjects dargestellt. Das Spieleobjekt *Loader* hat Abhängigkeiten zu den drei weiteren Spieleobjekten. Alle GameObjects enthalten ein C#-Skript, wobei im Skript des Spieleobjekts *Loader* drei separate Klassen im Klassenblock definiert sind. Der Inspektor des GameObjects *Loader* ist in Abbildung 2.2.b dargestellt. Der Inspektor zeigt als Namen den Klassentyp an, der benötigt wird. Die eigentliche Verdrahtung wird mittels Drag-and-drop durchgeführt. Dazu wird das GameObject in das entsprechende Feld im Inspektor gezogen. Unity lässt nur eine Verdrahtung zu, wenn sich die richtige Klasse im GameObject befindet. In Abbildung 2.2.b wurden *Skript1* und *Skript2* richtig verdrahtet. Bei *Skript3* ist noch keine Verdrahtung erfolgt und das Feld zeigt *None* an.



(a) Hierarchiefenster (b) Inspektor

Abbildung 2.2: Verdrahtung von GameObjects in Unity

2.1.4 gameObject

Es ist ebenfalls möglich, direkt auf das Spieleobjekt sowie auf dessen Komponenten von einem Skript zuzugreifen. Dazu kann das von der Klasse *MonoBehaviour* geerbte Property *gameObject* benutzt werden. *gameObject* ist von der statischen Klasse *GameObject* zu unterscheiden (siehe 2.1.5). *gameObject* ist eine Komponente und ermöglicht den Zugriff auf die Instanz des Spieleobjekts, mit dem es verbunden ist. *gameObject* kann z. B. verwendet werden, um ein Spieleobjekt um neunzig Grad zu drehen. Dazu wird zuerst im Hierarchiefenster ein leeres Spieleobjekt angelegt und eine Text-Komponente wird hinzugefügt. Letzteres kann über den Button „Add Component“ erreicht werden.

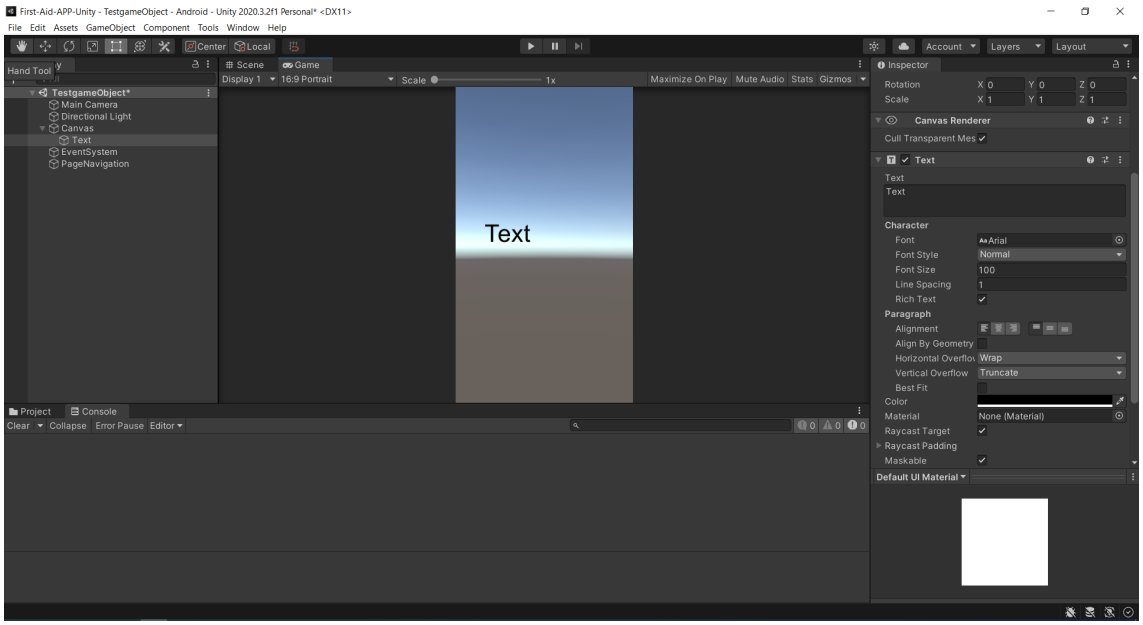


Abbildung 2.3: Erstellung eines Spieleobjekts und Manipulation mit einem Skript

Nach dem Hinzufügen der Text-Komponente können der Text, die Schriftfarbe und die Schriftgröße angepasst werden. Abbildung 2.3 zeigt das eben erstellte Spieleobjekt. Zum Schluss wird ein Skript an das Spieleobjekt gebunden. Das Skript kann über den Button „Add Component“ im Inspektor hinzugefügt werden. Es wird dann erstellt und automatisch verknüpft. Es ist ebenfalls möglich, das Skript über das Projektfenster zu

erstellen und dann per Drag-and-drop auf das Spieleobjekt zu ziehen, um es zu verknüpfen. Der in Listing 2.1 dargestellte Ausschnitt aus dem Skript zeigt die Verwendung. Es wird zuerst ein Vektor initialisiert. Dieser beinhaltet drei Parameter für die Werte der X-, der Y- und der Z-Achse. Über das *gameObject* kann mit der Rotate-Methode auf die Transform-Komponente zugegriffen werden.

Listing 2.1: Rotation von GameOjects

```
void Start()
{
    Vector3 vector = new Vector3(0, 0, 90);
    gameObject.transform.Rotate(vector);
}
```

Die Rotate-Methode nimmt den Vektor als Parameter entgegen. Sobald die App gestartet wurde und die Szene geladen wurde, wird die Start-Methode ausgeführt und der Text in der Szene wird rotiert.

2.1.5 Klasse GameObject

Wie im letzten Abschnitt beschrieben, kann man mithilfe eines Skriptes auf ein Spieleobjekt zugegriffen werden. Die statische Klasse *GameObject* kann das Gleiche, muss jedoch nicht zwingend dem gleichen Spieleobjekt zugewiesen sein. Zuvor muss allerdings nach dem Spieleobjekt gesucht werden. Es gibt mehrere Möglichkeiten wie GameObjects gefunden werden können. Alle Möglichkeiten können als Methode der Klasse *GameObject* aufgerufen werden.

Zwei der Möglichkeiten sind die Methoden *FindGameObjectWithTag* und *FindGameObjectsWithTag*. Die Methoden finden ein Spieleobjekt anhand seines Tags. Beiden Methoden muss ein String übergeben werden. Der String ist der Tag des Spieleobjekts. Die Methode *FindGameObjectWithTag* gibt das erste gefundene Spieleobjekt zurück. Es wird ein Objekt vom Typ *GameObject* zurückgegeben. Im Gegensatz dazu gibt *FindGameObjectsWithTag* ein Array zurück. Für den Fall, dass kein Spieleobjekt gefunden werden kann, wird ein leeres Array zurückgegeben. Zu beachten ist, dass mit der Verwendung der Methode sparsam umgegangen werden sollte. Die Methode durchsucht alle GameObjects, was sich negativ auf die Perfomance auswirkt. Eine andere Methode, um ein Spieleobjekt zu suchen, ist die Methode *Find*. Sie nimmt einen String als Übergabeparameter entgegen, der den Namen des Spieleobjekts darstellt.

2.1.6 UI-Elemente

Unity stellt eine Reihe von vordefinierten Graphical User Interface (GUI)-Elementen bereit, mit denen einfach eine Benutzeroberfläche erstellt werden kann. Sie sind normale Spieleobjekte mit den entsprechenden Komponenten. Die Elemente umfassen z. B. Button, Scrollbar, Image, Eingabefelder und Text. Die UI-Elemente können im Hierarchiefenster erstellt werden. Das Element *Button* besitzt eine Komponente mit demselben Namen. Jeder Button besitzt eine Liste von Click-Events, die hinzugefügt werden können. Nach dem Hinzufügen eines Click-Events, muss angegeben werden, welche Methode in welchem Skript beim Event ausgeführt wird. Das passiert ähnlich wie die Verdrahtung der GameObjects. Mit Drag-and-drop kann ein GameObject in das entsprechende Feld gezogen werden. Daraufhin können über eine Liste erst die Klasse und danach die Methode ausgewählt werden, die ausgeführt werden sollen. Das Hinzufügen eines Bildes funktioniert ähnlich. Zuerst wird ein UI-Element vom Typ *Image* erstellt. In der Komponente

Image gibt es ein Feld mit dem Namen Texture. Texturen werden in Abschnitt 2.1.7 erklärt. Danach kann ein Bild-Asset vom Projektfenster in das Feld gezogen werden.

Ein weiteres UI-Element ist das Canvas. Das Canvas beinhaltet alle anderen UI-Elemente. Der Canvas-Bereich wird in der Szenenansicht als Rechteck dargestellt und ist der Bereich der später auf dem Gerät gerendert wird. Elemente werden nur angezeigt, wenn sie sich als untergeordnetes Element im Canvas befinden.

2.1.7 Texture

In Unity sind Texturen Bild- oder Filmdateien, die über Spieleobjekte gelegt werden können. Für 3D-Objekte kann eine Textur auch ein GameObject komplett umhüllen. Es entsteht dabei ein visueller Effekt. Texturen können PNG- oder auch JPG-Dateien sein. Für die Verwendung in Skripten stellt Unity die Klasse *Texture2D* bereit. Die Klasse ist für 2D Texturen und es gibt ebenfalls eine Klasse für 3D-Texturen. Beim Instanziieren müssen die Höhe und die Breite angegeben werden.

Texturen, vor allem Bilddateien, haben ein bestimmtes Texturformat. Unity unterstützt über fünfzig verschiedene Formate [65]. Ein Beispiel für ein Texturformat ist *RGBA32*. *RGB* steht dabei wie üblich für den Rot-Grün-Blau-Anteil jedes Pixels eines Bildes. *a* ist der Alpha-Channel und gibt die Deckkraft bzw. die Transparenz eines jeden Pixels an. Die Zahl 32 gibt die Farbtiefe an. Es stehen jeweils 8 Bit für RGB und für den Alpha-Channel zur Verfügung.

2.1.8 Webcam

Für AR oder VR Anwendungen ist es notwendig, dass Unity auf die Webcam zugreifen kann. Unity stellt die Klasse *WebCamTexture* für die Verarbeitung des Kamerastreams bereit. In Listing 2.2 ist der Quellcode dargestellt, der den Live-Feed einer Kamera auf ein Image rendert. Zuerst werden alle verfügbaren Geräte in ein Array geschrieben. Bei einem Smartphone können das die Front- und die Backkamera sein. Als Nächstes wird das Objekt *cameraTexture* instanziiert. Dem Konstruktor können mehrere Parameter übergeben werden. Es kann das Quellgerät, wie im Listing verwendet, angegeben werden. Ebenso können eine bestimmte Breite und Höhe angefordert werden. Diese können jedoch nicht garantiert werden und sind geräteabhängig [55].

Listing 2.2: WebcamTexture

```
public class PhoneCamera : MonoBehaviour
{
  private WebCamTexture cameraTexture;
  public RawImage background;

  void Start()
  {
    WebCamDevice[] devices = WebCamTexture.devices; //Get all devices
    cameraTexture = new WebCamTexture(devices[0].name);
    cameraTexture.Play(); // Start the camera
    background.texture = cameraTexture; // Set the texture
   }
  ...
}
```

Im nächsten Schritt wird mit der Methode *Play()* die Kamera gestartet und die *WebCamTexture* wird dem Image zugewiesen. Die Kamera rendert nun korrekt auf das Image.

WebCamTexture besitzt nützliche Methoden und Properties. Die Methode *Stop()* sollte ausgeführt werden, wenn die Kamera nicht mehr benötigt wird und freigegeben werden kann. Der Aufruf kann z. B. in der Methode *OnDestroy()* erfolgen. Das Property *videoRotationAngle* gibt einen Winkel in Grad zurück, der dazu verwendet werden kann, die Orientierung des Images zu ändern, wenn das Smartphone gedreht wird.

2.2 OpenCV und OpenCV for Unity

OpenCV ist eine Open-Source-Library für Computer Vision. OpenCV wurde in C und C++ geschrieben und unterstützt mehrere Sprachen für die Entwicklung wie Python oder C++. OpenCV setzt einen starken Fokus auf die Entwicklung von Echtzeitanwendungen. Ein Kernaspekt von OpenCV ist, dass eine einfach zu nutzende Infrastruktur bereitgestellt wird, mit der eine schnelle Entwicklung von Anwendungen im Bereich Computer Vision erreicht werden kann. [vgl. 7, S. 1]

Computer Vision ist ein Teilgebiet der KI und steht mit dem Teilgebiet des Deep Learnings in engem Zusammenhang. Computer Vision hat das Ziel, Informationen aus Bild- und Videoquellen zu gewinnen [vgl. 43, S. 10]. Das Gebiet unterteilt sich wiederum in verschiedene Teilbereiche wie Object-Detection und -Localization, Object-Segmentation sowie Pose-Estimation.

Die Firma Enox Software bietet im Asset Store ein Plugin an [41]. Das Plugin ist ein Klon von OpenCV Java und unterstützt viele Deep-Learning-Frameworks wie ONNX, TensorFlow oder Darknet. OpenCV for Unity bietet auch eine Konvertierung von Unity Texture2D zu OpenCV for Unity Mat (Mat siehe Abschnitt 2.2.1).

2.2.1 Mat

OpenCV hat eine spezielle Datenstruktur für die Speicherung und für die Verarbeitung von Bildern und Matrizen. In OpenCV for Unity wird die Datenstruktur durch die Klasse *Mat* dargestellt und ist eine Schlüsselkomponente. Im Kern ist die Klasse eine für Computer Vision optimierte Matrix. [vgl. 38, S. 24]

Die Datenstruktur unterstützt eine effiziente Speicherverwaltung, in dem sie aus zwei Teilen besteht. Diese Teile umfassen den Header und den Datablock. Im Header befinden sich die Metainformationen wie die Anzahl der Zeilen, der Spalten und der Channels. Der Datablock beinhaltet die eigentlichen Daten. Abbildung 2.4 zeigt abstrakt ein Mat eines Bildes. Die Achsen „Zeilen“ und „Spalten“ sind üblich für eine Matrix. Das Farbbild setzt sich aus den Werten Rot, Grün und Blau zusammen. In OpenCV wird mit Channels die Tiefe eines Bildes bezeichnet. In der Abbildung hat das Mat des Farbbildes drei Channels. Vier Channels werden z. B. bei Bildern genutzt, die mit einer Tiefenbildkamera gemacht wurden. Beim Erstellen eines *Mat* kann der Typ angegeben werden, der die Größe einer jeden Zelle, z. B. 8 Bit, sowie die Anzahl der Channels festlegt. OpenCV stellt dafür Typen bereit. Diese folgen einer Namenskonvention. Der Typ *CV_8UC1* steht für 8 Bit **u**nsigned und 1 **C**hannel, während der Type *CV_32FC2* eine 32 bit *F*loat-Zahl mit 2 *C*hannels ist.

Mat unterstützt auch eine Reihe von Operationen. Neben arithmetischen Operationen kann auch die Matrix umgeformt werden. Dazu wird die Methode *reshape* bereitgestellt. Eine Umformung ist oft notwendig, um die Vorhersagen der neuronalen Netze nutzen zu können. Die Methode nimmt zwei Parameter entgegen. Der erste Parameter ist die Anzahl der neuen Zeile und der zweite Parameter ist die neue Zeile. Zentral ist dabei, dass die Anzahl der Elemente in der Matrix unverändert bleibt. Durch die smarte Speicherverwaltung ist kein Speicherzugriff notwendig [vgl. 38, S. 50].

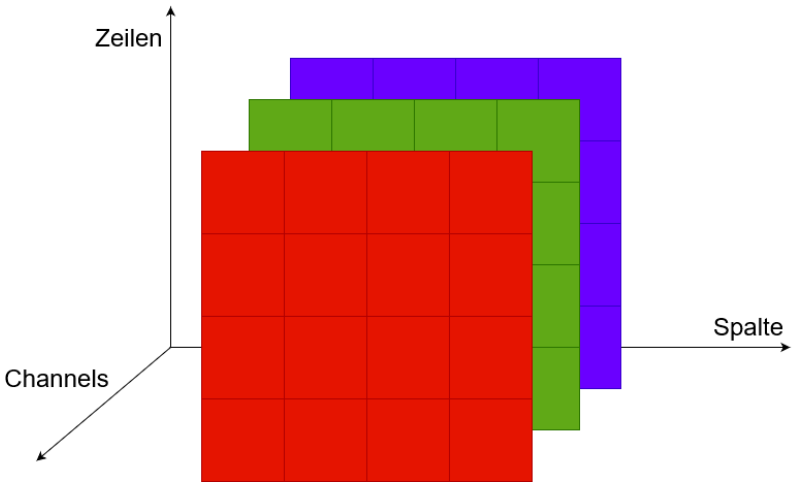


Abbildung 2.4: Grafische Darstellung von Mat

OpenCV for Unity arbeitet intern nur mit Mats. Es ist notwendig, ein Bild, also eine Unity-Texture oder eine WebcamTexture, in ein Mat umzuwandeln. OpenCV for Unity besitzt dafür die eigene Methode, *webCamTextureToMat* in der statischen Klasse *Utils*. Bevor diese Methode benutzten werden kann, muss ein neues Mat erstellt werden. Dem Konstruktor ist die Höhe, die Breite sowie der CvType, z. B. CV_8UC1, zu übergeben. Es ist zwingend erforderlich, dass der CvType von OpenCV und das Texturformat von Unity übereinstimmt. Ansonsten kann keine Konvertierung stattfinden. Ein korrektes Matching ist beispielsweise CV_8UC4 und RGBA32.

2.2.2 Deep Neural Network (DNN)

In OpenCV können alle gängigen Deep-Learning-Modelle verwendet werden. Die Modelle sind bereits trainiert. Mit OpenCV können keine Modelle trainiert werden. Alle relevanten Funktionen befinden sich im Modul *DnnModule*. Abbildung 2.5 zeigt skizziert den Ablauf von der Initialisierung bis zum Post-Processing.

Zuerst erfolgt die Initialisierung. Die Klasse *Net* wird benutzt, um das Modell zu importieren und um die Vorhersagen zu tätigen. Es werden die Default-Parameter festgelegt. Diese umfassen z. B. die Höhe und die Breite des Mats, die das Modell erwartet. Danach wird das Modell geladen. Die Klasse *Dnn* hat jeweils eine eigene Methode zum Laden der Modelle von den verschiedenen Frameworks. Für TensorFlow Modelle kann die Methode *readNetFromTensorFlow* benutzt werden. Die Methode ist überladen. Es können z. B. zwei Dateien übergeben werden. Die erste Datei ist eine Protobuf (pb)-Datei. Sie basiert auf Googles Protocol-Buffers [44] und enthält die Architektur des Modells und die Gewichte. Zum anderen benötigt OpenCV zusätzlich die Konfiguration. OpenCV for Unity unterstützt nur einen *Frozen Graph*. Ab TensorFlow 2.5 kann ein Modell nur noch im *SavedModel*-Format gespeichert werden [51]. Folglich muss eine ältere Version benutzt werden. Im nächsten Block wird ein Image Processing durchgeführt. Eine Texture2D oder eine WebCamTexture wird als Quelle für die Umwandlung benutzt. Die Größe wird ebenfalls aus der Quelle übernommen. Dem Konstruktor muss auch der richtige CV_Typ übergeben werden. Danach wird mit der Methode *webCamTextureToMat* die WebCamTexture in ein Mat umgewandelt.

Als Nächstes wird ein Binary Large Objects (Blob) vom Mat erstellt. Die Größe des Bildes wird auf die vom Modell benötigte Größe angepasst. Es kann noch ein Skalar angegeben werden und die Reihenfolge der Farbordnung, also z. B. RGB oder BGR, kann

verändert werden. Danach wird der Blob in das Modell gegeben. Die Methode *forward()* führt die Vorhersage bzw. die Klassifizierung durch. Die Methode gibt ein Mat zurück, das im Grunde ein Array mit dem Ergebnis ist.

Nachdem das Bild durch das Modell geschickt wurde, muss meistens noch ein Post-Processing durchgeführt werden. Das Mat kann selten direkt weiterverarbeitet werden. Das Post-Processing unterscheidet sich stark, je nachdem welche Form der Output hat. Die Methoden *reshape* und *get* der Klasse *Mat* eignen sich dazu. Mit der Methode *get* kann auf den Inhalt des Mats zugegriffen werden. Es werden die Spalte und die Reihe übergeben. Als dritter Parameter muss ein Array des Typs short, int, float oder double übergeben werden. Der Inhalt im Mat wird dann in dieses Array gespeichert.

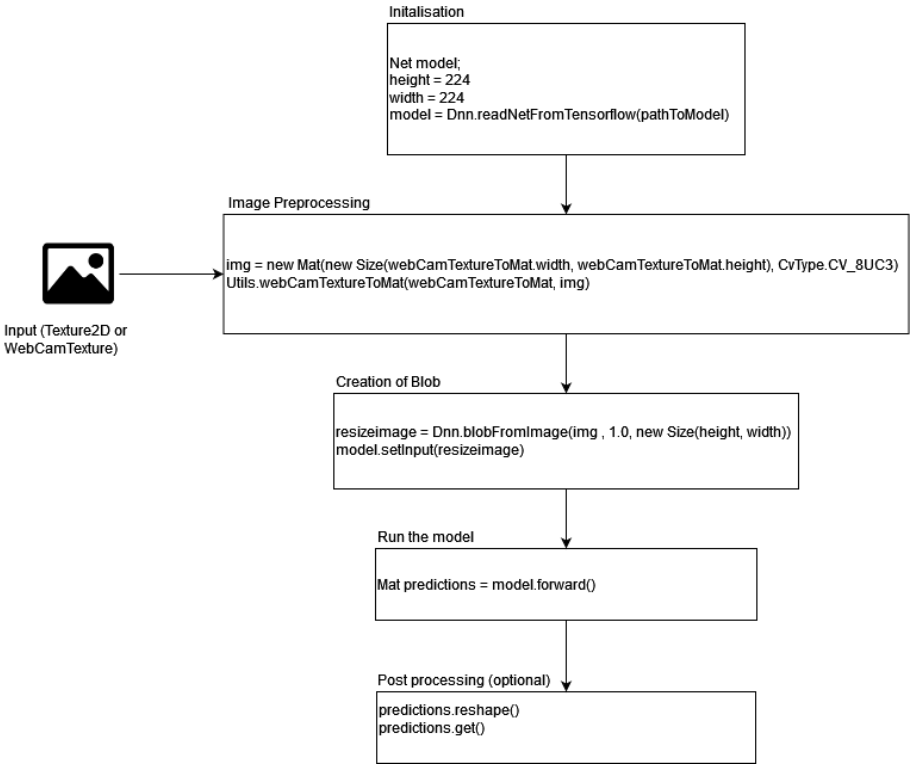


Abbildung 2.5: Ablauf Integration DNN in OpenCV for Unity

2.3 Objekterkennung

Im allgemeinen kann zwischen den verschiedenen Aufgaben in Computer Vision unterschieden werden. Diese sind unter anderem Klassifizierung, Klassifizierung mit Lokalisierung und Objekterkennung. Bei einer Klassifizierung gibt das Deep-Learning-Modell nur eine bestimmte Klasse aus, die das Modell erkannt hat. Ein Bild mit einem Hund erzeugt den Output *Hund*. Es ist dabei zu beachten, dass nur eine Klasse vorhergesagt wird. Eine Vorhersage eines Bildes mit einem Hund und einer Katze ist nicht möglich. Bei einem Bild mit einem Hund und einer Katze, wird das Modell die Klasse mit der höchsten Wahrscheinlichkeit klassifizieren bzw. vorhersagen. Bei einer Klassifizierung mit Lokalisierung wird zusätzlich das erkannte Objekt im Bild markiert. Üblich ist es, eine Box, auch Bounding Box genannt, um das Objekt zu zeichnen. Die Objekterkennung kann mehrere Klassen, also einen Hund und eine Katze, im Bild erkennen. Auch hier ist es üblich, Boxen um die erkannten Objekte zu ziehen. Die Farben der Boxen variieren je nach Klasse. Zusätzlich werden in der Regel die Klasse sowie der Confidence-Wert über

jeder Box angezeigt. Der Confidence-Wert gibt die Wahrscheinlichkeit an, mit der das Modell sicher ist, die Klasse erkannt zu haben.

2.4 MobileNet

Die Ursprünge von MobileNet gehen auf einen im Jahre 2017 veröffentlichten Artikel zurück [35]. Der Artikel stammt von Google und stellt eine neue Architektur für ein Deep-Learning- Modell vor, die speziell für mobile Endgeräte und für die Bildverarbeitung, insbesondere für die Klassifizierung von Bildern, optimiert ist. Die Grundlage von MobileNet ist ein Convolutional Neural Network (CNN).

Die Idee dabei ist, Konvolutionsschichten (‚convolutional layer,) , die teuer bei der Berechnung sind, durch eine tiefenorientierte, trennbare Konvolution [depthwise separable convolution (DeptwiseSepConv)] zu ersetzen. Der DeptwiseSepConv besteht aus einer Tiefenkonvolution und einer 1×1 Faltung, auch punktuelle Konvolution (‚pointwise convolution,) genannt. Der Vorteil ist, dass die Anzahl der Parameter bei diesem Ansatz erheblich reduziert wird. Die Ersparnis beträgt 25,1 Millionen Parameter bei einer Reduzierung der Genauigkeit um 1 Prozent [35, S. 5]. Im Jahr 2018 wurde Version 2 [50] herausgegeben und 2019 wurde Version 3 [34] veröffentlicht.

TensorFlow bietet die Möglichkeit, MobileNet Architekturen direkt herunterzuladen. Mit dem Aufruf tf.keras.applications.mobilenet.MobileNet() kann das Modell heruntergeladen werden. Es können zehn verschiedene Parameter übergeben werden, die das Verhalten des Modells beeinflussen. Im Folgenden werden einige Parameter näher vorgestellt:

weights Mit *imagenet* wird angegeben, ob das Modell vortrainiert heruntergeladen wird. Das Modell wurde mit dem ImageNet-Datensatz [37] trainiert.

include_top Boolescher Wert, der angibt, ob am Ende des Modells ein fully-connected (FC) Layer verwendet wird.

classes Die Anzahl der Klassen, die klassifiziert werden können. Der Parameter kann nur gesetzt werden, wenn include_top auf True gesetzt ist. Der Standardwert ist 1000, wenn kein Parameter gesetzt wurde.

classifier_activation Spezifiziert die Aktivierungsfunktion. Diese kann die Werte *None* oder *softmax* haben. Der Parameter wird ignoriert, wenn *include_top* auf False gesetzt wurde. Wenn der Wert auf *None* gesetzt wurde, werden die Logits ausgegeben. Bei *softmax* wird die Softmax-Aktivierungsfunktion benutzt.

2.4.1 Architektur

In Abbildung 2.6 ist die Architektur eines MobileNets skizziert und in Anlage B ist eine detaillierte Übersicht inklusive Typ, Stride, Filter Shape sowie Eingabegröße der verschiedenen Layers zu finden. MobileNet erfordert ein Bild im Format $224 \times 224 \times 3$, wobei jeweils 224 die Höhe und die Breite ist und drei Channels für RGB benötigt werden. MobileNet hat insgesamt 28 Layers, wenn der Depthwise-Convolution-Layer und der Pointwise-Convolution-Layer als getrennte Layer gezählt werden.

Die gesamte Architektur setzt sich aus einer Reihe von Convolutional-, DeptwiseSepConv-, Pooling- und einem FC-Layer zusammen. Der erste Layer ist ein FC-Layer gefolgt von einem Depthwise-Convolution-Layer. Der Depthwise-Convolution-Layer führt eine räumliche Faltung durch. Besonders ist dabei, dass jeder Channel separat betrachtet wird. Zu

diesem Zeitpunkt hat das Modell noch keine Korrelation zwischen den Channels hergestellt. Dazu ist es notwendig, die punktuelle Konvolution durchzuführen. [vgl. 9, S. 5] Die Faltung passiert also über die verschiedenen Channels hinweg. Sowohl nach dem Depthwise-Convolution-Layer als auch dem Pointwise-Convolution-Layer kommen jeweils ein Batch Normalization Layer sowie eine Rectified Linear Unit (ReLu).

Dieser Vorgang wird mehrmals wiederholt und die Anzahl der Filter variiert je nach Fortschritt. Die räumliche Größe reduziert sich dabei und die Anzahl der Channels nimmt zu. Das Flattening wird mit dem Pooling-Layer und dem FC-Layer erreicht. Der Pooling-Layer führt ein Average Pooling mit einer Filter-Shape von 7×7 durch. Der FC-Layer hat 1024 Neuronen und verbindet den Flatten-Layer mit dem Output Layer. Der Flatten-Layer enthält dabei die extrahierten Features, also die Merkmale, mithilfe derer die Klasse eines Bildes bestimmt werden kann. Als Aktivierungsfunktion wird Softmax benutzt.

Die Softmax-Funktion ist die übliche Funktion, die gewählt wird, wenn es sich um eine Mehrfachklassifizierung handelt. Im Gegensatz zu anderen Aktivierungsfunktionen, werden die Wahrscheinlichkeiten einer jeden Klasse als Ausgabe gegeben. Die Summe aller ausgegebenen Wahrscheinlichkeiten beträgt dabei immer 1. Sie ist wie eine normierte logistische Funktion anzusehen. [vgl. 47, S. 447]

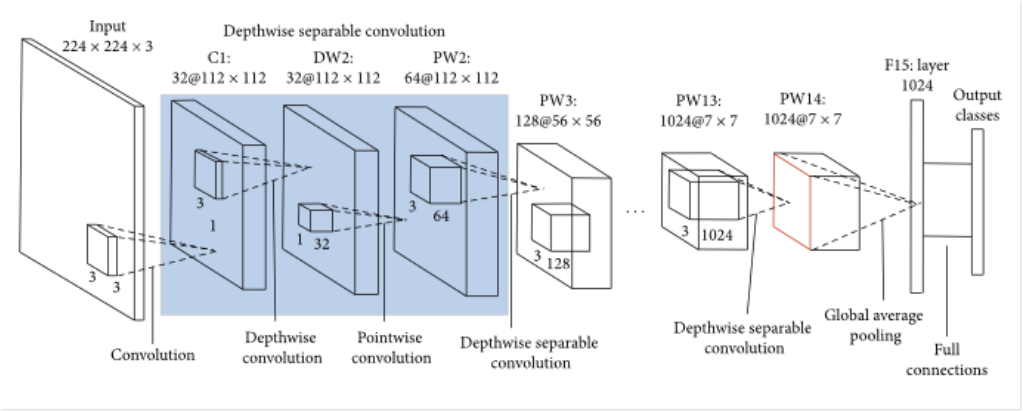


Abbildung 2.6: Architektur MobileNet [46]

2.5 Single-Shot-MultiBox-Detector

Während das MobileNet nur eine Möglichkeit zur Klassifizierung ist, gibt es andere Modelle, die für die Objekterkennung spezialisiert sind. Eine Architektur für diese Erkennung ist Single Shot MultiBox Detector (SSD). Dieser wurde im Jahre 2016 im Rahmen eines wissenschaftlichen Artikels vorgestellt [67]. SSD nutzt ein anderes Modell für die Objektklassifizierung als Basis, um die Feature-Extraction durchzuführen. Im ursprünglichen Artikel ist dies ein Visual Geometry Group 16 (VGG16) Modell. VGG16 kann mit jedem beliebigen Modell ausgetauscht werden. SSD verfügt über sechs Layers und es wird für jede Klasse eine feste Anzahl Bounding-Boxes bestimmt. Dadurch entstehen viele Boxen unterschiedlichster Größen und Seitenverhältnisse. Nach der Bestimmung der Boxen erfolgt die sogenannte Non-Max Suppression. Jeder Box ist der Confidence-Wert zugeordnet. Es müssen alle Boxen mit einem niedrigeren Wert als das Maximum entfernt werden. Das Problem dabei ist, dass nur Boxen entfernt werden können, die zu einem bestimmten Objekt auf dem Bild gehören. Dafür kann Intersection Over Union (IoU) benutzt werden.

IoU beschreibt das Verhältnis zwischen dem Wert der Überlappung und dem gesamten

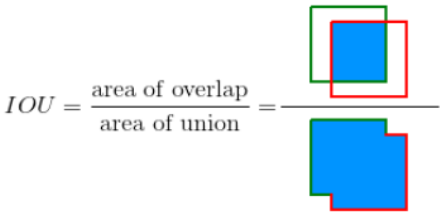


Abbildung 2.7: Intersection Over Union [42]

Bereich der zwei Boxen. Das Verhältnis ist der IoU-Wert und ein Hyperparameter von SSD. Normalerweise wird der Wert auf 0,5 gesetzt. SSD betrachtet nun alle Boxen, die einen gemeinsamen IoU-Wert größer als 0,5 haben, als zusammenhängend und behält nur noch die Box mit dem größten Confidence-Wert.

2.6 TensorFlow Object Model API

Die TensorFlow Object Detection API (TFOD) ist ein Open-Source -Framework zum Erstellen, zum Trainieren und zum Bereitstellen von Modellen für die Klassifizierung und für die Object Detection. Die API ist für TensorFlow 1 und TensorFlow 2 verfügbar, wobei TensorFlow 1 deprecated ist und nicht mehr weiterentwickelt wird. Die Versionen sind nicht untereinander kompatibel. Verfügbar ist die API auf GitHub [61]. Im Model Zoo befinden sich verschiedene Modelle mit unterschiedlichen Werten für die Hyperparameter. Unter anderem sind folgende Modelle im Model-Zoo verfügbar:

- MobileNet V1
- MobileNet V2
- Region Based Convolutional Neural Networks (RCNN)
- Residual Neural Network (ResNet)

Alle Modelle sind mit unterschiedlichen Datensätzen, z. B. mit dem, Common Objects in Context (COCO) Datensatz [64] oder dem Open-Images-Dataset [37], vortrainiert. Die API kann ebenfalls in einem Docker bereitgestellt werden. Allerdings unterstützt Microsoft nur eine Verwendung der Graphics Processing Unit (GPU) für Windows Container. Folglich kann keine Grafikkarte im Container benutzt werden, um die Modelle zu trainieren. Für die Installation ist eine Reihe von Schritten notwendig. Zuerst können Ordner erstellt werden, die die gesamten Daten, wie die Modelle, die Bilddateien oder die Skripte enthalten. Best Practice ist, die Ordner nach der folgenden Struktur aufzubauen:

addons Hier können Add-ons wie LabelImg oder Skripte abgelegt werden, die für mehrere Projekte benutzt werden können.

models Hier wird das GitHub-Repository von TFOD [61] geklont.

workspace In diesem Ordner können weitere Ordner für verschiedene Projekte angelegt werden.

Voraussetzung für die API ist Python 3.7. Es kann Anaconda mit dem entsprechenden Python verwendet werden, was aber optional ist. Für die Verwendung von NVIDIA GPU müssen Compute Unified Device Architecture (CUDA) und CUDA Deep Neural Network Library (cuDNN) installiert werden. Es wird ebenfalls empfohlen, eine virtuelle Umgebung für die Python-Pakete einzurichten. Das hat den Vorteil, dass eventuelle

Abhängigkeiten und Konflikte, die ggf. mit anderen Frameworks entstehen, vermieden werden. Bei der Verwendung von Anaconda kann dies mit dem Befehl *conda create -n TensorFlowObjectAPI pip python=3.7* in der Kommandozeile erfolgen. Zum Aktivieren der Umgebung wird der Befehl *activate TensorFlowObjectAPI* benutzt. TensorFlow kann mit dem Paketmanager *pip* von Python installiert werden. Die Version ist davon abhängig, ob die TensorFlow Object API in Version 1 oder 2 installiert wird. Für Version 2 muss das Paket *TensorFlow* mindestens mit der Version 2.2 benutzt werden. Ab TensorFlow-Version 2.2 ist die GPU-Unterstützung automatisch enthalten. Bei TensorFlow-Version 1 muss das Paket *TensorFlow_gpu* verwendet werden, da die GPU Unterstützung nicht standardmäßig in TensorFlow 1 enthalten ist. Als nächster Schritt werden in einem eigenen Ordner die benötigten Dateien von GitHub geklont. Die Versionen unterscheiden sich auch hier (Version 1 [58] und Version 2 [60]). Danach muss Googles Protocol Buffers (Protobuf) [45] heruntergeladen und kompiliert werden.

2.6.1 LabelImg

LabelImg ist ein frei verfügbares Tool, um Bilder zu annotieren. Es unterstützt die Formate PASCAL Visual Object Classes (PASCAL VOC), You only look once (YOLO) und CreateML. Es kann entweder direkt mit pip installiert werden oder als Datei heruntergeladen werden. Nach dem Start kann der Ordner geöffnet werden, in dem sich alle Bilder zum Labeln befinden. Mit dem Button *Create RectBox* kann eine Box um das Objekt gezogen werden, da annotiert werden soll. Nachdem die Box gezogen wurde, erscheint automatisch ein Popup, bei dem die Klasse eingegeben weden kann. LabelImg erstellt automatisch die entsprechende XML-Datei.

2.6.2 Vorbereitung des Modells

Bevor damit begonnen werden kann, das Modell zu trainieren, müssen die TensorFlow-Records erstellt werden und das Modell muss heruntergeladen sowie konfiguriert werden. Zunächst müssen die annotierten Daten wie üblich in Trainings- und Testdaten aufgeteilt werden. Das Verhältnis ist in der Regel 90% zu 10%. Die Dateien können in einem Ordner im Projektordner unter Workspace abgelegt werden. Es gibt jeweils einen Ordner für Test- und Trainingsdaten. Die Bild- und XML-Dateien befinden sich zusammen in einem Ordner. TensorFlow benötigt eine Label-Map. Die Label-Map ist eine Zuordnung von einer positiven ganzen Zahl und den Klassen. Sie wird als pbtxt-Datei gespeichert und besitzt ein JSON-ähnliches Format.

TensorFlow Records Dateien werden in zwei Schritten erzeugt. TensorFlow Record ist eine eigene Datenstruktur für die Verarbeitung von vielen und großen Bilddateien. Sie nutzt Protobuf zum Serialisieren und zum Deserialisieren. Durch die Speicherung im Binary-Format ist sie optimiert für Leseoperationen [vgl. 43, S. 221]. Zuerst ist eine Konvertierung von XML zu CSV und danach von CSV zu Record notwendig. TensorFlow stellt jeweils zur Konvertierung eigene Skripte bereit [12]. Dem Skript zum Konvertieren von XML zu CSV muss als Parameter der Speicherort der Bild- und XML-Dateien sowie der Zielordner für die CSV übergeben werden. Dem anderen Skript muss der Speicherort der CSV- und der Bilddateien sowie der Zielordner für die Record-Datei übergeben werden. Die Umwandlung muss für die Trainings- und Testdaten gesondert getätigt werden.

Bevor mit dem Training begonnen werden kann, muss das vortrainierte Modell heruntergeladen werden und die Training-Pipeline muss konfiguriert werden. Die Modelle können direkt aus dem TenorFlow-Detection-Model-Zoo bezogen werden [57] [59]. Zu jedem Modell werden eine Geschwindigkeit in Millisekunden sowie die Mean Average Precision (mAP) für einen Datensatz angegeben, z. B. COCO. Dies dient dem Vergleich.

Empfohlen ist, dass das Modell in einen Ordner mit dem Namen *pre-trained-models* gespeichert wird. Es muss noch ein Ordner *models* im Projektordner erstellt werden. Dieser beinhaltet unter anderem die Pipeline-Config-Datei und Dateien, die während des Trainings erstellt werden.

Der letzte Schritt vor dem Trainieren ist die Konfigurierung der Datei *pipeline.config*. Ein Beispiel für die config-Datei ist im Anhang C zu finden. In der Datei sind viele Parameter zum Anpassen des Modells hinterlegt. Die folgenden Parameter können ggf. angepasst werden:

num_classes Die Zahl gibt an, wie viele Klassen vorhergesagt werden sollen. Die Zahl sollte mit der Anzahl der Klassen in der Label-Map übereinstimmen.

data_augmentation_options Es kann angegeben werden, ob eine Data- Augmentation, wie Flip oder Crop, gemacht werden soll.

optimizer Bei dieser Option kann der Optimizer verändert werden, falls das Modell keine guten Resultate liefert.

fine_tune_checkpoint Es kann der Ort geändert werden, an dem sich die ckpt-Dateien befinden.

label_map_path Speicherort für die Label-Map. Die Datei muss mit angegeben werden. Der Eintrag ist jeweils für die Trainings- und für die Testdaten zu ändern.

input_path für die Trainingsdaten Speicherort, an dem sich die Record-Datei für die Trainingsdaten befindet.

input_path für die Testdaten Speicherort, an dem sich die Record-Datei für die Testdaten befindet.

num_steps Gibt an, wie viele Schritte trainiert werden soll.

2.6.3 Training

Zum Starten des Trainings befindet sich das Skript *model_main.py* im Ordner *models/research/object_detection*. Das Skript sollte in den Projektordner kopiert werden. Das Skript wird über die Kommandozeile von Windows ausgeführt.

Listing 2.3: Windows-Befehl zum Starten des Trainings

```
python model_main.py --alsologtostderr --model_dir=training/ --
    pipeline_config_path=training/ssd_inception_v2_coco.config
```

Es erfolgt alle 100 Schritte eine Ausgabe in der Kommandozeile . Die Initialisierungsphase des Trainings kann ggf. länger dauern. Der Trainingsverlauf kann mit TensorBoard verfolgt werden. TensorBoard ist ein Tool von TensorFlow, dass den Verlauf der Loss-Funktion und die Genauigkeit visualisiert. Zuerst ist es notwendig eine Kommandozeile zu öffnen und in das Verzeichnis zu wechseln, in dem sich die Checkpoint-Dateien befinden. Danach kann mit dem Befehl *tensorboard –logdir=.* das Board geöffnet werden.

2.6.4 Export

Nachdem das Training beendet wurde, kann das Modell exportiert werden. Das Skript befindet sich im Ordner *models/research/object_detection*. Die Argumente müssen angepasst

werden. Es muss der Pfad zur Pipeline-Config angegeben werden. Die Nummer im Argument *trained_checkpoint_prefix* muss auf die Zahl des letzten Checkpoints gesetzt werden. Ebenfalls muss der Speicherort des Modells angegeben werden.

Listing 2.4: Befehl zum Exportieren des Modells

```
python export_inference_graph.py --input_type image_tensor --
    pipeline_config_path training/ssd_inception_v2_coco.config --
    trained_checkpoint_prefix training/model.ckpt-13302 --
    output_directory trained-inference-graphs/output_inference_graph_v1.
    pb
```

2.7 Data Augmentation mit imgaug

Data Augmentation beschreibt eine Technik, die benutzt wird, um einen vorhandenen Datensatz mit weiteren künstlich erzeugten Bildern zu erweitern. Die Techniken umfassen beispielsweise Folgendes:

Flip Ein Bild wird entlang der horizontalen und der vertikalen Achse gespiegelt.

Crop Es wird ein zufälliger Teil des Bildes ausgeschnitten. Damit wird die Überanpassung verringert und das Modell verallgemeinert besser.

Rotate Das Bild wird rotiert. In der Regel wird zwischen -45 Grad und +45 Grad rotiert.

Color Die Farbe wird verändert, z. B. zu Graustufen.

Random Erasing Zufällig ausgewählte Teile des Bildes werden entfernt.

Data Augmentation wird zum einen benutzt, um den Datensatz mit künstlichen Daten zu vergrößern. Dies betrifft vor allem Bereiche, in denen es nicht ohne Weiteres möglich ist, einen Datensatz zu erstellen. Zum anderen verringert die Augmentation eine Überanpassung, in dem die Trainingsdaten variieren. imgaug ist eine auf Python basierende Bibliothek für Data Augmentation. Der Vorteil ist, dass imgaug einfach zu benutzen ist und eine große Anzahl an Techniken für die Augmentation ermöglicht. Zusätzlich unterstützt es mehrere Prozessoren für die Verarbeitung. Es kann über den Paketmanager von Python mittels *pip install imgaug* installiert werden. Um ein Bild zu augmentieren, muss das Bild zuerst geladen werden. Dies kann z. B. mit OpenCV gemacht werden.

Im Paket *augmenters* befinden sich alle Techniken zum Augmentieren. In der Regel wird eine Sequenz von *augmenters* als Array der Methode *iaa.Sequential()* übergeben. *Fliplr()* spiegelt das Bild entlang der horizontalen Achse. *augmenters* können noch zusätzliche Parameter übergeben werden. Im Falle von *Fliplr()* kann angegeben werden, wie viel Prozent der Bilder gespiegelt werden soll. Für die Augmentation der Bilder können der Methode *iaa.Sequential()* über eine Schleife die Bilder nacheinander übergeben werden. Die neuen Bilder können wahlweise wieder mit *imwrite* von OpenCV gespeichert werden.

3 Stand der Technik und Anforderungsanalyse

In diesem Kapitel erfolgt die Analyse. Zuerst wird der Stand der Technik dargelegt. Dieser wird aus drei verschiedenen Blickwinkeln betrachtet. Es handelt sich um die Segmentierung und die Klassifizierung von Wunden sowie um die Erkennung von Körperteilen. Danach erfolgt eine Analyse der vorhandenen Apps im Google Play Store. In Abschnitt 3.1.4 wird ein Zwischenfazit gezogen. Im nächsten Abschnitt werden die Anforderungen an die App erfasst. Die Anforderungsanalyse beinhaltet das Use-Case-Diagramm, die Anwendungsfälle sowie die funktionalen und nicht-funktionalen Anforderungen. Abschließend wird das Kapitel zusammengefasst.

3.1 Stand der Technik

Die Analyse des Stands der Technik umfasst drei für die Arbeit relevante Felder, die sich aus Abschnitt 1.2 ableiten lassen. Diese sind die Erkennung von Körperteilen, an denen sich die Wunde befindet, die Segmentierung sowie die Lokalisierung der Wunde.

3.1.1 Erkennung der Körperteile und Pose

Die Autoren Jalal et al. stellen in ihrem Artikel [36] ein Verfahren vor, bei dem eine menschliche Silhouette sowie ein Modell eines Körpers benutzt werden, um die Körperteile zu erkennen. Es wird dabei zuerst eine Segmentierung des Körpers mithilfe des Hauttons durchgeführt. Danach wird die menschliche Silhouette analysiert und die Body-Key-Points werden erkannt. Für die Berechnung der einzelnen Körperteile werden bestimmte Verhältnisse, z. B. die Kopfbreite zur Körperhöhe und zur Körperbreite, sowie andere Methoden benutzt.

Die Erkennung einer menschlichen Pose ist ebenfalls ein ausschlaggebender Aspekt und z. B. für eine Mensch-System-Interaktion sowie bei der Erkennung und bei der Vorhersage von menschlichen Handlungen relevant. Posen können mit einem künstlichen neuralen Netzwerk erkannt werden. V. T. Hoang, und K. H. Jo [33] stellen eine CNN Architektur vor, die aus drei verschiedenen Teilen besteht und gute Ergebnisse liefert.

Im Jahr 2020 wurde ein Artikel [40] veröffentlicht, bei dem ein Ansatz vorgestellt wird, mit dem mithilfe einer Kombination aus einer linearen Diskriminanzanalyse und einem künstlichen neuronalen Netzwerk zwölf verschiedene Körperteile sowie menschliche Handlungen erkannt werden können. Es werden dazu des KTH-Datensatz der KTH Royal Institute of Technology [48] und der Weizmann-Human-Action-Datensatz [2] benutzt. Dabei werden vier verschiedene Phasen durchlaufen. Als Erstes wird eine Segmentierung des Körpers mithilfe des von Jalal et al. [36] vorgestellten Verfahrens durchgeführt. Die Körperteile werden mittels eines Algorithmus und anhand der menschlichen Silhouette erkannt. Im Anschluss wird ein Tracking über die zuvor gefundenen Body-Key-Points durchgeführt. Die Feature-Extraction findet unter der Verwendung einer linearen Diskriminanzanalyse statt. Der durch die Analyse entstandene mehrdimensionale Vektor dient als Eingabe für ein künstliches neuronales Netz (KNN). Die Ausgaben des Netzes sind

verschiedene menschliche Handlungen wie Laufen, Gehen oder Klatschen. Das Verfahren wurde mit dem Datensatz des KTH Royal Institute of Technology [48] und mit dem Weizmann-Human-Action-Datensatz [2] getestet. Die Evaluierung zeigt, dass das Verfahren bessere Ergebnisse liefert als die in den anderen Artikeln [39, 49, 68] vorgestellten Methoden.

3.1.2 Segmentierung und Klassifikation

Eine Unterstützung durch eine Segmentierung und durch eine Klassifizierung von Wunden ist für bestimmte Anwendungen wie die Wunddokumentation interessant. Besonders die vollautomatische Segmentierung und die Vermessung der Wunde mithilfe eines Rulers sparen Zeit. CNN können für die Segmentierung verwendet werden. Wang et al.[10] stellen ein neues Framework vor, das ein MobileNetV2 und Connected-component labeling (CCL) kombiniert. Das Ziel ist die automatische Segmentierung von chronischen Fußgeschwüren. Das robuste Modell ist mit einem selbsterstellten Datensatz trainiert. Das MobileNetV2 wird zuerst benutzt, um eine Vorsegmentierung vorzunehmen. Als Ergebnis entsteht ein Graustufenbild. Die Wunde wird als ein zusammenhängender weißer Bereich dargestellt. Durch die erste Segmentierung kann auch ein Rauschen entstehen. Das Rauschen ist eine Segmentierung außerhalb der Wunde. Zusätzlich kann ebenfalls ein Bereich innerhalb der Wunde nicht korrekt segmentiert werden. In diesem Bereich entstehen in der Wunde ein oder mehrere schwarze Löcher. Daher wird CCL verwendet, um das Rauschen und die schwarzen Löcher zu entfernen. Das Framework wurde zum einen mit anderen Modellen wie MobileNetV2 oder Mask-RCNN und zum anderen mit einem angepassten Medetec-Datensatz [63] evaluiert. Die Resultate zeigen gute Ergebnisse. Der DICE Score beziffert sich auf 94,05% und wurde mit dem Medetec Datensatz errechnet.

Die Autoren Anisuzzaman et al. [3] zeigen in dem im Jahr 2020 veröffentlichen Artikel einen Ansatz zur Wundlokalisierung in einer App für Apple Smartphones. Es wird wahlweise ein You only look once Version 3 (YOLOv3) oder YOLOv3-tiny Modell benutzt. Mithilfe von Core ML wird die Integration des Modells in die iOS-App ermöglicht. Die App macht ein Foto der Wunde und erkennt, wo sich die Wunde befindet. Um die Wunde wird eine Box erstellt. Im Rahmen der Evaluation werden Precision, Recall, F1-Score und mAP berechnet. Bis auf Recall von YOLOv3-tiny sind alle Werte über 90 Prozent. Das Referenzmodell ist SSD und es wird der Medetec- Datensatz [63] zum Trainieren benutzt.

3.1.3 Analyse Google Play Store

Relevant für eine Betrachtung des Stands der Technik ist auch die Analyse der vorhandenen Apps. Dazu wird der Play Store von Google betrachtet. Für die Suche nach relevanten Apps wurden die Suchbegriffe „Erste Hilfe“ und „First Aid“ benutzt. Die beiden Ergebnisse der Suchanfragen werden zusammengefasst, wobei Apps, die in beiden Ergebnissen vorkommen, nur einmalig betrachtet werden. Insgesamt werden somit 15 verschiedene Apps untersucht. Hierzu werden die ersten zehn gelisteten Apps im Play Store von Google untersucht. Eine Auflistung und eine Beschreibung der Apps sind in Tabelle 3.1 und 3.2 dargestellt.

Tabelle 3.1: Übersicht Apps im Play Store Teil 1, Stand 11.06.2021

Name	Beschreibung
Erste Hilfe des SRK [20]	Hilfestellungen für verschiedene Erste-Hilfe-Maßnahmen. Der Anwender wählt den Notfall selbst aus. Die Handlungsanweisung erfolgt in Textform. Zusätzlich ist es möglich, ein Video mit der Erklärung aufzurufen.
Instant Aid - Erste Hilfe App [28]	Textgestützte Anleitung der Maßnahme. Dem Anwender werden verschiedene Fragen gestellt, die er beantwortet. Basierend auf den Fragen wird die Handlungsanweisung erstellt.
ASB App Erste Hilfe im Notfall [16]	Textbasierte Handlungsanweisung. Sechs verschiedene Erklärungsvideos. Der Anwender wählt den Notfall und bekommt die Sofortmaßnahme in einem Pop-up-Fenster angezeigt.
Erste Hilfe Weisses Kreuz [19]	Die Handlungsanweisungen werden basierend auf einem Fragen-Antwort-Dialog dem Anwender angezeigt. Die Handlungsanweisung wird durch Text und Bilder beschrieben.
Erste Hilfe - IFRC [22]	Die Handlungsanweisungen werden basierend auf einem Fragen-Antwort-Dialog dem Anwender angezeigt. Die Handlungsanweisung wird durch Text und Bildern beschrieben. Zusätzlich sind Videos auf YouTube verlinkt.
Erste-Hilfe-Techniken [18]	Einfache Beschreibung von Erste-Hilfe- Techniken.
Erste Hilfe Hand [17]	Erste-Hilfe-App für Handverletzungen. Der Anwender wählt den Typ der Verletzung selbst aus. Die Handlungsanweisung wird auf einer Seite angezeigt. Zu jeder Anweisung ist ein entsprechendes Bild vorhanden.
Kindernotfall-App [29]	Die App bietet dem Anwender eine Auswahl an Maßnahmen aus den Kategorien „Erkennen und Helfen“, „Maßnahmen von A-Z“ sowie „Kinderkrankheiten“. Die Maßnahmen in der Kategorie „Erkennen und Helfen“ beinhalten jeweils das Erkennen, z.B. dass keine Atmung vorhanden ist, das Helfen und zusätzliche Informationen. Die Anweisungen werden durch Zeichnungen der Aktionen unterstützt.
Erste Hilfe für Kinder [21]	Der Anwender wählt verschiedene Maßnahmen aus und wird auf YouTube weitergeleitet.
First Aid and Emergency Techniques [25]	Textbasierte Handlungsanweisung. Der Anwender wählt den Notfall selbst aus. In einigen Fällen sind Zeichnungen vorhanden.
First Aid [27]	Textbasierte Handlungsanweisung. Der Anwender wählt den Notfall selbst aus. Die meisten Anweisungen haben drei verschiedene Abschnitte. Zuerst erfolgen eine allgemeine Erklärung, dann die Merkmale zur Erkennung und die eigentliche Anweisung. Die Anweisung ist eine allgemeine Beschreibung, wie im konkreten Notfall zu verfahren ist. Es erfolgt keine detaillierte Handlungsanweisung zum betreffenden Notfall.

Tabelle 3.2: Übersicht Apps im Play Store Teil 2, Stand 11.06.2021

Name	Beschreibung
St John Ambulance First Aid [30]	Der Anwender wählt den Notfall selbst aus. Nach der Auswahl werden in einem Frage-Antwort-Verfahren zusätzliche Merkmale, wie ob es sich um ein Kind oder um einen Erwachsenen handelt, ausgewählt. Der Anwender bekommt eine detaillierte Anleitung, wie der Notfall zu behandeln ist. Die Handlungsanweisung wird teilweise mit einer Zeichnung und mit Skizzen unterstützt.
First Aid Kit: First Aid and Emergency Techniques [24]	Textbasierte Handlungsanweisungen, bei denen der Anwender die Maßnahme auswählt. Die Handlungsanweisung wird durch Bilder und Zeichnungen ergänzt. Zu vielen Maßnahmen sind Videos verfügbar.
First Aid for Emergency & Disaster Preparedness [26]	Textbasierte Handlungsanweisung. Der Anwender wählt den Notfall selbst aus. Die Handlungsanweisung wird mit Zeichnungen unterstützt.
First Aid For Cyclists [23]	Textbasierte Handlungsanweisung. Der Anwender wählt den Notfall selbst aus. Die Handlungsanweisung werden mit Zeichnungen unterstützt.

3.1.4 Zwischenfazit zum Stand der Technik

Zusammenfassend lässt sich sagen, dass bereits Verfahren für die Wundanalyse mittels künstlicher Intelligenz existieren. Ein jüngeres Verfahren ist dabei schon auf einem Smartphone lauffähig, das jedoch nur zwischen „Wunde“ und „Nicht-Wunde“ unterscheiden, also keine Klassifizierung vornimmt. Die meisten Publikationen beschäftigen sich isoliert mit den Themen der Wundsegmentierung oder der-Wundlokalisierung. Die Anwendungsfälle konzentrieren sich dabei auf spezifische Krankheiten, die eine ärztliche Behandlung zwingend erfordern, wie das diabetische Fußgeschwür. Verletzungen, die im häuslichen Bereich passieren und nicht unbedingt medizinisch versorgt oder beobachtet werden müssen, stehen nicht im Fokus.

Darüber hinaus geht hervor, dass die untersuchten Apps im Google App Store den Anwender nicht bei Entscheidungen unterstützen. Die untersuchten Apps lassen sich in zwei Kategorien einteilen. Bei der ersten Kategorie wählt der Anwender gezielt die Situation aus, bei der er unterstützt werden will, z.B. „Herzinfarkt“. Bei der anderen Kategorie muss der Anwender zuerst Fragen zu den Symptomen beantworten. Basierend darauf wird anschließend die Handlungsempfehlung erstellt. Problematisch ist dabei, dass der Anwender die Fragen korrekt beantworten muss, also die Symptome richtig erkennen muss. Gegebenenfalls muss er zuerst eine Erläuterung durchlesen. Beispielsweise werden Verbrennungen ersten und zweiten Grades dadurch unterschieden, ob sich Blasen bilden.

Ein weiterer Aspekt besteht beim generellen Vorgehen sowie in der Architektur. Bestimmte Teilkomponenten im Gesamtsystem sind bereits verfügbar, wie die Segmentierung einer Wunde oder das Erkennen von Körperteilen. Diese wurden allerdings noch nicht zu einem Gesamtsystem zusammengebaut. Die Architektur muss modular aufgebaut sein, um perspektivisch auch andere Arten von Verletzungen oder Erkrankungen zu erkennen, damit auch verschiedene trainierte Modelle benutzt werden können. Es sind noch keine Best Practices zu einer umfassenden Architektur vorhanden und entsprechende Erfahrungen müssen gesammelt werden.

3.2 Anforderungsanalyse

Die Anforderungen lassen sich aus der Problemanalyse und aus dem Stand der Technik ableiten. Die Lücke lässt sich durch die in diesem Kapitel erstellten Anforderungen schließen. Ein wesentlicher Teil davon ist, dass die Anwender aktiv unterstützt werden und dass keine Auswahl mehr getroffen werden muss. Dafür ist ein Empfehlungssystem notwendig, das eine Wundklassifizierung und eine Wundlokalisierung miteinander kombiniert. Die gesamte Lösung soll auf einem mobilen Endgerät lauffähig sein.

Im folgenden Abschnitt werden die funktionalen und nicht-funktionalen Anforderungen spezifiziert. Die funktionalen Anforderungen werden aus der Anwendersicht im User- Case-Diagramm (siehe Abschnitt 3.2.1) und anhand von Anwendungsfällen (siehe Abschnitt 3.2.2) beschrieben. Die übrigen funktionalen Anforderungen befinden sich in Abschnitt 3.2.3. Die nichtfunktionalen Anforderungen sind in Abschnitt 3.2.4 erfasst.

Im Folgenden wird ein mögliches Anwendungsszenario aus der Sicht eines Anwenders skizziert: Unter dem mentalen Druck einer Erste-Hilfe-Situation vergisst der Ersthelfer das Gelernte aus seinem Erste-Hilfe-Kurs. Er weiß nicht, wie er die Wunde der verletzten Person erstversorgen soll. Hier kommt das KI-Assistenzsystem zum Einsatz. Der Ersthelfer startet die Erste-Hilfe-App auf einem mobilen Endgerät, das kann beispielsweise ein Smartphone oder ein Tablet sein. Nun wird die Wunde über ein Foto oder über einen Livestream erfasst. Das KI-System klassifiziert die Wunde und zeigt dem Ersthelfer die Handlungsempfehlungen zur Versorgung an. Dabei werden nicht nur die einzelnen Versorgungsschritte, sondern auch die benötigten Verbandsmaterialien und Instrumente angezeigt. Der Ersthelfer hat ebenfalls die Möglichkeit, das erfasste Foto zu speichern sowie mehrere Fotos logisch zusammenzufassen, um den Wundverlauf zu dokumentieren. Diese Dokumentation kann für die nachträgliche Behandlung durch einen Experten (Arzt, Sanitäter) genutzt werden.

3.2.1 Use Case Diagramm

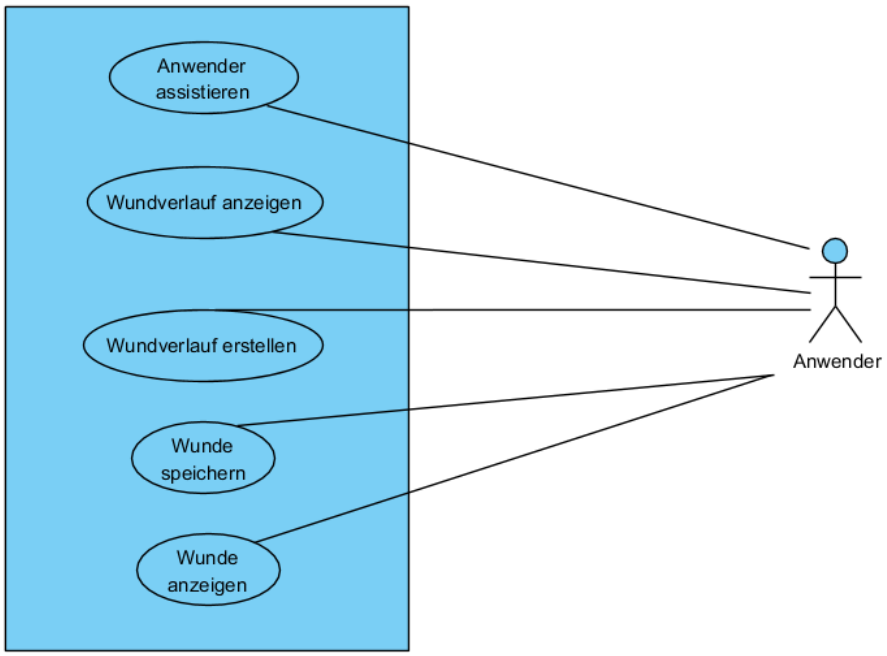


Abbildung 3.1: Use Case Diagramm

3.2.2 Anwendungsfälle

Tabelle 3.3: Anwendungsfall „Anwender assistieren“

Use-Case-ID	01
Name	Anwender assistieren
Ziel	Der Anwender möchte die Hilfe der App in Anspruch nehmen.
Kurzbeschreibung	Der Nutzer benötigt Unterstützung durch die App.
Kategorie	Primär
Vorbedingung	Die App benötigt Zugriff auf die Kamera. Der Anwender muss sicherstellen, dass die App über die entsprechenden Berechtigungen verfügt.
Auslöser	Anwender
Akteure	Anwender
Normalablauf	1. Der Anwender öffnet die App. 2. Der Anwender klickt auf «Hilfe». 3. Foto von der Wunde machen. 4. Die App analysiert das Foto. 5. Die App gibt dem Anwender eine Handlungsanweisung.
Nachbedingung Erfolg	Die App kehrt wieder zum Hauptmenü zurück.

Tabelle 3.4: Anwendungsfall „Wunde anzeigen“

Use-Case-ID	02
Name	Wunde anzeigen
Ziel	Der Anwender möchte eine zuvor gespeicherte Wunde ansehen.
Kurzbeschreibung	Der Nutzer kann sich die alte Wunde anzeigen lassen.
Kategorie	Optional
Vorbedingung	Die Wunde muss vorher abgespeichert werden.
Auslöser	Anwender
Akteure	Anwender
Normalablauf	1. Der Anwender klickt auf «Gespeicherte Daten» 2. Eine Liste wird angezeigt. 3. Auswahl einer Wunde durch den Anwender. 4. Die Wunde wird angezeigt.
Nachbedingung Erfolg	Die App kehrt wieder zum Hauptmenü zurück.

Tabelle 3.5: Anwendungsfall „Wundverlauf erstellen“

Use-Case-ID	03
Name	Wundverlauf erstellen
Ziel	Der Anwender möchte Fotos logisch zu einem Wundverlauf zusammenfassen. Der Anwender ist damit in der Lage, die Heilung einer Wunde zu dokumentieren.
Kurzbeschreibung	Der Anwender kann verschiedene Fotos derselben Wunde zu einem Wundverlauf zusammenfassen.
Kategorie	Sekundär
Vorbedingung	Es muss mindestens ein Foto vorhanden sein.
Auslöser	Anwender
Akteure	Anwender
Normalablauf	1. Der Anwender öffnet das Menü „Wundverlauf“. 2. Dem Anwender werden alle Wundverläufe angezeigt. Der Anwender kann über den Button „Hinzufügen“ einen neuen Wundverlauf erstellen. 3. Der Anwender kann ein neues Bild einer Wunde über den Button „Bild hinzufügen“ dem Wundverlauf hinzufügen.
Nachbedingung Erfolg	Die App kehrt wieder zum Hauptmenü zurück.

Tabelle 3.6: Anwendungsfall „Wundverlauf anzeigen“

Use-Case-ID	04
Name	Wundverlauf anzeigen
Ziel	Der Anwender möchte sich den Wundverlauf anzeigen lassen.
Kurzbeschreibung	Der Anwender kann verschiedene Fotos derselben Wunde zu einem Wundverlauf zusammenfassen.
Kategorie	Sekundär
Vorbedingung	Es muss ein Wundverlauf vorhanden sein.
Auslöser	Anwender
Akteure	Anwender
Normalablauf	1. Der Anwender öffnet das Menü „Wundverlauf“. 2. Dem Anwender werden in einer Übersicht die Verläufe angezeigt. 3. Der Anwender kann einen Wundverlauf öffnen, in dem er auf das Item der Liste klickt. 4. Dem Anwender wird der Wundverlauf mit dem Datum, sowie mit den dazugehörigen Fotos der Wunde angezeigt.
Nachbedingung Erfolg	Die App kehrt wieder zum Hauptmenü zurück.

Tabelle 3.7: Anwendungsfall „Wunde speichern“

Use-Case-ID	05
Name	Wunde speichern
Ziel	Der Anwender möchte ein Foto speichern.
Kurzbeschreibung	Der Anwender kann ein zuvor analysiertes Foto speichern.
Kategorie	Sekundär
Vorbedingung	Der Nutzer befindet sich im Fenster „Anwender assistieren“ gemäß des User-Case 01.
Auslöser	Anwender
Akteure	Anwender
Normalablauf	1. Der Anwender klickt auf den Button zum Speichern. 2. Der Anwender kann verschiedene zusätzliche Informationen, wie Ursache und Bemerkung zum Foto hinzufügen. 3. Der Anwender klickt auf „Speichern“.
Nachbedingung Erfolg	Die App kehrt zum Fenster „Anwender assistieren“ zurück.

Tabelle 3.8: Anwendungsfall „Objekterkennung von Wunden“

Use-Case-ID	06
Name	Objekterkennung
Ziel	Der Anwender möchte eine Objekterkennung von Wunden durchführen.
Kurzbeschreibung	Der Anwender kann eine Objekterkennung von Wunden von einem Livestream der Kamera durchführen.
Kategorie	Sekundär
Vorbedingung	Die App benötigt Zugriff auf die Kamera. Der Anwender muss sicherstellen, dass die App über die entsprechenden Berechtigungen verfügt.
Auslöser	Anwender
Akteure	Anwender
Normalablauf	1. Der Anwender öffnet das Menü „Live-Analyse“. 2. Dem Anwender wird ein Livestream der Kamera angezeigt. Der Stream wird analysiert und die Wunde wird mit einer Box markiert. Es werden der Wundtyp und der Confidence-Wert angezeigt.
Nachbedingung Erfolg	Die App kehrt wieder zum Hauptmenü zurück.

3.2.3 Funktionale Anforderungen

Balzer beschreibt die Anforderungen als „Fähigkeiten eines Systems, die ein Anwender erwartet, um mit Hilfe des Systems ein fachliches Problem zu lösen. Die Anforderungen werden aus den zu unterstützenden Geschäftsprozessen und den Ablaufbeschreibungen zur Nutzung des Systems abgeleitet “ [5, S. 489]. Mit den funktionalen Anforderungen werden die Funktionalität, die Daten und das Verhalten der App erfasst und festgelegt.

Wundlokalisierung Die Erkennung der Körperteile spielt gerade für zukünftige Anwendungen eine Rolle. Eine Erkennung muss zuverlässig funktionieren, wenn das gesamte Körperteil auf dem Foto zu sehen ist. Die App soll erkennen, ob auf dem Foto bzw. auf dem Livestream ein Kopf oder ein Arm zusehen ist.

Klassifizierung der Wunden Die App soll zuverlässig die folgenden Wunden erkennen:

- Kleine Schnittwunden
- Tiefe Schnittwunden
- Verbrennungen ersten Grades
- Verbrennungen zweiten Grades

Kleine Schnittwunden entstehen durch die Einwirkung von scharfen Gegenständen,z. B. von Messern, Scheren oder Glassplittern. Die Wunden bluten leicht und ein deutlicher Wundspalt ist erkennbar. Im Gegensatz zu kleinen Schnittwunden hat eine tiefe Schnittwunde klaffende Wundränder und blutet stark. Eine ärztliche Behandlung ist zwingend erforderlich, wenn die Wunde trotz Druckverband nicht

aufhört, zu bluten. Eine Verbrennung wird durch eine Hitzequelle, wie Feuer und heißes Wasser oder Öl,verursacht und die Haut ist dabei stark gerötet. Bei einer Verbrennung zweiten Grades bilden sich Blasen auf der Haut. Die Klassifizierung soll dabei automatisch erfolgen, ohne dass der Anwender den Wundtyp selbst bestimmen muss.

Handlungsanweisung Dem Anwender wird eine Handlungsanweisung basierend auf der Klassifizierung der Wunde angezeigt. Die Handlungsanweisung dient dazu, den Anwender gezielt bei einer Erstbehandlung einer Wunde zu unterstützen. Die Handlungsanweisung wird direkt auf dem Display des mobilen Endgerätes abgebildet. Optional kann ein unterstützendes Symbol angezeigt werden. Bei bestimmten Handlungsanweisungen kann der Anwender mit dem Symbol interagieren. Beispielsweise kann ein Pflaster bei der Handlungsanweisung für kleine Schnittwunden angezeigt werden.

Wundverlauf Ein Wundverlauf ist für eine Wunddokumentation von Bedeutung. Der Wundverlauf dokumentiert chronologisch die Heilung der Wunde durch entsprechende Bilder. Anhand des Wundverlaufes kann nachvollzogen werden, wie gut die Wunde verheilt oder ob sich die Wundheilung verschlechtert. Indikatoren dafür sind z.B. ein verändertes Wundbild oder hohe Exsudatmengen. Die App soll die Möglichkeit bieten, Fotos logisch zusammenzufassen.

3.2.4 Nicht-funktionale Anforderungen

Nicht funktionale Anforderungen beschreiben „wie“ die App sich verhält, sowie die Eigenschaften, die für die gesamte App gelten und grundlegend für den Architekturentwurf berücksichtigt werden müssen [vgl. 5, S. 489].

Einfache Bedienbarkeit - Für den Anwender muss die App einfach und intuitiv bedienbar sein. Die Anordnung und das Aussehen der GUI-Elemente sollen den Best Practices entsprechen. Die GUI muss übersichtlich gestaltet sein. Die Buttons und die Schriftgröße müssen ausreichend groß sein. Eine barrierefreie Benutzeroberfläche ist nicht zwingend erforderlich.

Performance Die Analyse des Fotos oder des Livestreams muss in einer angemessenen Zeit stattfinden.

Handlungsanweisungen Alle Handlungsanweisungen müssen klar formuliert und für den Anwender verständlich sein. Die einzelne Handlungsanweisung muss so formuliert sein, dass der Anwender dieser auch in einer Stresssituation folgen kann. Medizinisch relevante Begriffe sollten gegebenenfalls erklärt werden. Es kann beispielsweise nicht davon ausgegangen werden, dass ein Anwender weiß, wie ein Druckverband anzulegen ist. Die Handlungsanweisung kann mit Symbolen und optional mit einer Sprachfunktion ergänzt werden.

Mobile Endgeräte und HoloLens Die entwickelten Modelle müssen auf den mobilen Endgeräten lauffähig sein und sollten sich ggf. für andere Plattformen wie die HoloLens leicht erweitern lassen.

3.3 Zusammenfassung

Basierend auf den genannten Anforderungen sowie auf dem beschriebenen Problem lassen sich im Wesentlichen zwei verschiedene Bündel an Funktionen ableiten. Das erste Bündel umfasst alle Funktionen, die für eine App auf mobilen Endgeräten notwendig sind. Diese umfassen beispielsweise das User- Interface, die Modellklassen, der Persistence-Layer oder die Handlungsanweisungen. Das andere Bündel umfasst die Entwicklung zweier Modelle zur Klassifizierung und zur Erkennung von Wunden sowie die Lokalisierung der Wunde, damit festgestellt werden kann, an welchem Körperteil sich die Wunde befindet. Bei der Auswahl der Architektur des Modells ist darauf zu achten, dass die Balance zwischen der Genauigkeit und der Performance der App gegeben ist. Schwergewichtige Modelle wie VGG16 liefern zwar gute Ergebnisse bei der Erkennung von Bildern, können jedoch die gesamte Performance negativ beeinträchtigen. Die beiden Modelle müssen in ein Empfehlungssystem integriert werden, das die Handlungsanweisungen für den Anwender generiert.

4 Konzeption

In diesem Kapitel wird die App konzeptionell entworfen. Zuerst werden das grundsätzliche Design und der Zusammenhang der vier Komponenten präsentiert. Anschließend werden die Seiten der App konzipiert und Mockups der GUI werden vorgestellt. Danach werden zwei Entwurfsmuster präsentiert, die für die Architektur relevant sind und verwendet werden. Im Klassendiagramm werden zum einen die Klassen logisch in Pakete zusammengefasst und zum anderen werden die Klassen in der Unified Modeling Language (UML) modelliert. Danach wird spezifisch auf den Wundlokalisator und auf den Wundklassifikator eingegangen. Zum Schluss erfolgt der Entwurf des Empfehlungssystems.

4.1 Architektur

Abschnitt 3.1.4 beschreibt das Problem, dass der Anwender zuvor immer eine Auswahl treffen muss und nicht durch ein intelligentes System unterstützt wird. Es ist also ein Empfehlungssystem für die Versorgung der Wunden erforderlich. Das System stützt sich dabei auf zwei Komponenten für die Erkennung von Wunden und für die Lokalisierung. Beide Komponenten bestehen aus Methoden des maschinellen Lernens.

Die in Kapitel 3 erfassten Anforderungen können in verschiedene Module zusammengefasst werden. Die zu entwickelten Module sind folgende:

Wundklassifikator Das Modul ist für die Objekterkennung und für die Klassifikation von Wunden zuständig. Diese umfassen die vier verschiedenen Typen von Wunden. Die Klassifizierung bestimmt für jedes Bild eine Klasse von Wunden. Die Objekterkennung ist in der Lage, die Wunde im Bild zu erkennen und mithilfe einer Bounding-Box zu markieren. Für den Wundklassifikator kann ein Deep-Learning-Modell benutzt werden. Es eignet sich ein CNN für das Modell. Für die Objekterkennung kann zusätzlich noch eine Methode wie SSD eingesetzt werden. Die Entwicklung, das Training und die Evaluierung des Modells können in einem eigenen Framework wie PyTorch, TensorFlow, Keras, Darknet oder Caffe stattfinden und werden in die App importiert.

Wundlokalisator Dieses Modul ist für die Identifizierung des Körperteils verantwortlich. Für eine verbesserte Erkennung wird der Arm logisch in zwei Klassen geteilt. Diese Klassen umfassen den Ober- und Unterarm sowie die Hand. Für den Wundlokalisator ist eine Klassifizierung ausreichend. Wie beim Wundklassifikator wird ein Deep-Learning-Modell benutzt.Dieses wird mittels Deep-Learning-Frameworks erstellt.

Empfehlungssystem Das Empfehlungssystem ist das Regelwerk und erstellt die Handlungsempfehlung. Die Handlungsempfehlungen werden in Abhängigkeit vom Wundklassifikator und vom Wundlokalisator gegeben. Bestimmte Verletzungen, die sich am Kopf befinden, müssen ggf. ärztlich versorgt werden, während dies jedoch nicht gilt, wenn sich die gleiche Wunde z. B. am Bein oder am Arm befindet.

App Die App bildet das Grundgerüst und dient als Plattform für die anderen Module. Sie beinhaltet das User Interface (UI) und eine Datenbank für den Wundverlauf sowie für die Wundbilder. Die App ist für den Zugriff auf die Kamera und auf den Livestream verantwortlich und regelt den Zugriff. Die App importiert die Modelle des Wundklassifikators und des Wundlokalisators. Sie führt ebenfalls ein Pre-processing des Bildes durch, damit das Bild entsprechend von den Modellen verarbeitet werden kann.

Abbildung 4.1 veranschaulicht die verschiedenen Module und deren Zusammenarbeit.

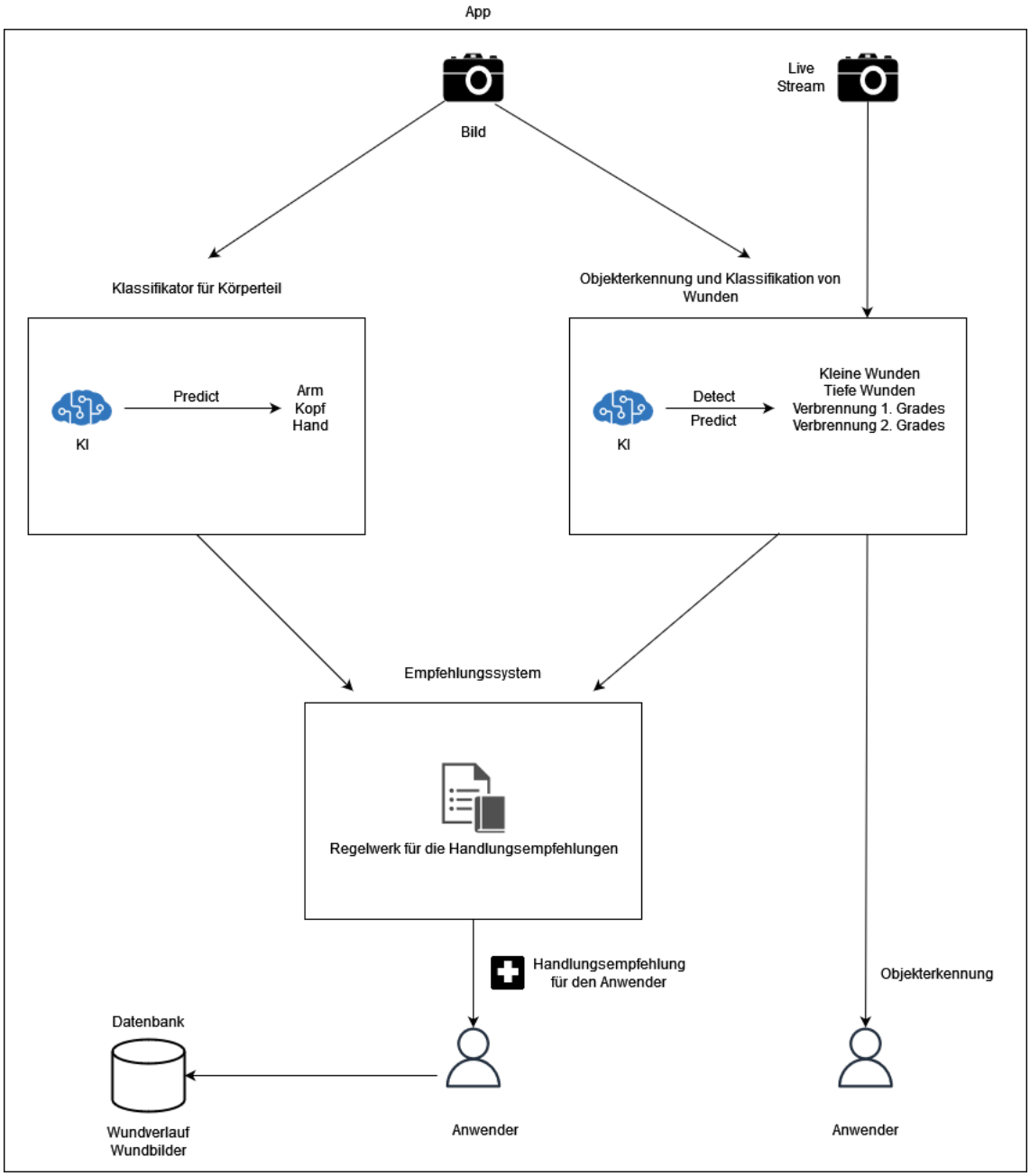


Abbildung 4.1: Übersicht App

4.2 Entwurf der App

4.2.1 Entwurf Seiten

Die gesamte App ist in acht Seiten aufgeteilt, wobei eine Seite eine bestimmte Aufgabe erfüllt. In Anhang D ist der Entwurf für alle Seiten zu finden. Nach dem Starten der App wird ein Hauptmenü angezeigt. Die Seiten sind so aufgebaut, dass möglichst wenig Buttons vorhanden sind, und die Erste-Hilfe-Funktion ist direkt vom Hauptmenü aus erreichbar. Dies stellt sicher, dass auch Anwender unter Stress die App möglichst gut handhaben können. Die Steuerelemente sind ausreichend groß, damit auch Anwender mit motorischen Einschränkungen die Steuerelemente leicht bedienen können.

In Abbildung 4.2 ist die Seitenstruktur dargestellt. Die gesamte App besteht aus neun Seiten. Von der Hauptseite sind die Seiten *Camera*, *History* und *Galerie* erreichbar. Auf der Seite *Camera* wird der Kamera-stream dargestellt. Der Anwender hat hier die Möglichkeit, die Wunde zu fotografieren oder die Objekterkennung zu aktivieren. Der Stream muss durch den Wundklassifikator analysiert werden und die Bounding-Box muss auf der GUI gezeichnet werden.

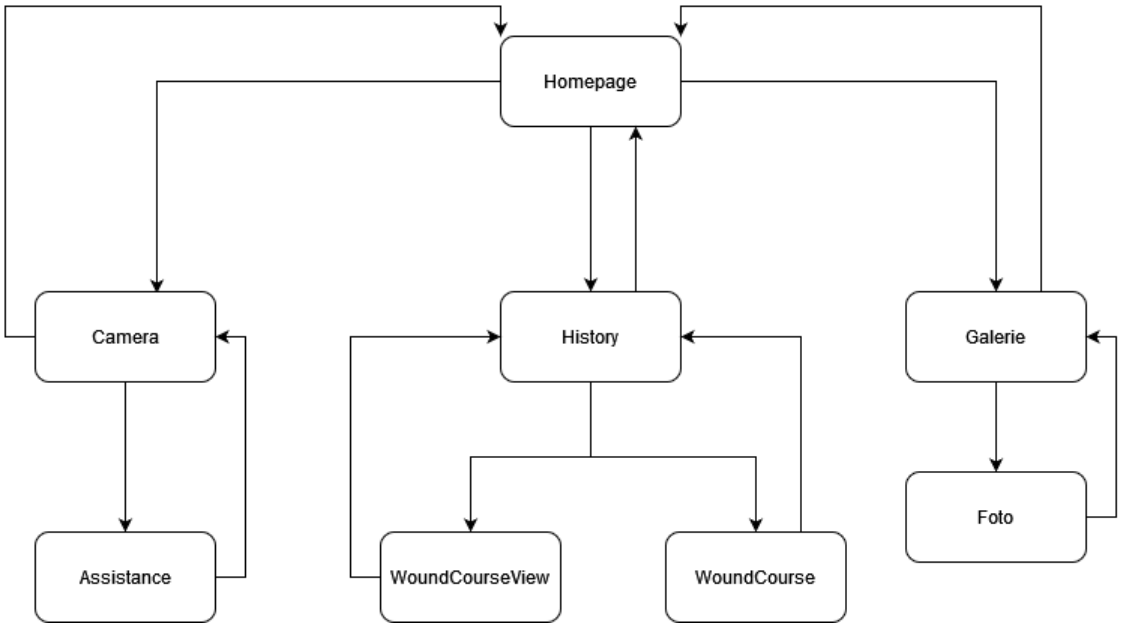


Abbildung 4.2: Navigation

Sobald der Anwender auf den Button zum Fotografieren klickt, wird die Seite *Assistance* geladen und das Bild durch den Klassifikator für Körperteile und durch den Wundlokalisator analysiert. Das Empfehlungssystem wertet die Ergebnisse aus und erstellt die Handlungsanweisung. Dem Anwender wird die Handlungsanweisung angezeigt.

Der Wundverlauf kann über die Seite *History* erstellt und bearbeitet werden. Der Anwender bekommt alle verfügbaren Wundverläufe angezeigt. Die Seite *WoundCourseView* zeigt den Wundverlauf detailliert an. Es können ebenfalls die Bilder von Wunden zu einem bestehenden Wundverlauf hinzugefügt werden. Auf der Seite *WoundCourse* kann auch ein neuer Wundverlauf erstellt werden. Die *Galerie* zeigt alle Fotos an und mit der Seite *Foto* kann sich der Anwender nochmals ein Foto betrachten. Dies wird benutzt, falls eine Wunde dem medizinischen Personal gezeigt werden soll.

4.2.2 Entwurfsmuster Schichtenmuster

Das Schichtenmuster fasst Komponenten gemäß ihrer Funktion logisch zusammen. Die Schichten sind vertikal angeordnet. Die App wurde in Präsentations-, Anwendungs- und Datenhaltungsschicht unterteilt.

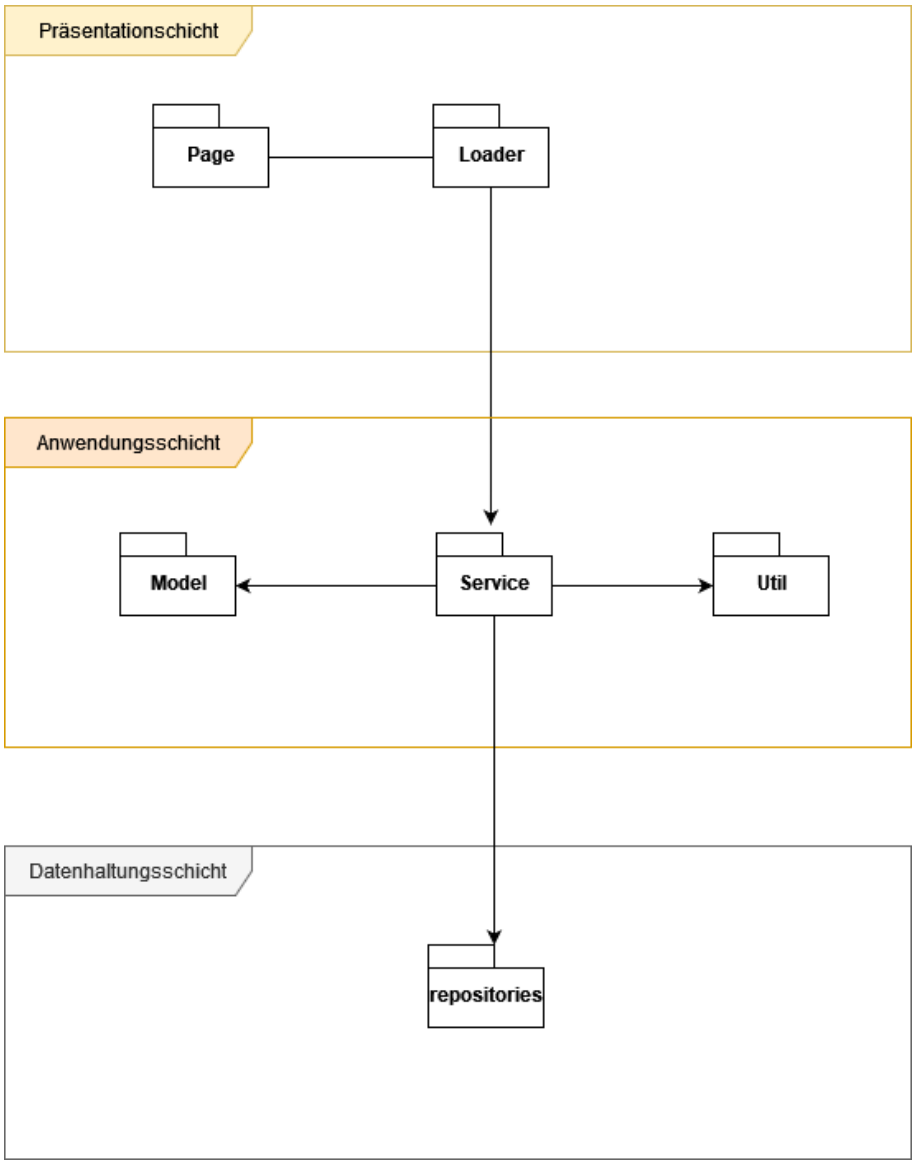


Abbildung 4.3: Schichtenmodell der App mit den wesentlichsten Paketen

Die Funktionen der Schichten werden nur von der darüberliegenden Schicht aufgerufen. Ebenso darf eine Schicht nicht übersprungen werden. Die Präsentationsschicht darf keine Funktionen der Datenhaltungsschicht aufrufen. Ein Vorteil des Schichtenmusters ist, dass die einzelnen Schichten besser ausgetauscht werden können. Das ist dem Fakt geschuldet, dass die Komponenten innerhalb einer Schicht eng verbunden sind, während die Kopplung unter den Schichten lose ist. Dies reduziert die Komplexität der Abhängigkeiten. Der Wartungsaufwand wird ebenfalls reduziert, weil Wartungstätigkeiten logisch getrennt nach Schichten durchgeführt werden können. Ein Nachteil des Schichtenmusters ist, dass immer alle Schichten durchlaufen werden müssen. In dem Fall, dass die Präsentationsschicht eine Information aus der Datenhaltungsschicht, z. B. ein Datenbankeintrag, benötigt, muss ein Zugriff über die Anwendungsschicht erfolgen. Die Daten aus

der Datenhaltungsschicht müssen eventuell mehrfach konvertiert werden z. B. von einer Entitätsklasse zu einer Modellklasse und dann in die Seite geladen werden. Diese Konvertierung findet auch in die andere Richtung statt, also von der Seite zu einer Modellklasse und dann zu einer Entitätsklasse. Es können keine Zwischenschritte übersprungen werden. Das mehrfache Konvertieren macht das Schichtenmodell ineffektiv.

4.2.3 Entwurfsmuster Model View Controller (MVC)

Das MVC-Muster ist im Bereich der grafischen Benutzeroberfläche zentral. Das Entwurfsmuster wird bei Webanwendungen und in der Entwicklung von Smartphone Apps benutzt. Es beschreibt die Aufteilung der Aufgaben in die Komponenten View, Modell und Controller. Die View ist die Ansicht. Sie visualisiert die Daten und beinhaltet grafische oder akustische Elemente einer GUI, die mit dem Anwender interagieren. Das Modell beinhaltet die Datenstruktur und die Daten. Die Daten sind nicht mit einer bestimmten Formatierung versehen, daher kann das Modell von verschiedenen Views benutzt werden. Der Controller enthält die Business-Logik und verbindet das Modell mit der View. Er nimmt die Eingabe des Anwenders entgegen und verarbeitet sie. Gleichzeitig aktualisiert der Controller die View. Der Controller ruft Methoden des Modells auf, um z. B. die Operationen Create, Read, Update and Delete (CRUD) durchzuführen.

Die Aufteilung in die drei Komponenten bringt Vorteile mit sich. Eine starke Kohäsion und eine lose Kopplung ermöglichen es, dass die Komponenten unabhängig voneinander sind und einfacher ausgetauscht und gewartet werden können. Zusätzlich können die Komponenten wiederverwendet werden.

4.2.4 Klassendiagramm

Die verschiedenen Klassen werden in Pakete unterteilt. Das Paket *Pages* enthält die Views mit allen Steuerelementen und das Paket *Controller* enthält die Controller-Klassen gemäß dem Entwurfsmuster MVC (siehe Abschnitt 4.2.3). Die Serviceklassen befinden sich im Paket *Serviceklassen*, während das Paket *Model* die Modellklassen für das Foto und für den Wundverlauf sowie für das Interface und für dessen Implementierung für die Handlungsanweisungen enthält. Im Ordner *Util* werden alle Klassen zusammengefasst, die bei bestimmten Aufgaben unterstützen. Die Klassen, die sich in diesem Ordner befindet, sind in der Regel alle statische Klassen.

Abbildung 4.4, 4.5 und 4.7 zeigen das Klassendiagramm, das zur besseren Übersicht in drei Teile getrennt wurde. Das vollständige Klassendiagramm befindet sich in Anhang E. Die Struktur benutzt die in Abschnitt 4.2.2 und 4.2.3 vorgestellten Entwurfsmuster.

Die Klasse *DAO* ist für den Zugriff auf die Repositories zuständig. DAO steht für Data Access Object (DAO). Die Klasse *DAO* trennt die direkte Abhängigkeit von bestimmten Datenbanken. Sie kann als Zwischenschicht betrachtet werden. Der Vorteil ist, dass das Repository einfacher ausgetauscht werden kann. Die Klasse benutzt dazu das Interface *Repository* und die CRUD Operationen werden generisch aufgerufen. Die konkrete Implementierung wird bei der Instanziierung festgelegt. Die Klasse *ModelObjectBuilder* hilft bei der Erstellung der Modellklassen *Foto* und *History*. Die Serviceklassen rufen die entsprechenden Methoden des *ModelObjectBuilder* auf und bekommen ein fertiges Objekt geliefert. Die Modellklasse *Foto* repräsentiert das Foto. Sie besitzt die Attribute *id*, *date* für das Erstellungsdatum und *IdOfPicture*, damit das Foto einem Wundverlauf zugeordnet werden kann. Die Klasse *History* stellt den Wundverlauf dar. Neben der ID speichert sie auch eine Liste mit Fotos sowie das Datum der Erstellung. Der Anwender kann angeben, was zur Verletzung geführt hat. Das kann im Attribut *incident* gespeichert werden. Das

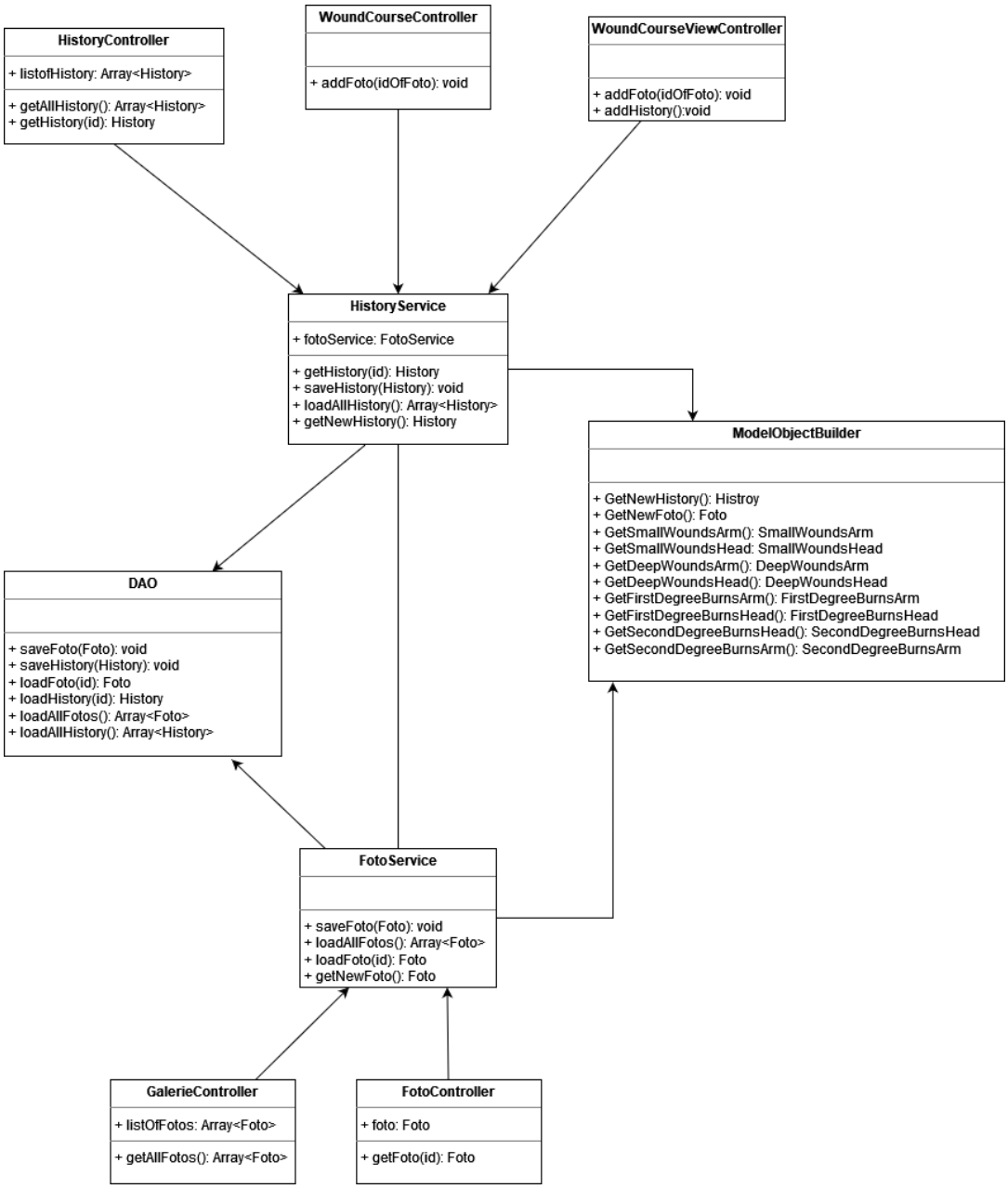


Abbildung 4.4: Klassendiagramm Teil 1, unvollständig

Attribut *note* speichert zusätzlich Angaben, die der Anwender machen kann. Eine Serviceklasse ist nur für eine Modellklasse zuständig und stellt Operationen bereit, um mit den Modellklassen zu arbeiten. Die Serviceklassen folgen einem Namensschema. Zuerst wird die Modellklasse genannt, gefolgt von *Service*. Für jede Seite in der App gibt es einen Controller, der die Logik für die Views enthält.

In Abbildung 4.5 sind die Klassen dargestellt, die das DNN nutzen. Die Klasse *AssistanceController* übergibt der Klasse *RecommendationSystemService* das Bild der Wunde. Die Klasse *RecommendationSystemService* repräsentiert das Empfehlungssystem, *WoundlocaliserService* ist die KI für die Klassifikation der Körperteile und *WoundDetectorService* stellt die Objekterkennung sowie die Klassifikation von Wunden aus Abbildung 4.1 dar. Für die meisten Deep-Learning-Modelle kann die Bilddatei nicht ohne Vorverarbeitung übergeben werden. Beispielsweise muss das Bild einem bestimmten Format, z.B. einer

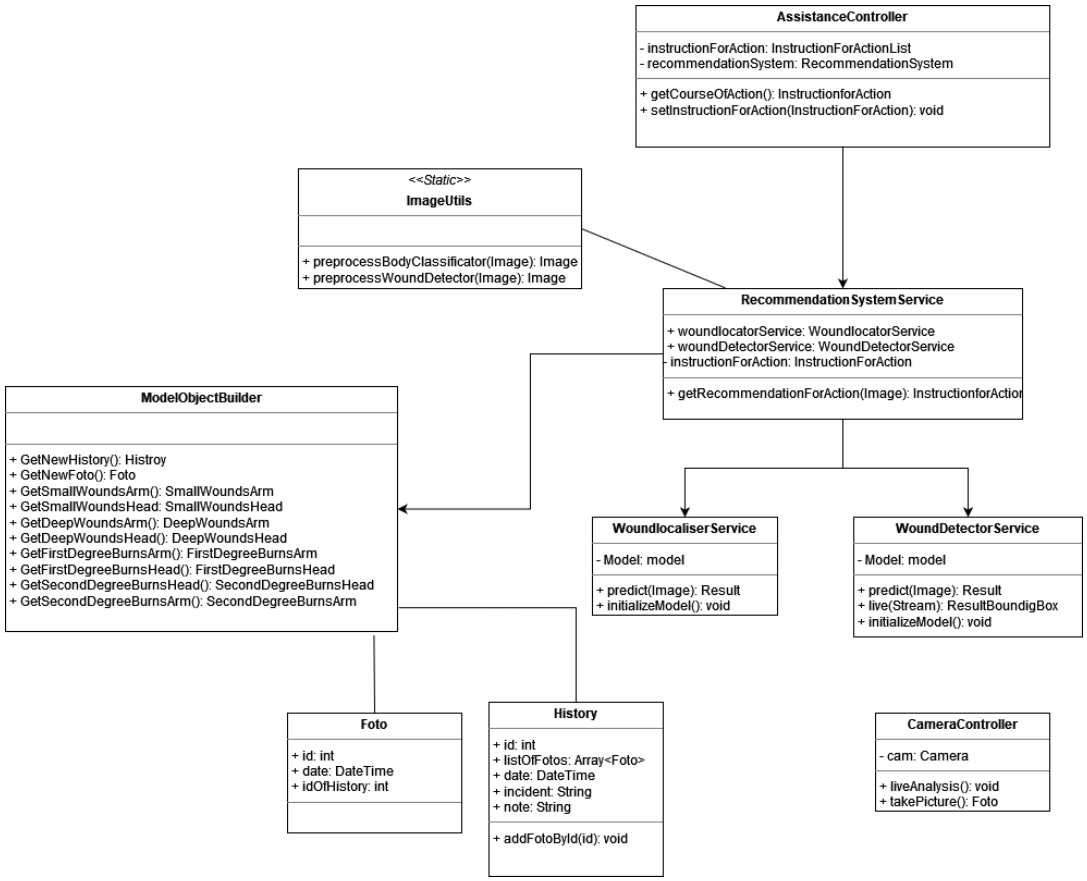


Abbildung 4.5: Klassendiagramm Teil 2, unvollständig

Höhe und Breite von 320 Pixeln, entsprechen. Die statische Klasse *ImageUtils* führt dieses Preprocessing durch. Anschließend wird die Methode *predict* der Klassen *WoundlocaliserService* und *WoundDetectorService* aufgerufen. Die beiden Methoden liefern die Ergebnisse der Vorhersage und der *RecommendationSystemService* wertet die Ergebnisse aus. Basierend auf der Auswertung wird die entsprechende Methode der Klasse *ModelObjectBuilder* aufgerufen und dem *AssistanceController* zurückgeliefert.

Der *WoundDetectorService* stellt ebenfalls eine Methode für die Live-Analyse bereit. Diese wird vom *CameraController* aufgerufen. Der Controller greift auf den Kamerastream zu. Danach führt der *WoundDetectorService* die Objekterkennung der Wunde aus und liefert die Koordinaten, damit die Bounding-Box um die Wunde erstellt werden kann.

4.2.5 Wundverlauf

Mit dem Wundverlauf ist der Anwender in der Lage, Fotos logisch zusammenzufassen. Die Beziehung zwischen dem Wundverlauf und den Fotos ist eine 1:N-Beziehung. Der Wundverlauf speichert automatisch das Erstellungsdatum beim ersten Speichern. Zusätzlich kann der Anwender angeben, wie die Wunde entstanden ist ,und er kann noch ergänzende Notizen abpassen. Abbildung 4.6 zeigt die Seite des Wundverlaufs. Die oberen Buttons dienen zum Zurückkehren auf die letzte Seite, zum Hinzufügen eines Bildes und zum Speichern des Wundverlaufs. Die Textfelder sind für die Eingaben der Anwender gedacht. Im unteren Bereich werden die bereits hinzugefügten Bilder angezeigt. Die Bilder sind nach Datum sortiert.

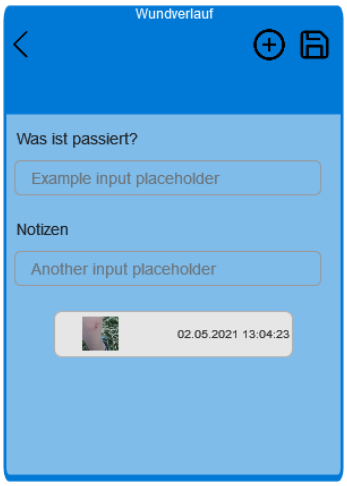


Abbildung 4.6: Eingabemaske für den Wundverlauf

4.3 Datensätze

Damit die neuronalen Netze eine Klassifizierung bzw. eine Erkennung durchführen können, müssen sie zuerst mit den entsprechenden Daten trainiert werden. Es sind zwei verschiedene Datensätze notwendig. Im Internet existieren frei zugängliche Datensätze die ggf. benutzt werden können. Die folgenden Datensätze werden mit Blick auf ihre Benutzbarkeit untersucht:

Medetec Das Medical Device Technical Consultancy Service (MEDETEC) [63] stellt einen kostenlosen Datensatz [63] von Wundbildern zur Verfügung. Der Datensatz enthält Bilder von offenen Wunden verschiedener Arten, wie venöse Beingeschwüre, arterielle Beingeschwüre, Dekubitus oder maligne Wunden. Der Datensatz enthält 654 nicht gelabelte Bilder.

Weizmann Human Action Der Weizmann-Human-Action-Datensatz [2] ist ein öffentlicher Datensatz, der 90 verschiedene Videosequenzen enthält. Die Videosequenzen zeigen Menschen, die verschiedene Handlungen, wie Gehen, Rennen oder Winken, ausführen.

KTH-Dataset Der KTH-Datensatz beinhaltet Videos zu sechs Aktivitäten wie Boxen oder Händeklatschen. Insgesamt befinden sich 600 Videos in diesem Dataset.

COCO dataset Der COCO Datensatz ist ein größerer Datensatz mit 80 verschiedenen Kategorien und insgesamt 330.000 Bildern. Der Datensatz kann benutzt werden, um Modelle für die Objekterkennung und für die Segmentierung zu trainieren. Es ist ebenfalls eine textuelle Beschreibung zu jedem Bild vorhanden, um einen Kontext zwischen den Objekten herzustellen.

WoundsDB WoundsDB [56] ist eine Zusammenstellung von Bildern mit chronischen Wunden. Die Wunden umfassen Unterschenkelgeschwüre, venöse Unzulänglichkeiten, Lymphödem, Diabetischer Fuß, Amputation und Ischämie. Die Bilder sind mit den Modalitäten Farbfotografie, Thermovision, Stereovision und Tiefenwahrnehmung aufgenommen worden. Die Wundregionen wurden durch Chirurgen abgegrenzt. Die Bilder liegen in bearbeiteter und unbearbeiteter Form vor. Insgesamt befinden sich 188 Fälle von 79 Patienten im Datensatz.

Alle diese Datensätze beinhalten nicht die benötigten Bilder. Einschlägige Portale für Data-Science und Deep Learning wie Kaagle.com liefern ebenfalls keine verwertbaren Datensätze. Folglich ist es notwendig, zwei eigene Datensätze zu erstellen. Im Rahmen dessen wurden das Städtische Klinikum Karlsruhe sowie das Knappschaftskrankenhaus Püttlingen angefragt, ob Bildmaterial für den Datensatz für den Wundklassifikator zur Verfügung gestellt werden kann. In der Antwort des Klinikums Karlsruhe wurde mitgeteilt, dass Schnittwunden zwar dokumentiert, aber nicht fotografiert werden. Größere Verbrennungen werden grundsätzlich in eine andere Klinik überstellt. Während der Datensatz für die Wunden nicht einfach selbst zu erstellen ist, kann der Datensatz für den Wundlokalisator in Eigenregie erstellt werden.

4.4 Wundlokalisierung

Die Klassifizierung von Bildern ist eine typische Aufgabe in Computer Vision. Best Practice ist die Verwendung von CNN zur Klassifizierung. Basierend auf CNN stehen verschiedene Architekturen wie:

- Fast R-CNN
- YOLO
- MobileNet
- ResNet

für die Klassifizierung zur Verfügung. Bei der Auswahl der Architektur muss die Balance zwischen der Genauigkeit und der Geschwindigkeit gegeben sein, da die Anwendung auf einer mobilen Plattform ausgeführt wird. MobileNet erfüllt diese Anforderungen, denn die MobileNet-Architektur ist speziell für die Verwendung auf mobilen Endgeräten optimiert. In Tabelle 4.1 ist die Zuordnung der Klassen bzw. der Labels festgelegt. Diese werden beim Trainieren des Modells und beim Empfehlungssystem benutzt. Das Modell gibt für jede Klasse den Confidence-Wert bzw. die Wahrscheinlichkeit an , mit der die Klasse vorhergesagt wird.

Tabelle 4.1: Zuweisung der Klassen für die Körperteile

Klasse	Bezeichnung	Beschreibung
0	arm	Arm
1	head	Kopf
2	hand	Hand

4.5 Wundklassifikator

Der Wundklassifikator verwendet wie der Wundlokalisator eine MobileNet-Architektur. Für die Objekterkennung kann MobileNet nicht allein verwendet werden. Dazu ist es notwendig, die Architektur zu erweitern.Es wird ein MobileNet als Basis benutzt, um die Klassifizierung bzw. Feature Extraction, durchzuführen und anschließend wird SSD

verwendet, um die Erkennung der Wunden auszuführen. Zu beachten ist, dass es sich um eine einzige Architektur und nicht zwei getrennte Modelle handelt.

Tabelle 4.2: Zuweisung der Klassen für die Wunden

Klasse	Bezeichnung	Beschreibung
0	small_cuts	Kleine Schnittverletzungen
1	deep_cuts	Tiefe Schnittverletzungen
2	1_burns	Verbrennungen 1. Grades
3	2_burns	Verbrennungen 2. Grades

In Tabelle 4.2 ist die Zuordnung der Klassen bzw. Labels dargestellt. Diese wird für das Training des Modells und im Empfehlungssystem benutzt. Das Modell liefert als Ergebnis eine Liste mit einer bestimmten Anzahl an Bounding-Boxen. Die maximale Anzahl der Boxen kann durch einen Parameter des Modells begrenzt werden. Zu jeder Bounding-Box, also zu jedem erkannten Objekt, stehen in der Liste die Koordinaten der Box, die Klasse sowie der Confidence-Wert. Für die Klassifizierung sind keine Boxen notwendig und es ist ausreichend, nachdem höchsten Confidence-Wert zu filtern und die zugehörige Klasse auszugeben.

4.6 Empfehlungssystem

Das Empfehlungssystem wertet die Vorhersagen des Wundklassifikators und des Wundlokalisators aus. Es gibt insgesamt acht verschiedene Handlungsanweisungen. Die Handlungsanweisungen befinden sich in Anhang F. Die Handlungsanweisungen sind an das Buch Verbandstoffe für die Kitteltasche [8] und den Artikel Erste Hilfe bei Verbrennungen [52] angelehnt. Die meisten Handlungsanweisungen unterscheiden sich hinsichtlich der Art der Verletzung, mit der Ausnahme Verbrennungen 2. Grades. Hier unterscheiden sich die Handlungsanweisungen. Bei Verletzungen am Kopf muss zusätzlich ein Arzt aufgesucht werden.

Die Handlungsanweisungen finden sich auch im Klassendiagramm der Abbildung 4.7 wieder. Jede Klasse implementiert das Interface *InstructionforAction*. In dem Interface befinden sich Methoden, um die nächste und vorherige Handlungsanweisung zu bekommen sowie die allererste. Die einzelnen Schritte werden in der Klasse *Instruction* gespeichert und enthalten die Anweisung als String sowie ein Symbol, welches bei Bedarf angezeigt werden kann. Die Methode *initializeInstructions* schreibt beim Instanziieren des Objektes die Anweisungen in das Array *arrayOfInstructions*. Die Handlungsanweisungen werden in der Klasse *ModelObjectBuilder* erstellt. Wie bereits beschrieben ist das Empfehlungssystem das Regelwerk für die Handlungsempfehlungen. Nach dem Auswerten der beiden KI-Modelle trifft das System basierend auf dem in der Abbildung 4.8 gezeigtem Entscheidungsbaum die Entscheidung.

Wie bereits beschrieben ist das Empfehlungssystem das Regelwerk für die Handlungsempfehlungen. Nach dem Auswerten der beiden KI-Modelle trifft das System basierend auf dem in der Abbildung 4.8 gezeigtem Entscheidungsbaum die Entscheidung.

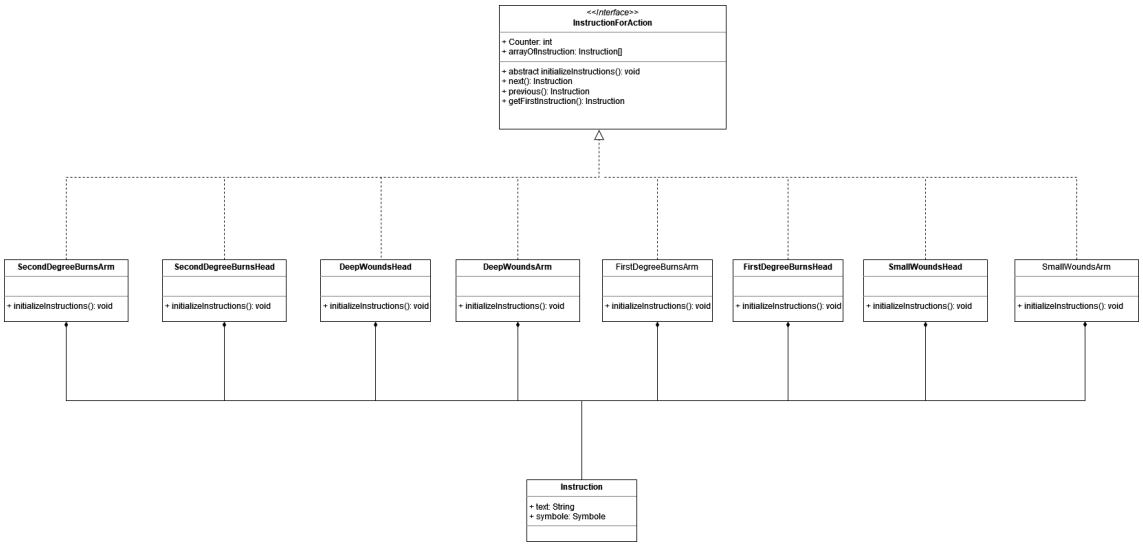


Abbildung 4.7: Klassendiagramm Teil 3, unvollständig

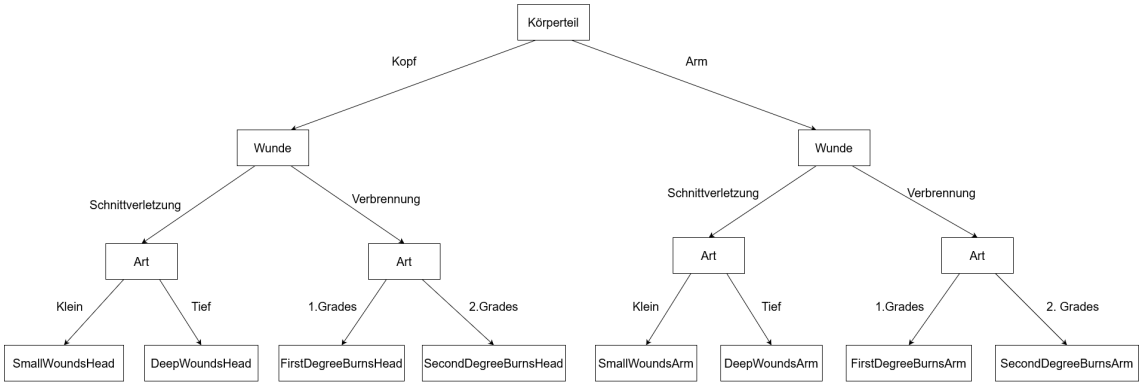


Abbildung 4.8: Entscheidungsbaum für die Handlungsempfehlungen

4.7 Zusammenfassung

In diesem Kapitel wurde die Architektur entworfen. Zwei Entwurfsmuster werden verwendet, mithilfe dessen die Softwarearchitektur gebildet wird. Der Kern des Systems ist das Empfehlungssystem, da mithilfe der Deep-Learning-Modelle und des Entscheidungsbaums die Handlungsempfehlung gibt. Eingebettet sind die drei Komponenten in die Komponente App. Diese stellt das Gerüst der Erste-Hilfe-App dar. Sie bietet die Möglichkeit, die extern erstellten Modelle betreiben zu können. Die App stellt alle Ressourcen, wie die Kamera und ein Speichersystem für die Wundbilder und den Wundverlauf, bereit.

5 Implementierung

In diesem Kapitel wird die technologische Umsetzung beschrieben. Zuerst wird auf die Implementierung der Unity-App eingegangen. Es werden dabei die einzelnen Seiten und die notwendigen Skripte bzw. Klassen erklärt und deren Zusammenhang wird aufgezeigt. Danach werden der Wundlokalisator und der Wundklassifikator detailliert beschrieben. Es wird dabei auf den Datensatz und auf das Training der Modelle eingegangen und die Modelle werden bewertet. Der Import und die Verwendung werden ebenfalls erläutert. Anschließend wird auf das Empfehlungssystem eingegangen und es wird dargelegt, wie der Wundlokalisator, der Wundklassifikator und das Empfehlungssystem interagieren, um die Handlungsempfehlung zu generieren. Abschließend wird die Bereitstellung auf dem Endgerät gezeigt und das Kapitel wird zusammengefasst.

5.1 App

Das Grundgerüst bildet die App. Diese verbindet die beiden Modelle mit dem Empfehlungssystem. Das Framework für die App ist Unity. Mehrere Vorteile sprechen für die Verwendung von Unity. Es unterstützt die Plattformen Android und iOS. Mit der Unterstützung von Android können neben Smartphones auch android-basierte Smartglasses adressiert werden. Darüber hinaus unterstützt Unity auch die Universal Windows Platform (UWP) und somit auch die HoloLens und HoloLens 2. Unity kann als All-in-one-Tool verstanden werden. Ein weiterer Vorteil ist, dass Unity durch den Asset-Store ein breites Spektrum an Erweiterungen anbietet. So lässt es sich leicht um AR-Anwendungen wie Vuforia erweitern. Bevor mit der Entwicklung begonnen werden kann, muss die Plattform ausgewählt werden, für die entwickelt werden soll. Als Plattform wird Android ausgewählt. Android ist mit einem Marktanteil von 72,4% [66] das meist genutzte Betriebssystem für mobile Endgeräte.

Nach der Erstellung des Projekts wird die Ordnerstruktur im Ordner *Asset* erstellt. Die Struktur ist wie folgt aufgebaut:

Resources/Icons In diesem Ordner befinden sich alle Icons, die in der GUI verwendet werden.

Images Bilder, die in der GUI verwendet werden, sind hier abgelegt. Diese umfassen z. B. den Hintergrund der App.

Scenes Dieser Ordner beinhaltet alle Szenen der App.

Scripts Dieser Ordner beinhaltet die Quelldateien der App.

StreamingAssets In diesem Ordner werden die Modelle abgelegt. Alle Dateien, die sich in diesem Ordner befinden, werden durch Unity nicht in das Android Application Package (APK) inkludiert. Unity kopiert diese Dateien unverändert auf das Speichersystem der Zielplattform. Unity kann während der Laufzeit auf die Daten zugreifen.

Im Ordner *Scripts* befinden sich weitere Ordner, die auch die Entwurfsmuster widerspiegeln.

Action Die Klassen für die Handlungsempfehlungen werden hier abgelegt.

DAO In diesem Ordner befindet sich die Klasse, die für Speicheroperationen zuständig ist.

Model Alle benutzten Modellklassen befinden sich in diesem Ordner.

Page Hier werden alle Skripte abgelegt, die in den Szenen benutzt werden.

Service Der Ordner beinhaltet die Klassen, die die Geschäftslogik enthalten. Dazuzählen der Wundlokalisator und der Klassifikator sowie das Empfehlungssystem.

Utils Der Ordner beinhaltet Hilfsklassen aller Art.

Die gesamte App setzt sich aus verschiedenen Unity-Szenen zusammen, die im Grunde alle gleich aufgebaut sind. Jede Szene hat das GameObject *PageNavigation* und ein Canvas. Das Canvas hat eine Auflösung von 720 x 1280 Pixel und ist auf das Testgerät abgestimmt. Jede Szene, außer der Szene Homepage, hat ein Icon, mit dem der Anwender auf die letzte Szene zurückgehen kann. Dieses Icon befindet sich auf der linken oberen Seite.

Für die Verwendung der beiden Modelle wird OpenCV for Unity auf dem Asset-Store benutzt. OpenCV for Unity lässt sich bestens in Unity integrieren und unterstützt die Verwendung von Modellen, die mit TensorFlow erstellt wurden. Zu beachten ist, dass OpenCV for Unity nicht das TensorFlow Format SavedModel unterstützt [11]. Alle Modelle müssen als Frozen Graph exportiert worden sein.

5.1.1 Seite Homepage

Die Seite *Homepage* ist die erste Seite, die geladen wird, wenn die App gestartet wird. Die Szene besitzt drei Buttons, die UI-Elemente von Unity sind. Zu jedem Button werden zwei UI-Elemente vom Typ Raw Image und Text hinzugefügt. Die Elemente sind auf dem Button mittels einer *Horizontal Layout Group* nebeneinander angeordnet. Jeder Button besitzt einen Click-Listener, um die Szene zu laden.

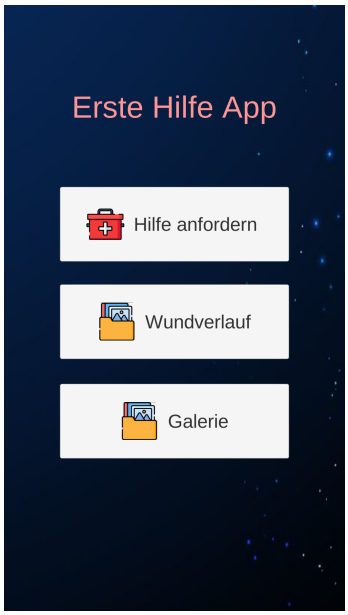


Abbildung 5.1: Seite „Homepage“

Die Klasse *PageNavigationService* ist für die Navigation zuständig. Es gibt für jeden Pfad, also z. B. von der Szene *Homepage* zur *Camera*, eine Methode. Diese Methoden werden von den entsprechenden Buttons über den Click-Listener aufgerufen. Die Klasse erbt von der Klasse *MonoBehaviour*. Der in Listing 5.1 dargestellte Quellcode zeigt exemplarisch eine Methode zum Aufrufen einer neuen Szene. Der SceneManager ist für die Verwaltung der Szenen während der Laufzeit verantwortlich. Mit der statischen Methode *LoadScene* können neue Szenen in der App geladen werden.

Listing 5.1: HomepageToCamera Methode

```
public void HomepageToCamera()
{
    SceneManager.LoadScene("Camera");
}
```

Zu beachten ist, dass die Szene zu den Build-Settings hinzugefügt werden muss (siehe Abschnitt 5.5). Der Methode wird der Name der Szene als String übergeben.

Jede Szene hat ein GameObject mit dem Namen PageNavigation. In diesem GameObject ist das Skript als Komponente hinzugefügt und mit dem entsprechenden Click-Listener des Buttons verdrahtet. Um die Auswahl der Methode zu vereinfachen, sind die Methoden nach einem Namensschema benannt. Zuerst wird die Quellszene genannt und danach die Zielszene.

5.1.2 Seite Camera

Die Seite Camera wird benutzt, wenn der Anwender ein Foto zur Erhaltung der Wunde machen will. Sie besitzt insgesamt fünf Buttons und eine Textur für die Kamerabilder. Ganz oben in der Mitte befindet sich der Button zum Wechseln der Front- und Rückkamera. Die Frontkamera kann benutzt werden, wenn sich die Wunde z. B. am eigenen Kopf befindet. Mit der Rückkamera ist ein Fotografieren schlecht möglich, weil der Anwender das Display nicht sehen kann. Im unteren Bereich befinden sich drei Buttons. Mittels des rechten Buttons kann der Anwender die Galerie öffnen und die bereits gespeicherten Fotos ansehen. Der mittlere Button startet die Analyse für die Erste Hilfe und der linke Button aktiviert die Erkennung von Wunden.

5.1.3 DeviceCameraController

Die Klasse *DeviceCameraController* ist für den Zugriff auf die Kameras zuständig und erbt von der Klasse *MonoBehaviour*. Diese Klasse erfüllt mehrere Aufgaben. Eine Aufgabe ist, dass das Bild der Front- oder Rückkamera auf die Textur in der Szene gerendert wird. Andererseits stellt die Klasse dem Wundlokalisator und dem Wundklassifikator das Bild bereit. Zusätzlich wird die Live-Erkennung gestartet, wenn der Anwender diese Klasse aktiviert. Für das Bild werden jeweils drei Objekte vom Typ *WebCamDevice* und *WebCamTexture* benötigt, jeweils zwei Objekte für die Front- oder Rückkamera sowie die aktive Kameratextur. Nach der Deklaration im Klassenkopf erfolgt die Zuweisung der Geräte. Im Kopf wird ebenfalls ein Objekt vom Typ *RawImage* deklariert. Das Objekt wird mit dem GameObject *CamBackground* verdrahtet, damit die Kamera auf das Image rendern kann.

Abbildung 5.2: Seite „Camera“

Listing 5.2: Start-Methode der Klasse DeviceCameraController, unvollständig

```
void Start()
{
    if (WebCamTexture.devices.Length == 0)//Check if cam is there
    {
        Debug.Log("No devices cameras found");
        return;
    }
    Debug.Log(activeCameraDevice.kind);

    // Get the devices
    frontCameraDevice = WebCamTexture.devices.Last();
    backCameraDevice = WebCamTexture.devices.First();

    frontCameraTexture = new WebCamTexture(frontCameraDevice.name);
    backCameraTexture = new WebCamTexture(backCameraDevice.name);

    ...

    SetActiveCamera(backCameraTexture);

    ...
}
```

In Listing 5.2 ist der Quellcode der Start-Methode dargestellt. Zuerst wird geprüft, ob eine Kamera verfügbar ist. Danach wird mit der Methode *Last* auf die Frontkamera zugegriffen und ein Objekt von Typ *WebCamTexture* wird instanziiert. Der Gerätename wird dabei dem Konstruktor übergeben. Das Gleiche wird mit der Rückkamera gemacht, nur mit dem Unterschied, dass die Methode *First* benutzt wird. Die backCameraTexture wird der Methode *SetActiveCamera* übergeben. Das bedeutet, dass standardmäßig das Bild der Rückkamera angezeigt wird. Der Quellcode der Methode ist in Listing 5.3 zu finden. Zuerst wird geprüft, ob die *activeCameraTexture* nicht Null ist. Dies ist für einen Wechsel der Kamera notwendig, da bei einem Wechsel zuerst die Kamera gestoppt werden muss. Als Nächstes wird die WebCamTexture, die übergeben wurde, der *activeCamera-*

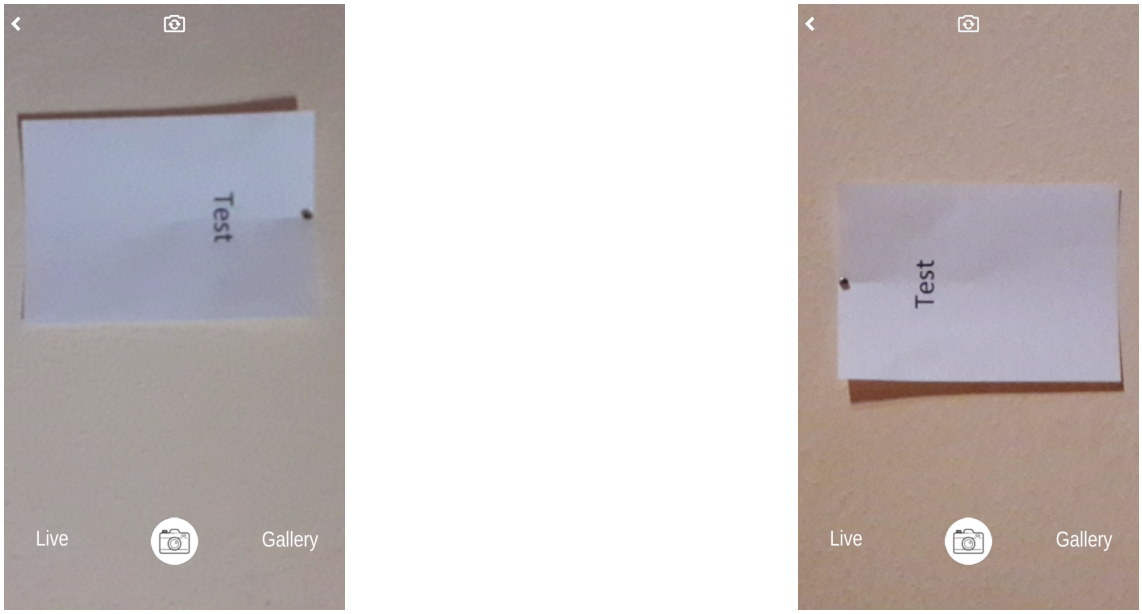
Texture zugewiesen und danach wird sie auf die Textur angewendet. Damit wird eine Entkopplung zwischen der Kamera und der Textur des Objekts *image* erreicht, wodurch zwischen der Front- und der Rückkamera gewechselt werden kann. Zum Schluss wird die Play-Methode aufgerufen, um mit dem Rendern zu beginnen.

Listing 5.3: Methode SetActiveCamera

```
public void SetActiveCamera(WebCamTexture cameraToUse)
{
    if (activeCameraTexture != null)
    {
        activeCameraTexture.Stop();
    }

    activeCameraTexture = cameraToUse;
    activeCameraDevice = WebCamTexture.devices.FirstOrDefault(device =>
    device.name == cameraToUse.deviceName);

    image.texture = activeCameraTexture;
    image.material.mainTexture = activeCameraTexture;

    activeCameraTexture.Play();

    rotationVector.z = -activeCameraTexture.videoRotationAngle;

    image.rectTransform.localEulerAngles = rotationVector;

    ...
}
```

Beim Einsatz der App auf einem mobilen Endgerät kann es passieren, dass eine Kamera den Stream mit einer anderen Rotation zur Verfügung stellt. Abbildung 5.3 zeigt zwei Bilder der Front- und der Rückkamera. Die beiden Kameras stellen die Bilder unterschiedlich bereit. Mithilfe des Attributes *videoRotationAngle* von activeCameraTexture kann der Winkel bestimmt werden, mit dem das GameObject gedreht werden muss. Im Klassenkopf ist ein Objekt vom Typ *Vector3* deklariert. Über *rotationVector.z* wird auf den z-Wert zugegriffen. Der Integer-Wert von *videoRotationAngle* muss noch negiert werden, damit im Uhrzeigersinn gedreht werden kann. Im Fall der Frontkamera wird 270 ausgewiesen und für die Rückkamera wird 90 zurückgegeben. Der neue Vektor wird *localEulerAngles* zugewiesen und Unity dreht das GameObject entsprechend.

Die Analyse der Wunde wird in der Szene *Assistance* durchgeführt. Dazu muss das Bild dieser Szene bereitgestellt werden. Sobald der Anwender den Button zum Analysieren der Wunde drückt, wird in der Klasse *PageNavigation* die nächste Szene geladen. Zusätzlich wird die Methode *TakePictureCache* der Klasse *DeviceCameraController* aufgerufen. Die Methode speichert die Textur der *activeCameraTexture* und übergibt die Textur der statischen Methode *setPicture* der Klasse *PictureStatic*. Diese dient als Zwischenspeicher. Die Klasse hat die Methode *getPicture*, mit der in der Szene *Assistance* die Textur wieder verwendet werden kann.



(a) Frontkamera (b) Rückkamera

Abbildung 5.3: Front- und Rückkamera ohne Rotation

5.1.4 Live-Erkennung von Wunden

Der Anwender hat die Möglichkeit, eine Wunderkennung durchzuführen. Für die Erkennung werden die Klassen *DeviceCameraController*, *WoundDetectorService* und *LiveAnalyse* benutzt. Abbildung 5.4 illustriert den grundsätzlichen Ablauf. Für das Anzeigen des Live-Streams wird nicht das GameObject *CamBackground* benutzt, sondern ein eigenes GameObject mit dem Namen *LiveAnalyseBackground*. Das ist notwendig, damit mit OpenCV for Unity die Bounding Boxen auf die Textur gezeichnet werden können, bevor sie auf dem RAW-Image *LiveAnalyseBackground* gerendert wird. Beide UI-Elemente haben die gleiche Größe, wobei das GameObject *LiveAnalyseBackground* deaktiviert ist, wenn die Erkennung deaktiviert ist.

Die Erkennung wird über den Button *Live* in der Szene *Camera* gestartet. Der Button führt über den Click-Listener die Methode *enableLiveAnalyzer* der Klasse *LiveAnalyse* aus. Diese ruft die Methode *toogleLiveAnalysis* des *DeviceCameraController* auf. Sie aktiviert und deaktiviert die Erkennung. Die Klasse *DeviceCameraController* hat ein Boolean-Attribut mit dem Namen *isLiveAnalysisEnabled*, das den Status speichert.

Die Methode *toogleLiveAnalysis* hat eine Switch-Anweisung mit zwei Fällen. Beim Fall, dass beim Aufruf der Methode die Variable *isLiveAnalysisEnabled* wahr ist, will der Anwender die Erkennung deaktivieren. Zuerst wird *isLiveAnalysisEnabled* auf falsch gesetzt. Danach wird das GameObject *LiveAnalyseBackground* mittels der SetActive-Methode deaktiviert. Damit auf das GameObject zugegriffen werden kann, ist im Klassenkopf ein öffentliches GameObject deklariert und in Unity verdrahtet. Die Textfarbe wird von Rot auf Weiß geändert, damit es für den Anwender erkenntlich ist, dass die Erkennung deaktiviert ist.

Der andere Teil des Switch-Case wird ausgeführt, wenn die Variable *isLiveAnalysisEnabled* falsch ist und der Anwender die Erkennung aktivieren will. Zuerst wird die Schriftfarbe des Buttons auf rot gesetzt und das GameObject *liveAnalyseBackground* wird aktiviert.

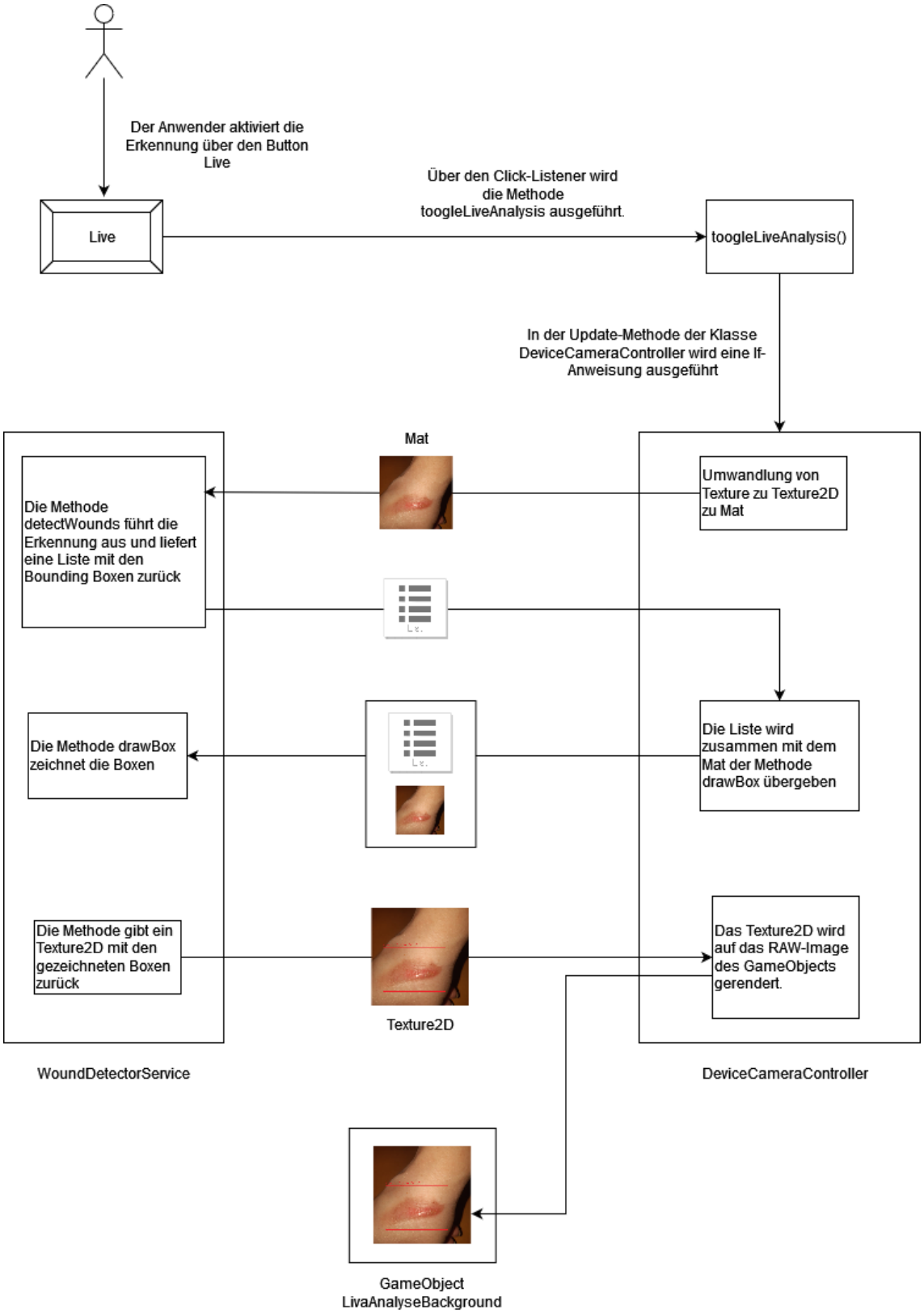


Abbildung 5.4: Übersicht Live-Erkennung von Wunden

Listing 5.4: If-Anweisung der Update-Methode

```
if (isLiveAnalysisEnabled)
{
    Texture2D newTexture = new Texture2D(activeCameraTexture.width,
        activeCameraTexture.height, TextureFormat.RGBA32, false);
    Utils.textureToTexture2D(image.texture, newTexture);
    if (Application.platform == RuntimePlatform.Android)
    {
        testTexture = ImageUtil.rotateTexture(newTexture, true);
    }

    Mat newMat = new Mat(new Size(newTexture.width, newTexture.height),
        CvType.CV_8UC3);
    Utils.texture2DToMat(newTexture, newMat);

    if(frames == 0 || frames == detectionPerFrame)
    {
        frames = 1;
        output = woundDetectorService.detectWounds(newMat);
    }

    if(output != null)
    {
        image_2.texture = woundDetectorService.drawBox(output, newMat);
    }
    frames++;
}
```

In Listing 5.4 ist der Quellcode der If-Anweisung dargestellt. Die If-Anweisung wird in der Update-Methode ausgeführt. Sie wird ausgeführt, wenn die Variable *liveAnalyseBackground* wahr ist. Zuerst wird ein neues Objekt von Typ *Texture2D* erstellt. Die Höhe und die Breite entsprechen denen von *activeCameraTexture*. Danach erfolgt die Umwandlung von einer Texture zu einer Texture2D. Als Quelle wird die Textur benutzt, auf der die Kamera gerendert wird. OpenCV for Unity besitzt nur eine Methode zum Umwandeln von Texture2D zu einem Mat und keine für Texture zu Mat. Daher muss an dieser Stelle eine Zwischenumwandlung stattfinden. Anschließend muss die Texture2D rotiert werden, wenn die App auf einer Android Plattform läuft. Das ist notwendig, weil das GameObject und nicht die Textur rotiert wird. Danach wird ein neues Mat erzeugt und mittels der Methode *texture2DToMat* umgewandelt. Die Breite und die Höhe werden von der newTexture genommen. Die Update-Methode wird für jedes Frame ausgeführt. Um die allgemeine Performance zu erhöhen, wird nicht jedes Frame durch den Wundklassifikator analysiert. Die Variable *frame* wird benutzt, um nur bei einem bestimmten Frame eine Erkennung auszuführen. *frames* wird mit 0 initialisiert. Die erste If-Anweisung besitzt zwei Bedingungen. Die erste Bedingung *frame == 0* stellt sicher, dass die Erkennung für den allerersten Frame durchgeführt wird. Die zweite Bedingung *frames == detectionPerFrame* definiert, in welchem Intervall eine Erkennung durchgeführt wird. Dazu wird die Variable *detectionPerFrame* benutzt. Die Variable kann über den Inspektor von Unity verändert werden. Innerhalb der If-Anweisung wird zuerst die Variable *frames* auf 1 gesetzt und die Methode *detectWounds* wird aufgerufen. Übergeben wird dabei das Mat.

Listing 5.5: detectWounds-Methode

```
public Mat detectWounds(Mat frame)
{
    Mat resizeimage;
    resizeimage = Dnn.blobFromImage(frame, 1.0, new Size(widthOfResize,
        heightOfResize));
    model.setInput(resizeimage);
    Mat output = model.forward();
    return output;
}
```

Die Methode *detectWounds* nimmt ein Mat entgegen. Die Methode befindet sich in der Klasse *WoundDetectorService*. Die Initialisierung des Modells wird in Abschnitt 5.3.5 beschrieben. Zuerst wird ein neues Mat deklariert, ein Blob wird erzeugt und mit der Methode *forward* wird die Erkennung der Wunden durchgeführt. Um auf die Boxen zuzugreifen, wird mit reshape das Ergebnis umgeformt. Nachdem Umformen hat das Mat *output* die Form $N \times 7$ und wird zurückgeliefert.

Nachdem die Bounding-Boxen bestimmt wurden, wird in der Klasse *DeviceCameraController* die zweite If-Anweisung ausgeführt. Die Anweisung übergibt der Methode *drawBox* das Mat sowie die Bounding Boxen und zeichnet diese in das Mat. Der Quellcode für diese Methode befindet sich in Listing 5.6. Die For-Schleife läuft über das Mat. Zuerst werden Variablen für die Confidence, für die Klassen-ID, sowie für den Klassennamen deklariert. Das Array *data* enthält die einzelnen Informationen für die erkannte Box. Mithilfe der Get-Methode werden die Informationen in das Array geschrieben. Danach wird der Confidence-Wert vom Array ausgelesen. Der Confidence-Wert unterscheidet sich von Box zu Box. In der If-Anweisung wird geprüft, ob der Confidence-Wert höher als ein bestimmter Schwellenwert ist. Der Schwellenwert kann über den Inspektor in Unity eingestellt werden, damit nicht der Quellcode verändert werden muss. Er ist standardmäßig mit 0,75 angegeben. Wenn der Schwellenwert überschritten ist, wird die If-Anweisung ausgeführt.

Innerhalb der If-Anweisung werden zuerst die Klassen-ID und danach der Klassenname ausgelesen. Die Klassennamen ist in dem String-Array gespeichert. Die Liste kann ebenfalls über den Inspektor bearbeitet werden. Die Klassennamen entsprechen den in Tabelle 4.2 festgelegten Bezeichnungen. Die nächsten vier Anweisungen lesen die Punkte aus, die für die Erstellung der Box notwendig sind. Die x-Werte und y-Werte müssen mit der Breite bzw. mit der Höhe multipliziert werden, damit die geänderte Größe des Mats berücksichtigt wird, da bei der Erkennung ein Mat der Größe 320×320 Pixel verwendet wird.

Listing 5.6: drawBox-Methode

```
public Texture2D drawBox(Mat output, Mat frame)
{
    int sizeMaxBoxes = output.size(2);

    for (int i = 0; i < sizeMaxBoxes; i++)
    {
    float confidence;
    int class_id;
    string className;
    float[] data = new float[7];

    output.get(i, 0, data);
```

```
    confidence = data[2];

    if (confidence > confidenceThreshold)
    {
        class_id = (int)data[1];
        className = labels[class_id - 1];

        float box_x1 = data[3] * frame.width();
        float box_y1 = data[4] * frame.height();

        float box_x2 = data[5] * frame.width();
        float box_y2  = data[6] * frame.height();

        Point vertex1 = new Point(box_x1, box_y1);
        Point vertex2 = new Point(box_x2, box_y2);
        Imgproc.rectangle(frame, vertex1, vertex2, new Scalar(0, 0, 255),
            3);

        Imgproc.putText(frame, className + ": " + confidence, new Point(
            box_x1, box_y1 - 5), Imgproc.FONT_HERSHEY_SIMPLEX, 1, new
            Scalar(0, 0, 255), 3);
        }
    }

        Texture2D texture2D = new Texture2D(frame.width(), frame.height()
            , TextureFormat.RGBA32, false);
        Utils.matToTexture2D(frame, texture2D);

        return texture2D;
}
```

Als Nächstes werden zwei Objekte vom Typ *Point*, die die Eckpunkte des zu zeichnenden Rechtecks darstellen, erstellt. Die Methode *Imgproc.rectangle* zeichnet in das übergebene Mat das Rechteck. Übergeben werden ebenfalls die zwei Punkte, ein Scalar-Objekt für die Farbe der Linien und ein Integer, der die Dicke der Linie festlegt. Das Scalar-Objekt besteht aus drei Werten, welche dem RGB-Format entsprechen. Danach werden die Klasse und der Confidence-Wert über die Box eingefügt. Dafür ist die Methode *Imgproc.putText* zuständig. Der Methode werden mehrere Informationen übergeben. Es werden das Mat und ein String mit dem Text, sowie ein Punkt übergeben, an dem das Textfeld platziert wird. Ebenfalls wird die Schriftart, ein Scalar-Objekt für die Schriftfarbe und eine Zahl für die Dicke übergeben. Zum Schluss wird das Mat wieder in eine Textur umgewandelt und an die Klasse *DeviceCameraController* zurückgegeben.

5.1.5 DAO

Die Klasse *DAO* ist für die Speicher- und Leseoperationen zuständig. Die DAO-Klasse nutzt dafür drei Repositories. Die Repositories benutzen das Dateisystem. Die Klassen *History* und *Foto* werden als Java Script Object Notation (JSON) String abgelegt. Das Bild der Wunde wird getrennt im Portable Network Graphic (PNG) Format gespeichert. Die Klasse *DAO* sowie alle Repositories erben von MonoBehaviour und werden als GameObject in der Szene angelegt, in der sie benutzt werden. Alle Repositories haben ein öffentliches und ein privates String-Attribut. Das öffentliche Attribut spezifiziert den Ordner, in dem die JSON-Dateien gespeichert werden und kann über den Inspektor festgelegt werden. Das private Attribut wird innerhalb der Klasse benutzt und speichert den Pfad. In Listing 5.7 ist der Aufbau der Start-Methode exemplarisch dargestellt. Die Repositories unterscheiden sich bei der Benennung der Attribute. *pathToHistory* setzt sich zusammen

aus dem Rückgabewert von Application.persistentDataPath und aus dem Attribut folderHistory. Application.persistentDataPath gibt einen Verzeichnispfad zurück. Dieser ändert sich je nach Plattform. Anschließend wird geprüft, ob das Verzeichnis bereits existiert. Sollte das Verzeichnis nicht existieren, wird es erstellt.

Listing 5.7: Aufbau Start-Methode der Repositories

```
void Start()
{
    pathToHistroy = Application.persistentDataPath + "/" + folderHistory;

    if (Directory.Exists(pathToHistroy) == false)
    {
        Directory.CreateDirectory(pathToHistroy);
    }
}
```

Die Klasse *FotoRepository* besitzt drei Methoden. In der Methode *SaveFoto* wird zuerst der gesamte Pfad inklusive Dateinamen in einer String-Variablen gespeichert. Dazu wird das Attribut *pathToFotos* mit der ID des Fotos und *".json"* konkateniert. Als Nächstes wird das Foto-Objekt in JSON umgewandelt. Dafür wird die statische Methode *JsonUtility.ToJson()* verwendet. Es können nur einfache Attribute und Feldtypen umgewandelt werden, die serialisierbar sind. Nicht serialisierbar sind private Felder und statische Felder sowie, Felder mit dem Attribut NonSerialized. Diese werden ignoriert. Das Datum des Fotos ist vom Typ DateTime und kann nicht serialisiert werden. Der Modellklasse *Foto* wird daher ein zusätzliches Attribut *DateAsString* vom Typ String hinzugefügt. Der Methode wird das Foto-Objekt und ein Boolean übergeben. Das Boolean steuert, ob der Output lesbar sein soll. Im Falle von False werden alle Leerzeichen und Zeilenumbrüche entfernt. Der Rückgabewert wird in einem String und mit der Methode *File.WriteAllText* gespeichert. Die Methode nimmt den Pfad sowie den JSON-String als Parameter entgegen. Zum Laden eines Fotos wird die Methode *LoadFoto* benutzt. Der Methode wird die ID des Fotos übergeben, das geladen werden soll. Zuerst wird der Pfad erstellt. Dieser wird wie beim Speichern mit dem Pfad zum Verzeichnis, mit der übergebenen ID und mit dem String *".json"* konkateniert. Der Aufruf *foto = JsonUtility.FromJson<Foto>(File.ReadAllText(path));* lädt zuerst die JSON-Datei und wandelt das JSON-Objekt dann in ein Foto-Objekt um. Die Methode *FromJson* ist generisch. Im Diamant-Operator muss die Klasse angegeben werden, in die umgewandelt werden soll. Abschließend wird der String, der das Datum enthält, wieder in DateTime umgewandelt. Für das Anzeigen der Fotos in der Galerie ist es notwendig, dass alle Fotos als Liste geladen werden. Hierfür hat die Klasse *FotoRepository* die Methode *getAllFotos*, die eine Liste mit allen Foto-Objekten zurück gibt. Der Quellcode ist in Listing I.1 dargestellt. Zuerst wird eine Liste vom Typ Foto instanziiert und ein regulärer Ausdruck der Form *[0-9]** wird erstellt. Danach wird das Verzeichnis mit allen Fotos ausgelesen. In dem Array *fileInfo* stehen alle Dateipfade. Innerhalb einer ForEach-Schleife wird iterativ über alle Pfade gegangen. Die ID muss aus dem Pfad ausgeschnitten werden, deshalb wird der reguläre Ausdruck benutzt. Danach wird die ID von einem String in einen Integer umgewandelt und der Methode *LoadFoto* übergeben. Anschließend wird das Foto zur Liste hinzugefügt.

Die Klasse *HistoryRepository* ist gleich aufgebaut wie das *FotoRepository*. Abweichend sind nur das Speichern und das Laden des Arrays. Das Serialisieren und das Deserialisieren von einen Foto-Array ist nicht möglich. Die Klasse *History* wird daher um ein int-Array ergänzt, das die IDs der Fotos speichert. Der Aufruf *(int[])history.listOfFotos.ToArray(typeof(int))* wandelt das Foto-Array in ein Array mit IDs um. Der Methode *ToArray* wird der Typ angegeben, in den umgewandelt werden muss. Zusätzlich muss noch ein expliziter

Cast vorgenommen werden.

Die Methode *SavePicture* der Klasse *PictureRepository* speichert das Foto. Mittels der Klasse *Counter* wird eine fortlaufende Nummer generiert und diese wird als Dateiname verwendet. Der Pfad wird mit der ID konkateniert. Die Methode *File.WriteAllBytes* schreibt das Bild auf das Speichersystem. Die LoadPicture-Methode lädt das Bild vom Speichersystem. Listing 5.8 zeigt die Methode. Zuerst wird der Pfad festgelegt und zwei Mats werden deklariert. Mithilfe der Methode *Imgcodecs.imread* kann das Bild geladen und dem Mat *orginalImg* zugewiesen werden. Anschließend muss das Mat in einen anderen Farbraum konvertiert werden, damit es richtig dargestellt werden kann. Die Methode *LoadPictureTexture2D* wird für das Laden einer Miniaturansicht der Bilder benutzt. Die Miniaturansicht wird in verschiedenen Szenen verwendet und der Quellcode dafür befindet sich in Listing I.2. Der Unterschied zwischen den Methoden ist, dass eine Texture2D zurückgegeben wird.

Listing 5.8: LoadPicture-Methode

```
public Mat LoadPicture(int id)
{
    string path = pathToPicture + "/";
    Mat orginalImg;
    Mat picture;

    orginalImg = Imgcodecs.imread(path + id + ".png", Imgcodecs.
        IMREAD_ANYCOLOR);

    picture = new Mat(orginalImg.rows(), orginalImg.cols(), CvType.
        CV_8UC3);
    Imgproc.cvtColor(orginalImg, picture, Imgproc.COLOR_RGBA2BGR);
    return picture;
}
```

5.1.6 Seite Assistance

Die Seite *Assistance* gibt dem Anwender die Handlungsempfehlungen. In Abbildung 5.5 ist die Seite dargestellt. Im oberen Bereich auf der rechten Seite befindet sich der Button zum Speichern des Fotos. Darunter werden das erkannte Körperteil und der Confidence-Wert sowie der Wundtyp und dessen Confidence-Wert angezeigt. Im unteren Bereich erscheinen die Handlungsempfehlungen. Auf der rechten und linken Seite des Textes befinden sich Buttons zum Anzeigen der nächsten bzw. der vorherigen Empfehlung.

Die Klasse *AssistanceManager* stellt als zentrale Komponente alle wesentlichen Funktionen, die in der Szene verwendet werden, bereit. In der Start-Methode wird zuerst das Bild geladen. Das Bild wird über die statische Methode *getPicture* der Klasse *PictureStatic* geladen. Das Bild wird, wie bei der Seite Camera, auf einen Hintergrund gerendert. In einer If-Anweisung wird geprüft, ob die App auf einem Android-Gerät läuft. Sollte das der Fall sein, muss das Bild rotiert werden, um es korrekt darzustellen.

Der Anwender hat die Möglichkeit, das aufgenommene Foto zu speichern. Das Icon zum Speichern ruft die Methode *savePicture* auf, die sich in der Klasse *SavePicture* befindet. In der Methode wird zuerst das GameObject *toastPicture* aktiviert. Das GameObject wird als Toast benutzt. Dieses wird als kleines Pop-up dargestellt und gibt dem Anwender Feedback, dass das Bild gespeichert wird. Das GameObject besitzt eine Text-Komponente, die standardmäßig deaktiviert ist. Als Nächstes wird durch den Aufruf *GetNewFoto* vom der Klasse *modelObjectBuilder* ein neues Objekt von Typ *Foto* konstruiert. Danach wird der

Toast für eine Sekunde eingeblendet, in dem das GameObject *Toast* für die entsprechende Zeit aktiviert und danach deaktiviert wird.

Danach wird durch den Aufruf *GetNewFoto* von der Klasse *modelObjectBuilder* ein neues Objekt vom Typ *Foto* konstruiert. Bei der Konstruktion wird beim Objekt über den Konstruktor eine fortlaufende ID-Nummer vergeben. Die ID wird von der Hilfsklasse *Counter* generiert. Diese hat ein Integer und eine Methode, die dann aufgerufen wird. Die Methode inkrementiert dann den Integer. Als nächster Schritt wird über den Aufruf *PictureStatic.getPicture()* das Bild als Texture2D einem Objekt zugewiesen und mit der Methode *EncodeToPNG* in ein Byte-Array umgewandelt. Das Array wird der Methode *savePicture* der Klasse *DAO* übergeben. Als Rückgabewert wird die ID des *Picture* zurückgegeben. Anschließend wird die ID dem Attribut *IdOfPicture* des Fotos zugewiesen. Danach wird das Datum über den Aufruf *System.DateTime.Now.ToString()* und *System.DateTime.Now* dem Attribut *DateAsString* bzw. *Date* das Datum erstellt.

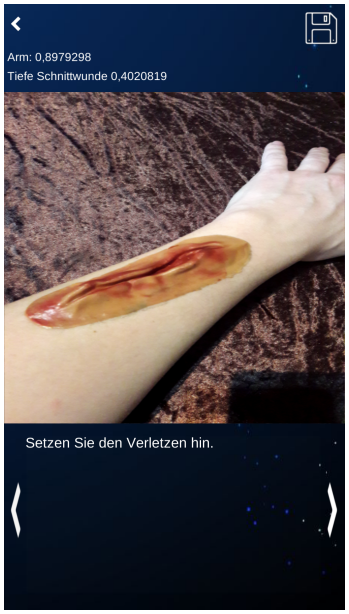


Abbildung 5.5: Seite Assistance

Die Handlungsanweisung kann zusätzlich mit Symbolen ergänzt werden. Exemplarisch dazu wird ein Pflaster benutzt, das über die Wunde gelegt werden kann. In diesem Abschnitt wird beschrieben, wie die Touch-Steuerung funktioniert. Das Pflaster wird nur bei bestimmten Handlungsempfehlungen angezeigt und die Funktionsweise dazu wird im Abschnitt 5.4 erläutert. Die Touch-Steuerung wird in der Update-Methode der Klasse *TouchControl* realisiert. Der Quellcode dafür ist in Listing I.3 zu finden. Die Touch-Eingabe wird im Prinzip in zwei If-Anweisungen realisiert. Die erste If-Anweisung hat als Bedingung, dass zwei Finger das Display berühren. Dies kann mit der Anweisung *Input.touches.Length == 2* überprüft werden. Zuerst werden die beiden Objekte *t1* und *t2* instanziiert. Die beiden Objekte stellen die Finger dar. Danach wird mit einer If-Anweisung überprüft, ob der Anwender mit der Gestensteuerung begonnen hat und die Distanz zu den beiden Fingern wird in einem Vektor gespeichert. Ebenso werden die Werte für die Skalierung des GameObjects gespeichert. Im Else-Teil der If-Anweisung wird geprüft, ob die Steuerung fortgesetzt wird bzw. ob sich die Finger noch auf dem Display befinden. In diesem Teil der Anweisung kann das Pflaster rotiert werden und die Größe kann verändert werden. Für die Veränderung der Größe wird der Abstand zwischen den Fingern mittels *Vector2.Distance* berechnet. Anschließend wird über das Verhältnis zwischen

der aktuellen Distanz und der Distanz beim Beginn ein Faktor erstellt. Dieser wird mit den Werten der Skalierung multipliziert und dem GameObject erneut zugewiesen, um die Größenveränderung durchzuführen. Die Rotation des Pflasters wird über die Berechnung des Winkels der Positionen der beiden Finger realisiert. Die Methode *Atan2* berechnet den Winkel. Die Methode *Quaternion.Euler* führt die Rotation an den Achsen durch. Bei der Übergabe wird der Winkel mit der Konstante *Rad2Deg* multipliziert. Die Konstante entspricht dem Wert $360/(PI * 2)$ und sorgt für die Umwandlung von einem Radiant zu einem Winkel. Die zweite If-Anweisung sorgt für die Bewegung des Pflasters. Listing I.3 zeigt den dafür notwendigen Quellcode. Im Gegensatz zur Rotation und zur Skalierung ist für die Bewegung nur ein Finger notwendig. Die Bedingung *Input.touches.Length == 1* überprüft, ob nur ein Finger verwendet wird. In der ersten If-Anweisung wird geprüft, ob mit der Gestensteuerung begonnen wurde. Die Position des Fingers wird über den Vektor *position* gespeichert. Die neue Position des Fingers wird dann dem GameObject zugewiesen.

Listing 5.9: Update-Methode der Klasse TouchControl, Teil 1

```
if (Input.touches.Length == 2)
{
    Touch t1 = Input.touches[0];
    Touch t2 = Input.touches[1];

    if (t1.phase == TouchPhase.Began || t2.phase == TouchPhase.Began)
    {
    initialFingersDistance = Vector2.Distance(t1.position, t2.position);
    initialScale = gameObject.transform.localScale;
    }
    else if (t1.phase == TouchPhase.Moved || t2.phase == TouchPhase.Moved
        )
    {
        var currentFingersDistance = Vector2.Distance(t1.position, t2.
            position);
        var scaleFactor = currentFingersDistance / initialFingersDistance
            ;

        gameObject.transform.localScale = initialScale * scaleFactor;

        Vector3 diff = t1.position - t2.position;
        var angle = (Mathf.Atan2(diff.y, diff.x));
        transform.rotation = Quaternion.Euler(0f, 0f, Mathf.Rad2Deg *
            angle);
}
```

Die Klasse *AssistanceList* gibt dem Anwender die Möglichkeit zwischen den einzelnen Anweisungen hin und her zu blättern. Der gesamte Quellcode befindet sich in Listing I.10.

5.1.7 Seite History

Die Szene *History* beinhaltet einen Button zum Erstellen des Wundverlaufs sowie eine Unity-Scroll-View. Der Button besitzt einen Click-Listener, der die Methode *HistoryToWoundCourse* im *PageNavigator* ausführt und die Szene *WoundCourse* lädt. Die Scroll-View ist ein UI-Element von Unity. In der Liste befindet sich ein GameObject, das als Vorlage für die übrigen gilt. Jeder Wundverlauf wird mit dem Erstellungsdatum sortiert und in der Liste angezeigt. Die Klasse *HistoryList* ist für das Laden der Wundverläufe zuständig. Der Quellcode ist in Listing I.4 dargestellt. Im Klassenkopf werden Objekte deklariert,

die später benötigt werden. Unter anderem wird das Template-GameObject geladen. Der Aufruf *dao.getAllHistories* lädt alle Wundverläufe vom Speichersystem. Danach folgt eine If-Anweisung, die prüft, ob die Liste leer ist. Sollte die Liste leer sein, wird dem Anwender ein Text eingeblendet, dass kein Wundverlauf vorhanden ist. Für den Fall, dass die Liste nicht leer ist, wird iterativ mittels einer Foreach-Schleife über die Liste gegangen. Mit dem Aufruf *Instantiate* wird das Template-GameObject geklont. Danach wird vom History Objekt das Datum und die ID der Wunde von dem Attribut *HistoryId* dem neuen GameObject zugewiesen. Das Attribut befindet sich in der Klasse *HistoryManager* und wird über *GetComponentInChildren<HistoryManager>()* gefunden. Abschließend wird das Template-GameObject mittels der Destroy-Methode von der Liste gelöscht.

Jedes GameObject besitzt ein weiteres GameObject, das das Skript *HistoryManager* hat. Die Klasse erfüllt zwei Aufgaben. Zum einen speichert sie die ID des Wundverlaufs. Zum anderen ist die Methode *showHistory* für das Laden der Szene *WoundCourseView* verantwortlich. Das Skript hat ein Attribut und eine Methode. Die erste Anweisung in Listing 5.14 weist dem statischen Attribut historyId die ID der Wunde zu. Das ist notwendig, damit die Szene *WoundCourseView* weiß, welcher Wundverlauf zu laden ist.

Listing 5.10: showHistory-Methode der Klasse HistoryManager

```
public void showHistory()
{
    WoundCourseViewManager.historyId = gameObject.GetComponentInChildren<
        HistoryManager>().HistoryId;
    SceneManager.LoadScene("WoundCourseView");
}
```

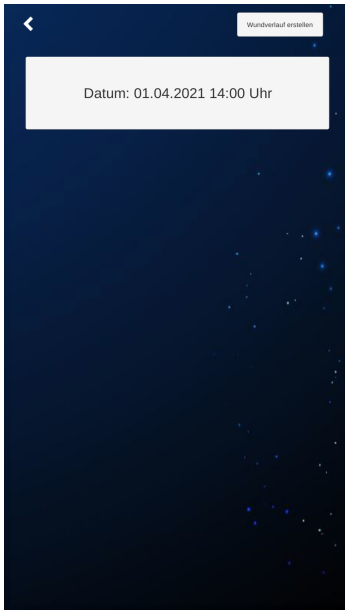


Abbildung 5.6: Seite History

5.1.8 Seite WoundCourseView

Die Szene *WoundCourseView* wird für das Anzeigen des Wundverlaufs benutzt. Es sind zwei Eingabefelder vorhanden, die der Anwender entsprechend benutzen kann. Im unteren Bereich befindet sich eine Scroll-View zum Anzeigen der Fotos. Für jedes Foto in der Liste werden das Foto in der Miniaturansicht sowie das Erstellungsdatum mit der Uhrzeit angezeigt. Das Foto hat einen Click-Listener, der das Foto größer anzeigt, wenn der Anwender auf das Foto klickt. Im oberen rechten Bereich befinden sich zwei Buttons. Der rechte Button dient zum Speichern des Wundverlaufs und der linke zum Hinzufügen eines Fotos.

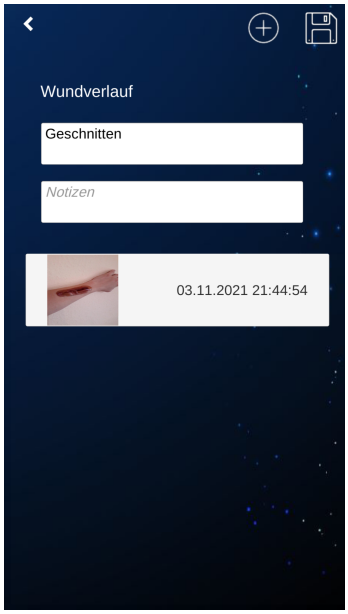


Abbildung 5.7: Seite WoundCourseView

Die Klasse *WoundCourseViewManager* ist für das Initialisieren der Szene verantwortlich. Das statische Attribut *historyId* beinhaltet die ID des Wundverlaufs, der geladen werden soll. In der Methode *Awake* wird dieser durch die DAO-Klasse geladen. Anschließend werden die Eingabefelder befüllt. Die Initialisierung muss in der Awake-Methode durchgeführt werden, da der Wundverlauf als Erstes geladen werden muss. Ansonsten kommt es zu einer NullPointerException, da andere Klassen bereits in Start-Methode auf den Manager zugreifen.

Danach müssen alle Fotos des Wundverlaufs geladen werden. Dafür ist die Klasse *WoundCourseViewFotoList* verantwortlich. Der Quellcode der Klasse befindet sich in Listing I.5. Nach der Deklaration der notwendigen Objekten werden alle Fotos des Wundverlaufs in einer ArrayList geladen. Das Array wird anschließend sortiert und die Reihenfolge der Bilder wird mit der Methode *Reverse* geändert, damit sich in der Liste die neusten Bilder oben befinden. Danach wird geprüft, ob die ArrayList leer ist. Sollte die Liste leer sein, wird der Text „Keine Fotos hinzugefügt“ angezeigt. Der Text ist im Attribut *textNoFoto* gespeichert. Der Textinhalt kann über den Inspektor festgelegt werden. Im Fall, dass die Liste nicht leer ist, wird mit einer Foreach-Schleife über die Fotos iteriert. Der Aufbau gleicht dabei der Klasse *HistoryList*, bei der ein Template-GameObject benutzt wird. Zuerst wird dieses Template-GameObject geklont und die entsprechenden Informationen wie das Datum und die ID des Fotos werden der Klasse *FotoManager* zugewiesen. Die Klasse ist ebenfalls gleich aufgebaut wie die Klasse *HistoryManager* und erfüllt die gleichen Aufgaben für die Bilder im Wundverlauf. Als nächster Schritt wird ein

Objekt vom Typ *Texture2D* für die Miniaturansicht erstellt. Für die Miniaturansicht gibt es ein weiteres GameObject mit der Komponente *RawImage*. Dies ist ein Kind-Element. Die Textur hat eine Höhe und Breite von jeweils 150 Pixeln. Die Methode *LoadPictureTexture2D* der DAO-Klasse lädt das Bild. Dabei wird die ID des Bildes übergeben. Über den Aufruf *game.GetComponentInChildren<RawImage>().texture = texture2D;* wird auf das Kind-GameObject mit dem RawImage zugegriffen und der Textur zugewiesen.

Für einen besseren Überblick über den Ablauf des Hinzufügens eines Fotos, ist in Abbildung 5.8 der Ablauf skizziert. Zum Hinzufügen der Fotos wird über den Click-Listener die Methode *addFoto* der Klasse *FotoAddWoundCourseView* aufgerufen. Unity löscht den Inhalt der beiden Eingabefelder, wenn das GameObject *Popup* aktiviert wird. Daher ist es notwendig, die Felder zu cachen. Die Methode *cacheHistory* führt dies aus. Danach wird das GameObject aktiviert, welchem dem Anwender Feedback gibt, ob Fotos vorhanden sind. Im Anschluss wird das GameObject *Popup* aktiviert. *Popup* enthält eine Scroll-View mit allen Bildern. Für das Laden der Bilder ist die Klasse *GalerieListViewPopup* verantwortlich. Die Klasse *GalerieListViewPopup* funktioniert wie die Klasse *WoundCourseViewFotoList*, jedoch mit dem Unterschied, dass nicht die Fotos des Wundverlaufs geladen werden, sondern alle Fotos die im Speichersystem zur Verfügung stehen. Dazu wird die Methode *getAllFotos* der Klasse *DAO* aufgerufen. Jedes Foto hat einen Click-Listener. Dieser ruft die Methode *addFotoById* der Klasse *woundCourseViewManager* auf und übergibt das Attribut *IdOfFoto*. Die Methode fügt das Bild der Liste im History-Objekt hinzu. Anschließend wird das GameObject *Popup* wieder deaktiviert. Abschließend muss die Scroll-View, mit den Fotos des Wundverlaufs, aktualisiert werden. Die Methode *updateList* muss zuerst, wie in Listing 5.11 gezeigt, gelöscht werden. Alle Fotos haben das Label *FotosWound*. Über dieses Label werden alle Fotos gesucht und in eine Liste geschrieben. Danach wird über die Liste iteriert und die Bilder werden mittels der Destroy-Methode entfernt.

Der Wundverlauf wird mit dem oberen rechten Button gespeichert. Bei einem Klick des Anwenders auf diesen Button wird die Methode *saveHistory* in der Klasse *WoundCourseViewManager* aufgerufen. Bevor das History-Objekt der DAO-Klasse zum Speichern übergeben wird, muss noch das Datum in einen String umgewandelt werden und dem Attribut *DateAsString* zugewiesen werden.

Listing 5.11: Löschen der GameObjects in der Scroll-View

```
GameObject[] currentFotos;

//Clears the list
currentFotos = GameObject.FindGameObjectsWithTag("FotosWound");
foreach (GameObject gameObject in currentFotos)
{
    Destroy(gameObject);
}
```

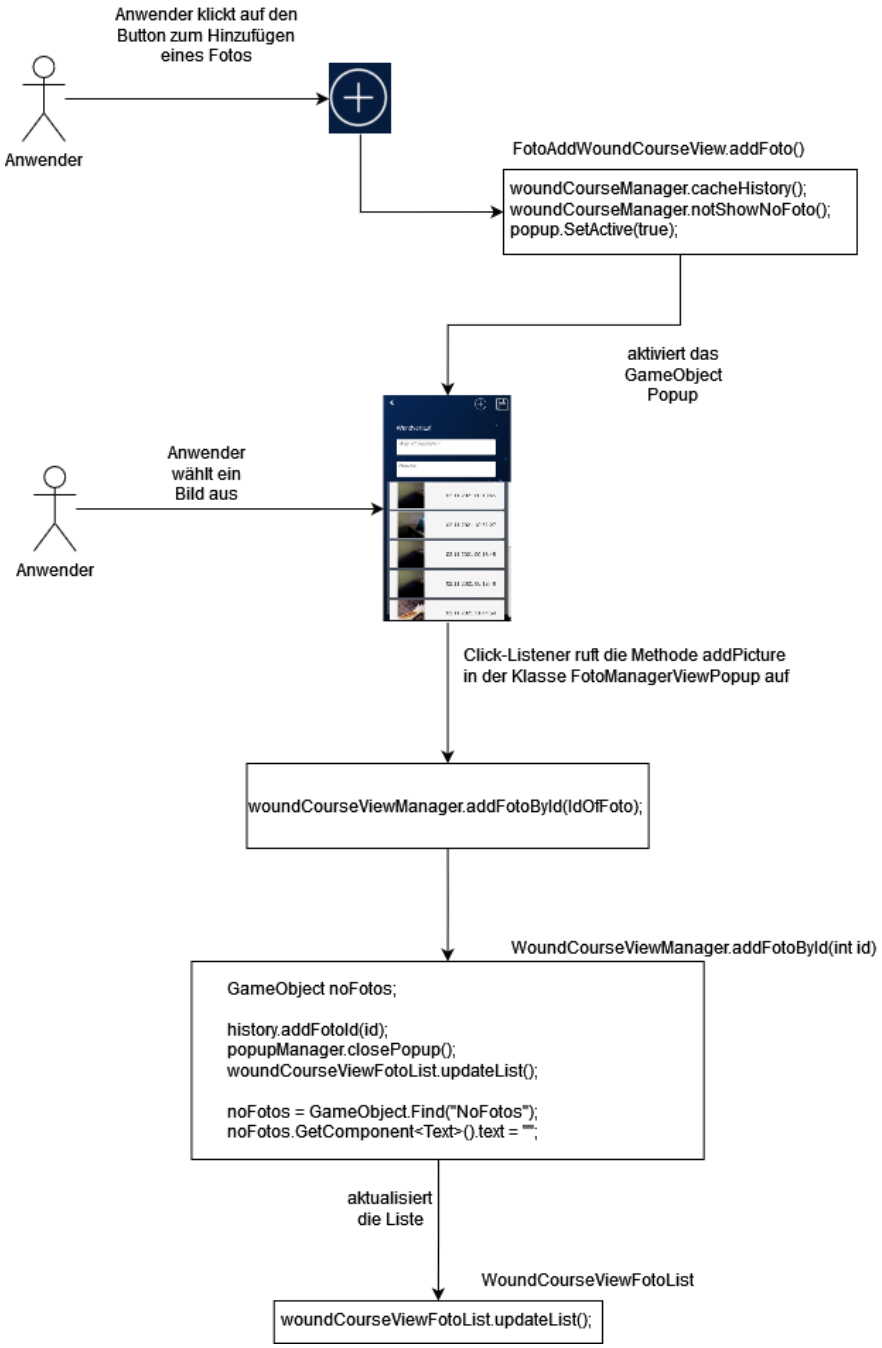
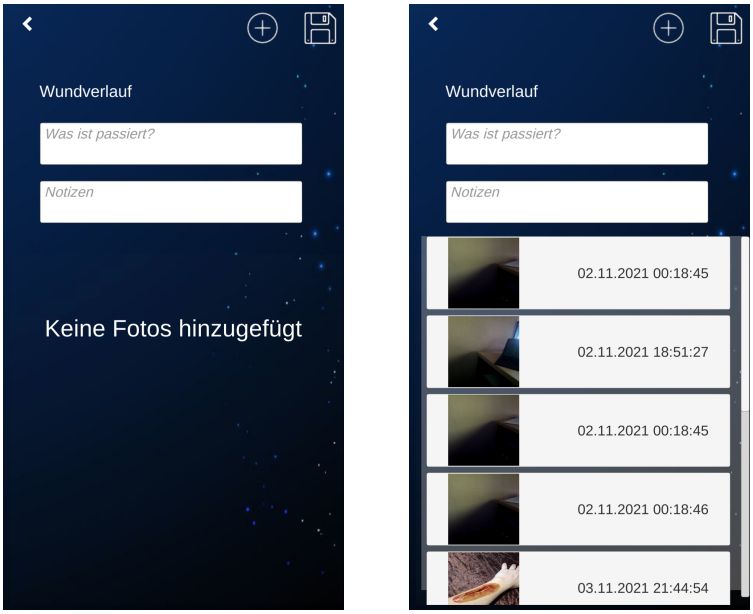


Abbildung 5.8: Ablauf für das Hinzufügen eines Bildes

5.1.9 Seite WoundCourse

Die Szene *WoundCourse* ist über die Szene *History* erreichbar. Sie dient dem Anwender dazu, einen neuen Wundverlauf zu erstellen. Das Hinzufügen von Wundbildern, das Anzeigen der bereits hinzugefügten Wundbilder und das Speichern passieren in Abschnitt 5.1.8 beschrieben. Unterschiedlich ist im Wesentlichen nur das initiale Laden. Während bei der Szene *WoundCourseView* der Wundverlauf vom Speichersystem geladen wird, wird bei der Szene *WoundCourse* in der Awake-Methode über den Aufruf *history = modelObjectBuilder.GetNewHistory();* ein neues Objekt vom Typ *History* erzeugt.

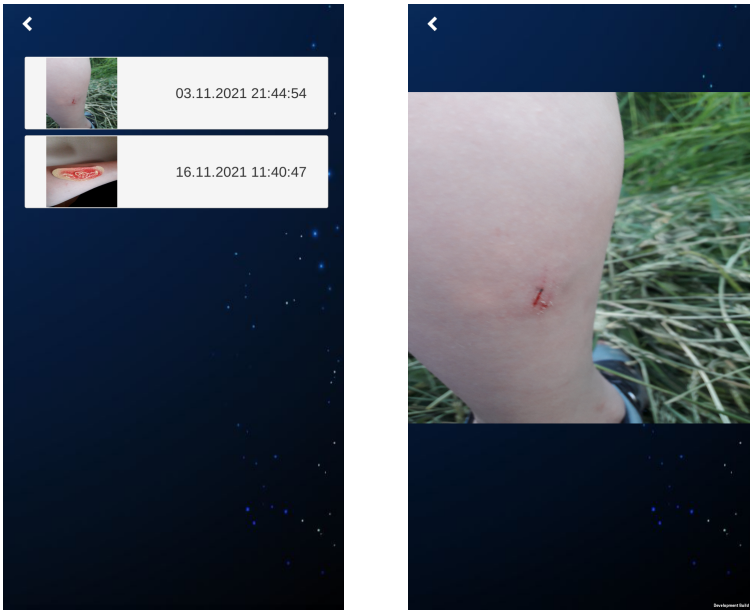


(a) Leerer Wundverlauf (b) Popup

Abbildung 5.9: Seite WoundCourse

5.1.10 Seite Galerie und Foto

Die Szene Galerie bietet dem Anwender die Möglichkeit, alle gemachten Fotos anzuzeigen. Die Szene besitzt eine Scroll-View für die Wundbilder. Das Laden der Wundbilder wird von der Klasse *GalerieList* durchgeführt. Die Klasse ist genauso aufgebaut wie der in Listing I.5 beschriebene Quellcode. Der Unterschied ist nur, dass anstelle des Aufrufs *arrayOfFotos = woundCourseViewManager.getArrayOfFotosId();* der Aufruf *arrayOfFotos = dao.getAllFotos();* erfolgt. Alle Bilder werden also direkt vom Speichersystem geladen. Wie beim Popup ist jedem Bild ein weiteres GameObject untergeordnet, das die Klasse *FotoManager* beinhaltet. Diese enthält die Methode *showPicture*, die vom Click-Listener aufgerufen wird, wenn der Anwender ein Bild anklickt. Nachdem Aufrufen der Methode wird die Id des Bildes dem statischen Attribut *fotoId* der statischen Klasse *FotoStatic*. Das ist notwendig, damit die ID in der Szene Foto geladen werden kann. Anschließend wird die Szene Foto geladen. Die Klasse *FotoLoader* lädt das Foto daraufhin vom Speichersystem. Der Quellcode für die Klasse befindet sich im Listing I.7. Nach der Deklaration von Objekten in der Methode *Start* wird zuerst das Foto-Objekt über die *FotoId* mittels der DAO-Klasse geladen. Nach dem Laden des Foto-Objekts kann das Bild mit dem Aufruf *imageAsMat = dao.LoadPicture(foto.IdOfPicture);* geladen werden. Das Mat muss noch in eine Texture2D umgewandelt werden. Die Textur wird dann auf das GameObject *CamBackground* gerendert.



(a) Seite Galerie (b) Foto

Abbildung 5.10: Screenshot der Seiten Galerie und Foto

5.2 Wundlokalisator

Für den Wundlokalisator soll ein Deep-Learning-Modell verwendet werden. Dazu eignet sich eine auf CNN basierende Architektur. TensorFlow wird als Framework benutzt, um das Modell zu erstellen. TensorFlow ist eines der meist genutzten Frameworks für Machine-Learning. Das von Google entwickelte Open-Source-Framework läuft auf verschiedenen Hardware-Plattformen, sowohl auf Desktops und Servern als auch auf mobilen Endgeräten. Es hat eine große Community, was für eine Entwicklung förderlich ist, weil genügend Ressourcen wie Bücher, Foren und Tutorials zur Verfügung stehen. TensorFlow unterstützt die Verwendung einer Central Processing Unit (CPU) und GPU. [vgl. 47, S.425] Die Entwicklung des Modells findet außerhalb der Entwicklungsumgebung der App statt. Es wird dazu Jupyter verwendet, das in der Anaconda-Plattform verfügbar ist. Es wird die GPU einer Grafikkarte vom Typ NVIDIA GeForce GTS 1050 Ti verwendet, damit die Trainingsdauer verringert wird.Bevor auf die einzelnen Schritte der Entwicklung eingegangen wird, wird in Abbildung 5.11 das generelle Vorgehen dargestellt. Das gesamte Notebook ist Anhang G zu finden.

Zuerst wird der Datensatz erstellt und als TF-Dataset in das Notebook geladen. Die Architektur wird danach angepasst und die Hyperparameter werden festgelegt. Dann folgt das Training des Modells. TensorFlow erstellt beim Training Checkpoint-Dateien. Diese werden nach dem Training benutzt, um den Frozen-Graph zu erstellen und zu exportieren. Der Frozen-Graph kann in der App importiert werden, um die Klassifizierung durchzuführen (siehe Abschnitt 5.3.5).

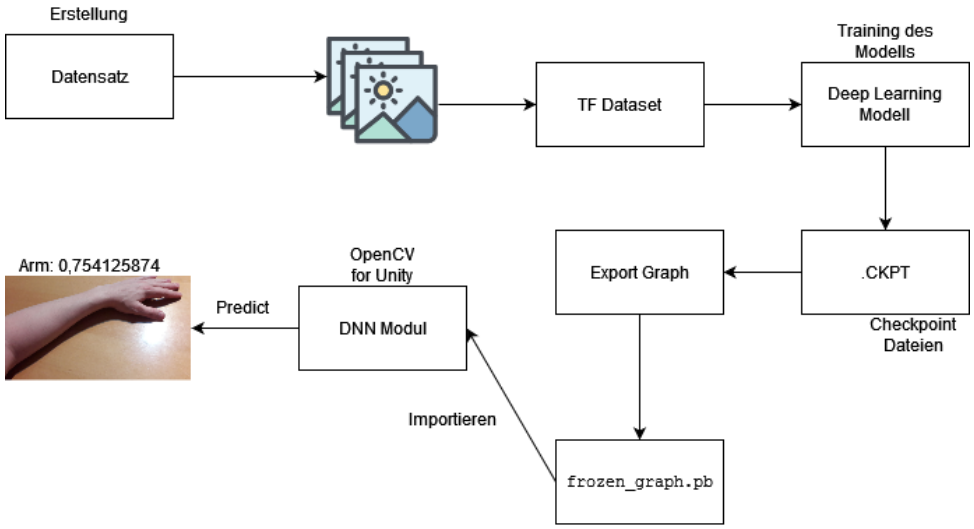


Abbildung 5.11: Übersicht Entwicklung des Wundlokalisators

5.2.1 Datensatz

Der Datensatz umfasst insgesamt:

- 16 Bilder für den Arm
- 28 Bilder für den Kopf
- 16 Bilder für die Hand

Dies sind Bilder des Verfassers dieser Arbeit und von Freiwilligen, die im Rahmen dieser Arbeit Bilder zur Verfügung gestellt haben. Der Datensatz ist in die entsprechenden Klassen aufgeteilt. imgaug wird benutzt, um den Datensatz mit zusätzlichen Bildern zu ergänzen. Es werden Crop und Flip verwendet.

Für die Erstellung des TensorFlow-Datensatzes muss der Datensatz in einer speziellen Ordnerstruktur vorliegen. Der Datensatz ist in einen Trainings- und einen Validations-Satz aufgesplittet. Die Ordner werden jeweils *training* und *validation* benannt. Innerhalb der Ordner werden drei weitere Ordner angelegt. Der Ordnername entspricht der Klasse, die in Tabelle 4.1 definiert ist. In den Ordner befinden sich die Bilder der jeweiligen Klasse. Der Datensatz für das Training wird mit imgaug augmentiert. Der Quellcode befindet sich in Listing G.2. Es werden die Techniken Crop und Flip verwendet. Crop wird auf die Hälfte aller Bilder angewendet. Das Skript lädt zuerst alle Bilder in ein Array. OpenCV wird für das Laden und für das Speichern der Bilder benutzt. In einer For-Schleife wird die Methode *augImage* für jedes Bild aufgerufen. Die Methode gibt die zwei neu erzeugten Bilder zurück und speichert sie wieder in den Ordner.

5.2.2 Training

Wie bereits erwähnt, wird für die Implementierung des Modells ein Jupyter-Notebook verwendet. Die im Notebook verwendete Sprache ist Python. Zu Beginn werden alle benötigten Frameworks und Bibliotheken importiert und TensorFlow wird so konfiguriert, dass die GPU anstelle der CPU verwendet wird. Die MobileNet-Architektur wird angepasst, um die Genauigkeit zu steigern. Das MobileNet wird direkt von TensorFlow mittels der Methode *tf.keras.applications.mobilenet.MobileNet* in das Notebook heruntergeladen. Es

handelt sich bei der Vorhersage um eine Mehrfachklassifizierung, also um die Vorhersage von mehr als zwei unterschiedlichen Klassen. Daher wird als Aktivierungsfunktion die Softmax-Funktion verwendet.

Das Modell wird bereits vortrainiert heruntergeladen. Der Parameter *weights* wird auf imagenet gesetzt. Das hat den Vorteil, dass wiederkehrende Muster während der Feature Extraction bereits anhand eines großen Datensatzes erlernt wurden. Dies steigert die Genauigkeit.

Listing 5.12: Anpassungen des MobileNets

```
x = mobile.layers[-6].output
x = Dense(1024,activation='relu')(x)
x = Dense(1024,activation='relu')(x)
x = Dense(512,activation='relu')(x)
output = Dense(3,activation='softmax')(x)
model = Model(inputs=mobile.input, outputs=output)
```

Die Layers des Modells werden wie in Listing 5.12 modifiziert. Zuerst werden die letzten sechs Layers entfernt. Danach werden drei neue Dense-Layer hinzugefügt. Die Aktivierungsfunktion der Layers ist ReLu. Der letzte Layer ist ebenfalls ein Dense-Layer mit drei Neuronen, der die Klassen darstellt. Als nächster Schritt werden die TensorFlow-Datensätze für das Training und für die Validation erstellt. Für die Erstellung wird die Methode *tf.keras.utils.image_dataset_from_directory* verwendet. Neben der Angabe des Dateipfades wird auch die Größe des Bildes angegeben, in die jedes Bild umgewandelt wird. Im Falle von MobileNet wird eine Breite und eine Höhe von je 224 Pixeln benutzt. Der Parameter *Label-Mode* ist auf *categorical* gesetzt, damit die Labels als Vektor dargestellt werden. Die Labels bzw. Klassen werden anhand des Ordnernamens automatisch festgelegt.

Als Nächstes wird das Modell kompiliert und es wird mit dem Training begonnen. Der Optimizer ist Root Mean Square Propagation (RMSProp) und verwendet eine Lernrate von 0,0001. Als Loss-Funktion wird *categorical_crossentropy* verwendet. Die Loss-Funktion und der Label-Mode des TensorFlow Datensatzes muss übereinstimmen. Während des Trainings verwendet TensorFlow die Genauigkeit um die Leistung des Modells zu bewerten und darzustellen. Der Wert befindet sich im Intervall zwischen null und eins, wobei eins 100% entspricht. Mit dem Aufruf *model.fit()* beginnt das Modell zu trainieren. Die Trainingsdauer wird auf zwölf Epochen festgelegt. Als Metrik wird *accuracy* während des Trainings und der Evaluierung verwendet.

Listing 5.13: model.compile() und model.fit()

```
model.compile(optimizer=tf.keras.optimizers.RMSprop(learning_rate=0.0001)
    , loss='categorical_crossentropy', metrics=['accuracy'])

history = model.fit(x=train_set, validation_data=vaild_set, epochs=12)
```

5.2.3 Auswertung

Die Ergebnisse des Trainings werden von der Methode *fit* als History-Objekt zurückgegeben. Dieses Objekt beinhaltet die Werte der Metrik nach jeder Epoche. Die Werte können in einem Diagramm darstellen. Abbildung 5.12 zeigt, dass die Kurve der Validation-Loss ihren höchsten Ausschlag bei Epoche 2 besitzt und sich immer weiter der Trainings-Loss annähert. Ab der elften Epoche beginnt die Validation-Loss anzusteigen, was darauf hindeutet, dass ab dieser Epoche eine Überanpassung stattfindet.

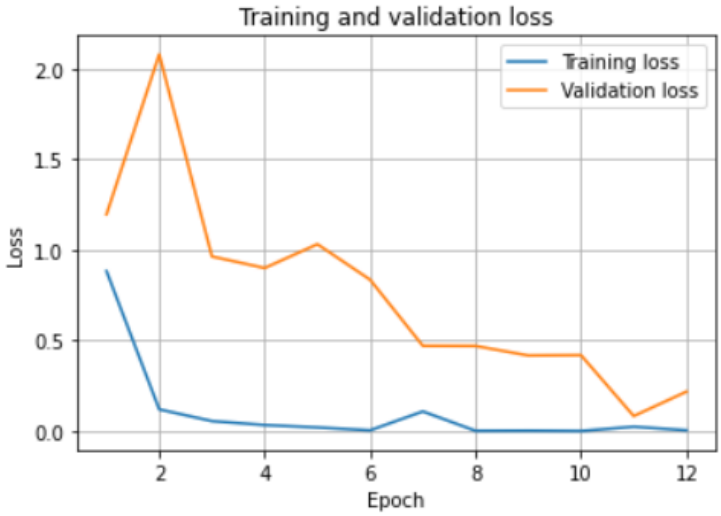


Abbildung 5.12: Verlustfunktionen des Wundlokalisators

In Abbildung 5.13 ist zu sehen, dass sich die Validation-Accuracy ab Epoche 11 sich nicht mehr signifikant verändert. Die Training-Accuracy steigt ab der zweiten Epoche auf 1 und verändert sich im Prinzip nicht mehr. Es ist zu beachten, dass das Modell mit einem kleinen Datensatz trainiert worden ist.

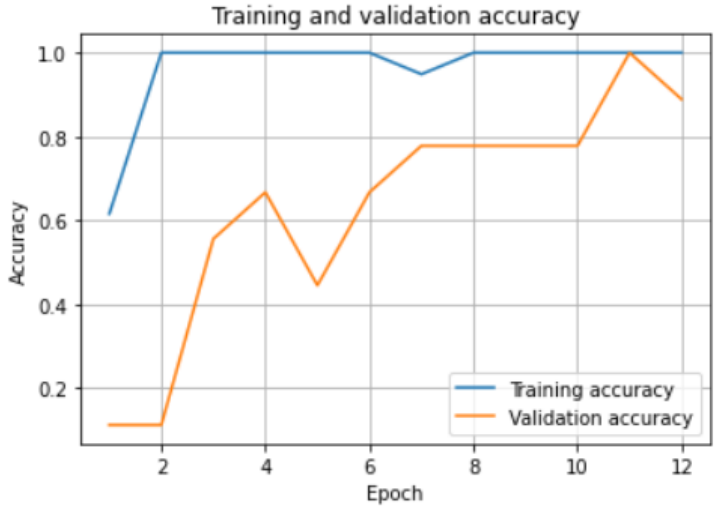


Abbildung 5.13: Genauigkeit

5.2.4 Export

Der letzte Schritt stellt den Export des Modells dar. Zuerst wird das Modell im TensorFlow SavedModel-Format gespeichert. Das SavedModel-Format enthält alle Gewichte, die Architektur sowie die Konfiguration. Das Modell muss in einen Frozen-Graph umgewandelt werden. Dazu wird zuerst die Signatur geladen. Die Signatur definiert den Input und den Output für die Berechnungen. Sie wird benötigt, um die *concrete function* zu erstellen. Bei der Erstellung der *concrete function* muss die Eingabeform angegeben werden. Die Funktion enthält alle für den Export notwendigen Berechnungen. Danach wird der Frozen-Graph umgewandelt und in ein Verzeichnis gespeichert.

5.2.5 Verwendung Wundlokalisator

Das Modell wird in der Klasse *Woundlocaliser* verwendet. Es wird OpenCV for Unity benutzt, um die Vorhersagen zu tätigen. Die Initialisierung erfolgt in der Start-Methode. Zuerst wird die Höhe und die Breite des Mats angegeben, die das Modell erwartet. Danach wird der Pfad angegeben. Für Android ist es notwendig, dass die pb-Datei im Ordner *StreamingAssets* liegt. Der Inhalt wird auf das Speichersystem des Android-Smartphones abgelegt und ist nicht Bestandteil des APK. Anschließend wird die Methode *readNetFromTensorFlow* verwendet, da es sich um ein TensorFlow-Modell handelt.

Listing 5.14: Start-Methode der Klasse Woundlocaliser

```
void Start()
{
    heightOfResize = 224;
    widthOfResize = 224;
    pathToModel = Application.dataPath + "/TensorFlowModels/Body/
        frozen_graph_body.pb";

    if (Application.platform == RuntimePlatform.Android)
    {
        pathToModel = Utils.getFilePath("frozen_graph_body.pb");
    }
    model = Dnn.readNetFromTensorFlow(pathToModel);
}
```

Die Vorhersagen werden durch den Aufruf der Methode *Predict* realisiert. Der Methode wird ein Wundbild als Mat übergeben. Danach wird in der Methode ein neues Mat deklariert, das für die Erstellung des Blob verwendet wird. Das Array *results* ist ein float-Array der Größe 3. Es speichert die Confidence-Werte für das Körperteil, an dem sich die Wunde befindet. Sie werden nicht extra im Array gespeichert. Die Klassen werden über die Indizes des Arrays bestimmt und diese entsprechen Tabelle 4.1. Das heißt, dass Index 1 dem Kopf entspricht. In der If-Anweisung wird geprüft, ob das übergeben Mat Null ist. Der Fehler wird dann auf der Konsole in Unity oder im Log in Android ausgegeben. Anschließend wird das Blob erzeugt, das Mat wird als Eingabe gesetzt und die Körperteile vorhergesagt. Ein Reshape ist an dieser Stelle nicht notwendig. Daher kann direkt die Methode *get* verwendet werden und das Ergebnis kann in das Array geschrieben werden.

Listing 5.15: Predict-Methode der Klasse Woundlocaliser

```
public float[] Predict(Mat frame)
{
        Mat resizeimage;
        float[] results = new float[3];

        if (frame.empty())
        {
            Debug.LogError("Frame is not loaded");
        }

        resizeimage = Dnn.blobFromImage(frame, 1.0, new Size(
            widthOfResize, heightOfResize));
        model.setInput(resizeimage);
        Mat output = model.forward();
        output.get(0, 0, results);

        return results;
}
```

5.3 Wundklassifikator

Der Wundklassifikator ist wie der Wundlokalisator ein Modell aus dem Deep Learning. Für die Implementierung des MobileNets mit SSD wird die TensorFlow Object Detection API (TFOD) API verwendet.

5.3.1 Datensatz

Für die Erstellung des Datensatzes werden unter anderem Google und spezielle Portale, wie pexels und Unsplash, benutzt. Alle verwendeten Bilder fallen unter die ‚Creative Commons Zero (CC0)'-Lizenz. Der Datensatz umfasst:

- 18 Bilder für die kleinen Schnittwunden
- 22 Bilder für die tiefen Schnittwunden
- 10 Bilder für Verbrennungen ersten Grades
- 7 Bilder für Verbrennungen zweiten Grades

Die Bilder werden mit LabelImg gelabelt und der Datensatz wird in Trainings- und Testdaten aufgeteilt. Es sind 90% Trainingsdaten und 10% Testdaten. Die XML-Dateien müssen noch in CSV-Dateien umgewandelt werden und die TensorFlow-Records-Dateien müssen wie in Abschnitt 2.6.2 erstellt werden. Zum Schluss wird eine Label-Map-Datei erstellt. Diese wird nachfolgend in der pipeline.config benutzt und ist eine pbtxt-Datei, die die Klassen nach Tabelle 4.2 enthält.

5.3.2 Training

Bevor mit dem Training begonnen werden kann, ist es notwendig, die Umgebung wie in Abschnitt 2.6 beschrieben zu konfigurieren. Im Ordner *workspace* wird ein neuer Ordner erstellt, der die projektspezifischen Dateien enthält. Im Ordner *wound_detector* werden die annotierten Bilder und die Records-Dateien abgelegt.

Als Modell wird MobileNet Version 3 mit SSD benutzt. Das Modell wird vom Model Zoo heruntergeladen und im Workspace abgelegt. Als Nächstes muss die pipeline.config editiert werden. In der Konfigurationsdatei wird das Attribut *num_classes* auf vier gesetzt, da vier Klassen vorhergesagt werden müssen. Die Pfade zu den TensorFlow-Records-Dateien werden bei *train_input_reader* und bei *eval_input_reader* gesetzt. Der IoU-Wert wird auf 0,58 geändert. Das verbessert die Genauigkeit der Bounding-Boxen. Die *data_augmentation_options* für die Augmentierung werden ebenfalls verwendet. Die Bilder werden horizontal und vertikal gespiegelt sowie zugeschnitten.

Damit ist die Konfiguration der pipeline.config abgeschlossen. Um das Training zu starten, wird die Datei model_main.py aus dem Ordner *models* in den Workspace kopiert. Der Kommandozeilenbefehl zum Starten ist in einem Windows Batch file (bat) Skript geschrieben, damit nicht jedes Mal die Parameter neu eingegeben werden müssen.

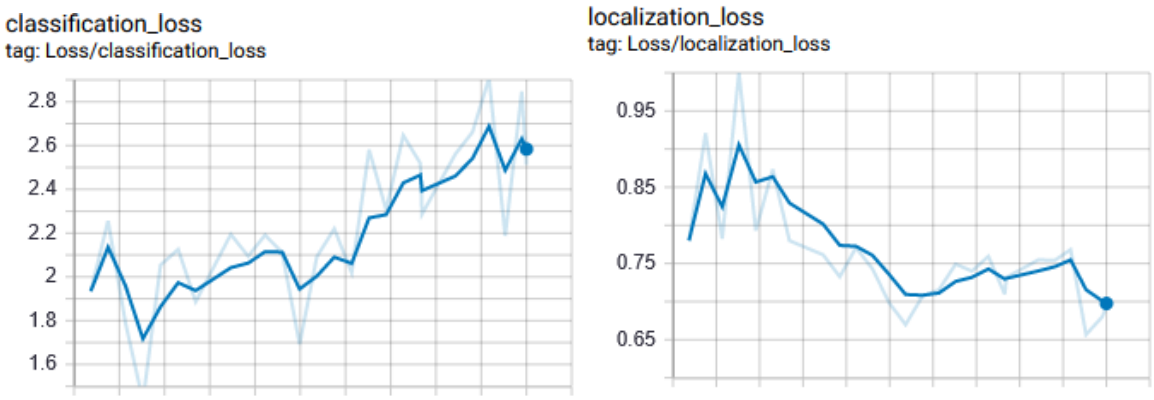
Listing 5.16: train.bat

```
python model_main.py --alsologtostderr --model_dir="F:\Net"
    --pipeline_config_path=pre-trained-models/
        ssd_mobilenet_v3_large_coco_2020_01_14/pipeline.config
    --num_train_steps=10000
```

Zuerst wird festgelegt, dass Fehler ebenfalls auf die Standardausgabe geleitet werden. Danach wird das Verzeichnis angegeben, in das die Checkpoint-Dateien gespeichert werden und der Pfad zur pipeline.config angegeben. Die Trainingsschritte sind auf 10 000 gesetzt. Nach dem Starten kann mit TensorFlow der Trainingsverlauf in Echtzeit überwacht werden.

5.3.3 Auswertung

Beim Training des Modells werden zwei Verlustfunktionen benutzt. Die eine Verlustfunktion betrifft die Klassifizierung, während die andere für die Lokalisierung, also für die Bounding-Boxen, verwendet wird. Abbildung 5.14 zeigt beide Verlustfunktionen. Die Verlustfunktion der Lokalisierung zeigt einen typischen Verlauf. Während des Trainings wird diese immer weiter ab. Die Verlustfunktion der Klassifizierung zeigt einen atypischen Verlauf. Die Verlustfunktion nimmt während des Trainings zu. Das zeigt, dass das Modell an dieser Stelle nicht gut funktioniert. Gründe können zum einen sein, dass die Klassen nicht gut ausbalanciert sind. Für die Klasse der tiefen Schnittverletzungen sind 18 Bilder vorhanden, während für Verbrennungen zweiten Grades nur sechs Bilder vorliegen. Zum anderen sind insgesamt sehr wenige Bilder im Datensatz vorhanden. Bei einer Aufteilung von 90 zu 10 des Datensatzes bleiben nur fünf Bilder für das Training bei Verbrennungen zweiten Grades.



(a) Verlustfunktion der Klassifizierung

(b) Verlustfunktion der Lokalisierung

Abbildung 5.14: Verlustfunktionen

Das Modell produziert ein Output der Form $1 \times 1 \times (N \times 7)$ wobei N der Anzahl der gefundenen Bounding-Boxen entspricht. Die einzelnen Einträge entsprechen einem Array der Größe 7. Dies hat die Form:

[image_id, class_id, confidence, (x_min, y_min), (x_max, y_max)]

Der erste Eintrag stellt die ID des Bildes dar. Der zweite Eintrag gibt die Klasse gemäß Tabelle 4.2 an. Danach kommt der Confidence-Wert. Zum Schluss werden die Koordinaten von zwei Punkten angegeben, mithilfe derer die Box generiert werden kann.

5.3.4 Export

Als letzter Schritt muss das Modell exportiert werden. Der Konsolenbefehl wird zwecks Wiederverwendung in ein bat-Skript geschrieben. Der Befehl ist in Listing 5.17 dargestellt.

Listing 5.17: train.bat

```
python export_inference_graph.py --input_type image_tensor
    --pipeline_config_path pre-trained-models/
        ssd_mobilenet_v3_large_coco_2020_01_14/pipeline.config
    --trained_checkpoint_prefix F:\Net\model.ckpt-10000
    --output_directory trained-inference-graphs/output_inference_graph_v1
        .pb
```

Die Nummer der letzten Checkpoint-Datei im Ordner *Net* und die im Konsolenbefehl müssen gleich sein. Der Frozen-Graph wird danach in das angegebene Verzeichnis exportiert.

5.3.5 Verwendung Wundklassifikator

Der Wundklassifikator ist in der Klasse *WoundDetectorService* realisiert. Der gesamte Quellcode der Klasse ist in Anlage H in Listing H.2 zu finden. Im Klassenkopf werden alle notwendigen Objekte deklariert. Die Labels werden über den Inspektor festgelegt, damit das Modell zukünftig einfacher um weitere Klassen erweitert werden kann. Breite, Höhe sowie die Dateinamen für die pb- und für die pbtxt-Dateien werden über den Inspektor festgelegt. Die Initialisierung erfolgt in der Methode Awake. Auf Android-Plattformen wird die Methode *Utils.getFilePath* benutzt, um die Dateien zu laden. Wie beim Wundlokalisator wird die Methode *readNetFromTensorFlow* benutzt, um das TensorFlow-Modell zu laden. Die Methoden *detectWounds* und *drawBox* werden in Abschnitt 5.1.4 beschrieben.

Für das Empfehlungssystem wird die Methode *classifiesWounds* benötigt. Die Methode funktioniert im Grunde wie die Methode *detectWounds*. Der Unterschied ist dabei, dass keine Bounding-Boxen in das Mat gezeichnet werden und nur die Klasse mit dem höchsten Confidence-Wert unter allen erkannten Boxen zurückgegeben wird. Im Kopfbereich der Methode werden zunächst das Mat deklariert. Das Array *predictions* wird als Ergebnis später wieder zurückgeben. Im Array werden die Klasse und der Confidence-Wert gespeichert. Das Array *result* wird benutzt, um die einzelnen Vorhersagen von dem Mat *output* zuladen. Anschließend wird die Klassifizierung vorgenommen. Danach muss ein Reshape durchgeführt werden. In der Variable *sizeMaxBoxes* wird die maximale Anzahl an Bounding-Boxen definiert, die das Modell liefert, angegeben. In der For-Schleife wird über das gesamte Mat iteriert, um die Bounding Box mit dem höchsten Confidence-Wert zu finden. Zum Schluss werden die Werte dem Array *predictions* zugewiesen.

Listing 5.18: Predict-Methode der Klasse Woundlocaliser

```
public float[] classifiesWounds(Mat frame)
{
    Mat resizeimage;
    float[] predictions = new float[2];
    float[] result = new float[7] { 0, 0, 0, 0, 0, 0, 0 };

    resizeimage = Dnn.blobFromImage(frame, 1.0, new Size(widthOfResize,
        heightOfResize));
    model.setInput(resizeimage);
    Mat output = model.forward();
    int sizeMaxBoxes = output.size(2);
    output = output.reshape(1, output.size(2));

    for (int i = 0; i < sizeMaxBoxes; i++)
    {
        float[] temp = new float[7];
        output.get(i, 0, temp);
```

```
        //Compare the probabilities
        if (temp[2] > result[2])
        {
                result = temp;
        }
    }
    //Class
    predictions[0] = result[1];

    //Probability
    predictions[1] = result[2];

    return predictions;
}
```

5.4 Empfehlungssystem

Das Empfehlungssystem befindet sich in der Klasse *Recommander*. Es wird in der Szene *Assistance* verwendet. Der Quellcode der Klasse befindet sich in Listing H.2. Ausgangspunkt ist die Klasse *AssistanceManager*. In der Start-Methode wird dem Empfehlungssystem ein Wundbild übergeben. Das Empfehlungssystem gibt ein Objekt vom Typ *Recommandation* zurück und enthält die folgenden Attribute:

Handlungsempfehlung Die Handlungsempfehlungen in Form eines Objektes des Typs *InstructionForAction*

woundlocation Integer, der das Körperteil gemäß 4.1 angibt.

woundclass Integer, der den Wundtyp gemäß 4.2 angibt.

confidenceWoundlocation Confidence-Wert des Körperteils.

confidenceWoundclassification Confidence-Wert des Wundtyps.

Die Start-Methode des AssistanceManagers lädt, wie im Abschnitt 5.1.6 beschrieben, zuerst das Foto. Nach dem Laden wird die Textur der Methode *AnalyseImageByTexture2D* der Klasse *Recommander* übergeben. Die Methode ist in Listing 5.19 zu finden. Zuerst werden einige Methoden deklariert und ein neues Mat mit der Höhe und der Breite der Textur wird generiert. Dies wird über die entsprechenden Attribute abgefragt. Ein neues Objekt von Typ *Recommandation* wird von ModelObjectBuilder erstellt. Danach erfolgt die Umwandlung des Mats in die Textur. Die Methode *Predict* gibt ein Float-Array mit den Confidence-Werten für die Körperteile aus. Anschließend wird die Methode *getIndexOfArrayByMaxValue* aufgerufen, damit die Klasse bestimmt werden kann. Die Methode nimmt das Array entgegen und gibt den Index des Arrays zurück, das den höchsten Confidence-Wert besitzt.

Als nächster Schritt wird die Klassifizierung der Wunde durchgeführt. Dazu wird die Methode *classifiesWounds* aufgerufen und als Parameter wird das Bild übergeben. In dem zurückgegebenen Array steht die Klasse an der ersten Stelle. Da der Integer-Wert in einem Float-Array steht, muss eine explizite Typumwandlung gemacht werden. Die bisher bestimmten Werte werden dem Objekt *recommandation* zugewiesen. Bevor das Objekt zurückgeliefert wird, wird die Methode *LoadInstructionForAction* aufgerufen und das Körperteil, an dem sich die Wunde befindet, sowie der Wundtyp werden als Parameter übergeben.

Listing 5.19: Predict-Methode der Klasse Woundlocaliser

```
public Recommandation AnalyseImageByTexture2D(Texture2D frame)
{
    int predictedWoundType;
    Mat img;
    float[] predictionsBodyParts;
    int predictedBodyPart;
    Recommandation recommandation;
    float[] predictionsWounds;

    img = new Mat(frame.height, frame.width, CvType.CV_8UC3);
    recommandation = modelObjectBuilder.GetRecommandation();

    Utils.texture2DToMat(frame, img);

    predictionsBodyParts = woundlocator.Predict(img);
    predictedBodyPart = getIndexOfArrayByMaxValue(predictionsBodyParts);

    predictionsWounds = woundDetectorService.classifiesWounds(img);
    predictedWoundType = (int)predictionsWounds[0];

    recommandation.woundlocation = predictedBodyPart;
    recommandation.confidenceWoundlocation = predictionsBodyParts[
        predictedBodyPart];
    recommandation.woundclass = predictedWoundType;
    recommandation.confidenceWoundclassification = predictionsWounds[1];
    recommandation.instructionForAction = LoadInstructionForAction(
        predictedBodyPart, predictedWoundType);

    return recommandation;
    }
```

Die Methode *LoadInstructionForAction* implementiert den Entscheidungsbaum aus Abbildung 4.8. Listing H.2 beinhaltet den Quellcode. Zu Beginn der Methode wird von ModelObjectBuilder ein Objekt vom Typ *Recommandation* erzeugt. Danach folgen zwei If-Anweisungen mit jeweils einer Switch-Anweisung. Die erste If-Anweisung prüft, ob es sich bei der Stelle um den Arm oder um die Hand handelt. Die Cases werden in Abhängigkeit vom Wundtyp ausgeführt. Die Handlungsempfehlung wird vom ModelObjectBuilder erzeugt. In Tabelle 5.1 ist die Zuordnung zwischen den Klassen und der entsprechenden Handlungsempfehlung dargestellt.

Tabelle 5.1: Zuordnung Klasse zu Wundtyp und Körperteil

Klasse	Wunde	Körperteil
SmallWoundsArm	Kleine Schnittverletzung	Arm & Hand
DeepWoundsArm	Tiefe Schnittverletzung	Arm & Hand
FirstDegreeBurnsArm	Verbrennung 1. Grades	Arm & Hand
SecondDegreeBurnsArm	Verbrennung 2. Grades	Arm & Hand
SmallWoundsHead	Kleine Schnittverletzung	Kopf
DeepWoundsHead	Tiefe Schnittverletzung	Kopf
FirstDegreeBurnsHead	Verbrennung 1. Grades	Kopf
SecondDegreeBurnsHead	Verbrennung 2. Grades	Kopf

Jede Klasse für die Handlungsempfehlungen erbt von der Klasse *InstructionForAction*. Diese Klasse ist eine abstrakte Klasse. Die Methode *initializeInstructions* ist ebenfalls abstrakt und wird im Konstruktor aufgerufen. Der Hintergrund dafür ist, dass in der Methode *initializeInstructions* die konkreten Anweisungen erstellt werden. Die Klasse hat das Attribut *counter* und ein Array vom Typ *Instruction*. Der Counter kann als Pointer gesehen werden, der auf dem Index des Arrays steht. Der Counter wird inkrementiert und dekrementiert. Die Klasse besitzt drei Methoden. Die Methode *getFirstInstruction* gibt das erste Instruction-Objekt aus dem Array zurück. Die Methode *next* gibt bei Aufruf die nächste Handlungsanweisung im Array zurück. Innerhalb der Methode wird in einer If-Anweisung geprüft, ob der Counter nicht an der letzten Stelle steht, damit noch eine Handlungsempfehlung geladen werden kann. In der If-Anweisung wird zuerst ein Objekt vom Typ *Instruction* deklariert. Danach wird der Count inkrementiert. Anschließend wird mit dem Aufruf *instruction = arrayOfInstruction[counter];* die Handlungsempfehlung von Array geladen. Zuletzt wird die Handlungsempfehlung zurückgegeben. Die Methode *previous* gibt die vorherige Anweisung zurück. Auch sie besitzt eine If-Anweisung. Die If-Anweisung wird nur ausgeführt, wenn der Counter größer als null ist. Innerhalb der If-Anweisung wird zuerst ein Objekt vom Typ *Instruction* deklariert und der Counter dekrementiert. Nachdem das Empfehlungssystem mit dem Objekt *Recommandation* fertig ist, wird dieses in der Start-Methode der Klasse *AssistanceManager* weiterverarbeitet. Beginnend wird mit dem Aufruf *assistanceList.setInstructionForAction(recommandation .instructionForAction);*. Dieser weist das Objekt *instructionForAction* zu. Der Anwender bekommt in der Szene zum einen das klassifizierte Körperteil und zum anderen den erkannten Wundtyp angezeigt. Für beides wird der Confidence-Wert angegeben. Dies wird in zwei Switch-Anweisungen realisiert. Die erste Anweisung prüft die Wundstelle. In den Case-Anweisungen wird der String für das UI-Element erstellt und zugewiesen. Der Wert ist dabei im Attribut *confidenceWoundlocation* des recommandation-Objekts gespeichert. Die zweite Switch-Anweisung ist identisch aufgebaut, jedoch werden der Wundtyp und der entsprechende Confidence-Wert angezeigt.

Die Klasse *AssistanceList* gibt dem Anwender die Möglichkeit, zwischen den einzelnen Anweisungen in der Szene hin und her zu blättern. Der gesamte Quellcode befindet sich in Listing I.10. Die Methode *setInstructionForAction* wird von der Klasse *AssistanceManager* aufgerufen. Die zwei restlichen Methoden *next* und *back* werden für das Blättern benutzt. Innerhalb des try-Blocks (siehe Listing 5.20) prüft die If-Anweisung, ob im Array *symbole* ein Eintrag vorhanden ist. In dem Fall, dass das Array nicht Null ist, wird das GameObject *aidSymboleGameObject* aktiv gesetzt. Im Array befindet sich der Dateiname des Symbols, das geladen werden soll. Die statische Methode *Load* lädt das Bild, das sich im Ordner Icons befindet.

Listing 5.20: Try-Block der next-Methode

```
try
{
    instruction = instructionForAction.next();
    instructionAsString.text = instruction.text;
    if(instruction.symbole != null)
    {
        aidSymboleGameObject.SetActive(true);
        aidSymbole.texture = Resources.Load<Texture2D>("Icons/" +
            instruction.symbole[0]);
    }
}
```

5.5 Bereitstellung der App

Unity verfügt über entsprechende Funktionen, um die App auf dem Smartphone bereitzustellen. Damit die Bereitstellung seitens des Smartphones funktioniert und Unity eine Verbindung aufbauen kann, müssen zuvor im Android-Smartphone die Entwickleroptionen aktiviert werden. Die Option *USB debugging* muss ebenfalls aktiviert sein, wenn das DEbuggen ermöglicht werden soll. Während der Entwicklungs- und der Testphase können die Entwickleroptionen über die Build-Settings eingestellt werden. Damit ist es möglich, eine Analyse und eine Fehlerbehebung direkt auf dem Gerät auszuführen. Dazu sind die folgenden Optionen notwendig.

Development Build Damit die übrigen Optionen für die Entwicklung überhaupt funktionieren, muss ein Development-Build erstellt werden. Erst dann können die anderen Optionen aktiviert werden. Der Build enthält alle notwendigen Pakete dafür.

Script Debugging Um die Fehlerbehebung zu ermöglichen, kann diese Option aktiviert werden. Visual Studio ist danach in der Lage, die Skripte zu debuggen. Visual Studio kann sich über USB oder WLAN mit dem Smartphone verbinden.

Wait for managed Debugger Beim Start der App wird eine Meldung angezeigt und der Entwickler wird informiert, dass der Debugger verbunden werden kann. Das Laden der App wird fortgesetzt, wenn die Meldung bestätigt wurde. Damit besteht genügend Zeit, um eine Verbindung herzustellen.

Die Entwickleroptionen werden nur während der Entwicklung verwendet. Für eine Evaluierung können diese deaktiviert werden, damit die Größe verringert wird und die Performance der App gesteigert werden kann. Damit Unity das APK-Paket installieren kann, ist es notwendig, dass das Android-Smartphone über USB mit dem Computer verbunden ist. Das Debugging kann auch über ein USB-Kabel oder über eine WLAN-Verbindung durchgeführt werden. Während der Entwicklung kann ebenfalls die von Android bereitgestellte Android Debug Bridge (ADB) benutzt werden, um Konsolenausgaben, beispielsweise *Debug.Log* auszugeben. Dazu kann das Kommandozeilentool *adb* verwendet werden. Dieses befindet sich im Ordner *platform-tools* des Android-SDKs. Mit dem Aufruf *adb logcat -s Unity* wird die Konsole ausgegeben. Mit dem Parameter *-s Unity* werden nur Ausgaben von Unity angezeigt.

5.6 Zusammenfassung

In diesem Kapitel wurde die Realisierung der im Kapitel 3 erstellten Anforderungen durchgeführt. Der Kern der App beinhaltet die zwei TensorFlow-Modelle und das Empfehlungssystem. Unity dient dabei als Plattform. Für die Entwicklung des Wundlokalisators wurde die meiste Zeit für die Anpassung der Architektur verwendet. Bei der Entwicklung des Wundklassifikators mussten zuerst die richtigen Hyperparameter gefunden werden. Die Entwicklung der App und die Integration der Modelle waren teilweise schwierig. OpenCV for Unity unterstützt nicht alle Formate, daher musste darauf geachtet werden, dass das Modell richtig exportiert wird. Ebenso schwierig war das spätere Testen auf dem Smartphone.

Die App wurde regelmäßig in Unity gestartet und die Funktionsweise wurde getestet. Nachdem die Grundfunktionen der App fertiggestellt wurden, wurde die App auf das Smartphone deployed. Nach einigen Tests stellte sich heraus, dass die App sich bei bestimmten Funktionen anders verhält als in Unity. Ein Beispiel dafür ist die Verwendung

der Kamera. Wie bereits beschrieben, wird das Kamerabild, das von Android bereitgestellt wird, abweichend von der Webcam am Computer dargestellt. Das Bild muss daher rotiert werden. Ein anderes Beispiel ist, dass bei einer *Texture2D* nicht zwingend das *TextureFormat* angegeben werden muss. In Unity funktioniert dies ohne Probleme, während auf dem Smartphone die geladene Textur immer grau ist und nicht richtig dargestellt wird. Daraus resultierte, dass der Quellcode immer wieder angepasst werden musste, was zusätzliche Entwicklungszeit benötigte.

6 Evaluation

In diesem Kapitel erfolgt die Evaluierung. Diese findet mittels einer Einzelfallstudie statt. Zuerst wird die Einzelfallstudie im Allgemeinen erörtert. Danach wird der Aufbau des Fragebogens erklärt. Abschließend werden die einzelnen Fragen ausgewertet.

6.1 Einzelfallstudie

Für die Evaluation wird eine Einzelfallstudie durchgeführt, um qualitatives Feedback von Anwendern zu bekommen. Der Fragebogen ist nach der ISONORM 9241/110-S aufgebaut und befindet sich in Anlage J. Er beginnt mit dem allgemeinen Teil, bei dem das Geschlecht und das Alter erfragt werden. Danach wird gefragt, ob der Proband schon einmal eine Erste-Hilfe-Situation erlebt hat und wie offen er gegenüber digitalen Lösungen ist. Dann folgen Blöcke von Fragen für die App, über die Handlungsempfehlungen und über das Pflaster. Abschließend befinden sich zwei Freitextfelder im Bogen, in denen die Testperson ein zusätzliches Feedback geben kann. Die Fragenblöcke sind so aufgebaut, dass der Proband eine bestimmte Aussage bewerten soll. Die negativen Aussagen befinden sich auf der rechten Seite und die positiven auf der linken Seite. Die Bewertungsskala reicht dabei von sehr schlecht bis sehr gut. Die Werte sind zwischen 1 und 7. Damit die Einzelfallstudie etwas realistischer gestalten werden konnte, wurden verschiedene Wunden aus Silikon und Latex verwendet.

(a)

(b)

Abbildung 6.1: Wunden

6.2 Ablauf der Einzelfallstudie

Im ersten Schritt wird die Testperson abgeholt und das Vorgehen des Tests wird erläutert. Danach wird dem Probanden kurz die App gezeigt und es wird erklärt, welche Funktionen er nutzen kann. Anschließend wird der Testperson ein Rahmen gegeben. Damit bekommt die Testperson einen gewissen Kontext und kann sich leichter in das Szenario einfinden. Der Nutzer bekommt die Aufgabe, die Erste-Hilfe durchzuführen und einem Wundverlauf mit einem gemachten Foto zu erstellen.

6.3 Auswertung

Insgesamt haben 10 Probanden an der Einzelfallstudie teilgenommen. Davon sind vier männliche und fünf weibliche Testpersonen. Bei einem Fragebogen wurde das Geschlecht nicht angegeben. Sechs Probanden gaben an, schon einmal in einer Ersten-Hilfe-Situation gewesen zu sein. Die restlichen Probanden verneinten diese Frage. Als Nächstes bewerten die Probanden die Aussage, wie weit sie offen für digitalen Lösungen zur Unterstützung in der Ersten-Hilfe sind. 4 von 10 Probanden stimmten dieser Aussage voll und ganz zu, während 5 von 10 der Aussage zustimmen. Ein Proband beantwortete die Frage nicht.

1. Aussage: Die App ist sinnvoll.

- 2 von 10 Probanden antworteten mit 5.
- 7 von 10 Probanden antworteten mit 6.
- 1 von 10 Probanden antworteten mit 7.
- Das arithmetische Mittel beträgt 5,9.
- Der Median beträgt 6.
- Die Standardabweichung beträgt 0,57.

Abbildung 6.2 zeigt das Diagramm.

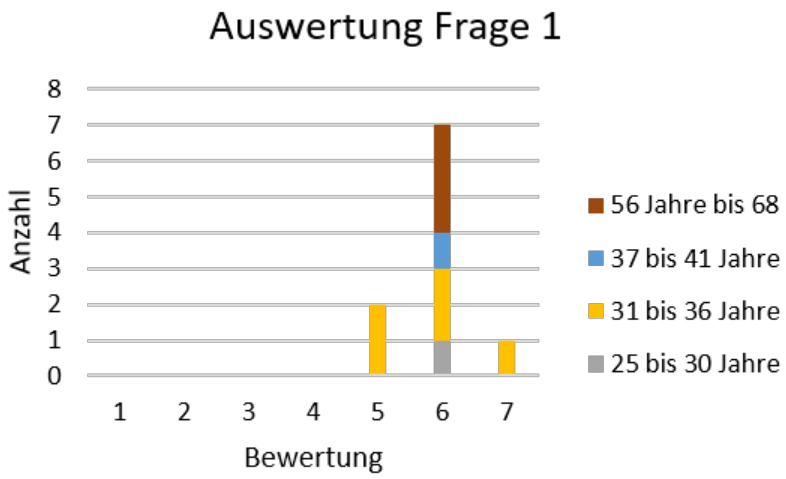


Abbildung 6.2: Auswertung von Frage 1

2. Frage: Die App werde ich benutzen.

- 1 von 10 Probanden antworteten mit 3.
- 1 von 10 Probanden antworteten mit 4.
- 3 von 10 Probanden antworteten mit 5.
- 5 von 10 Probanden antworteten mit 6.
- Das arithmetische Mittel beträgt 5,2.
- Der Median beträgt 5,5.
- Die Standardabweichung beträgt 1,03.

Abbildung 6.3 zeigt das Diagramm.

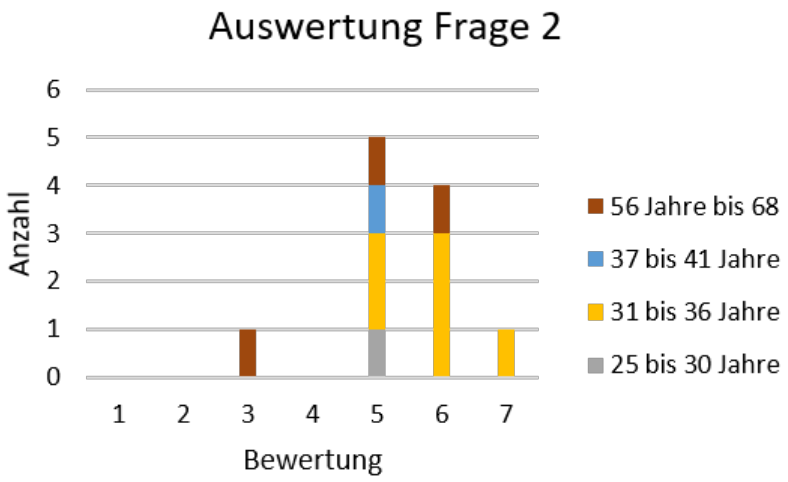


Abbildung 6.3: Auswertung von Frage 2

3. Frage: Die App ist leicht zu bedienen.

- 9 von 10 Probanden antworteten mit 6.
- 1 von 10 Probanden antworteten mit 7.
- Das arithmetische Mittel beträgt 6,1.
- Der Median beträgt 6.
- Die Standardabweichung beträgt 0,3.

Abbildung 6.4 zeigt das Diagramm.

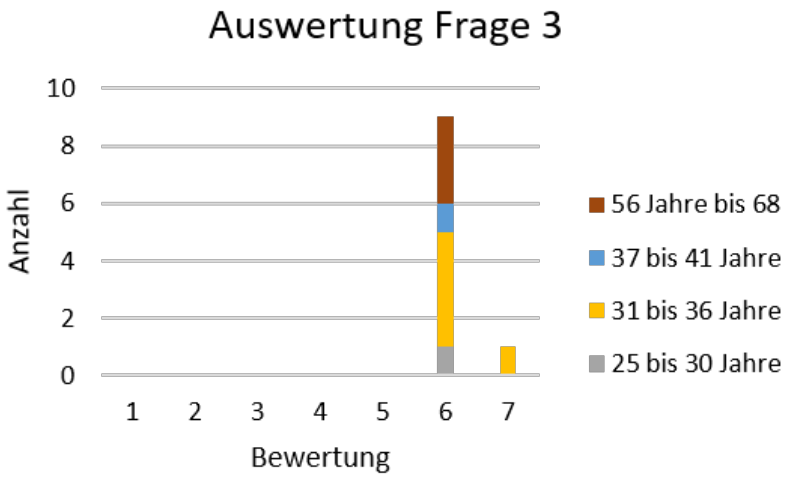


Abbildung 6.4: Auswertung von Frage 3

4. Frage: Die App bietet eine gute Performance.

- 1 von 10 Probanden antworteten mit 2.
- 1 von 10 Probanden antworteten mit 3.
- 5 von 10 Probanden antworteten mit 5.
- 2 von 10 Probanden antworteten mit 6.
- 2 von 10 Probanden antworteten mit 7.
- Das arithmetische Mittel beträgt 4,9.
- Der Median beträgt 5.
- Die Standardabweichung beträgt 1,4.

Abbildung 6.5 zeigt das Diagramm.

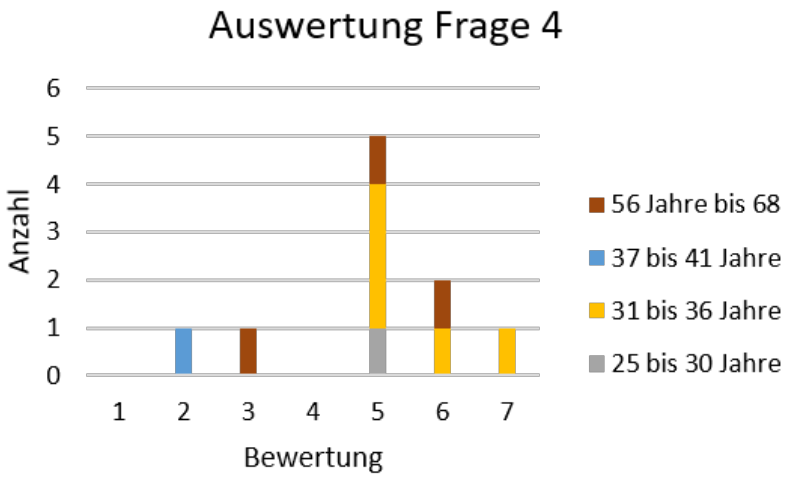


Abbildung 6.5: Auswertung von Frage 4

5. Frage: Die App ist leicht ohne fremde Hilfe oder Handbuch erlernbar.

- 1 von 10 Probanden antworteten mit 2.
- 2 von 10 Probanden antworteten mit 6.
- 7 von 10 Probanden antworteten mit 7.
- Das arithmetische Mittel beträgt 6,3.
- Der Median beträgt 7.
- Die Standardabweichung beträgt 1,6.

Abbildung 6.6 zeigt das Diagramm.

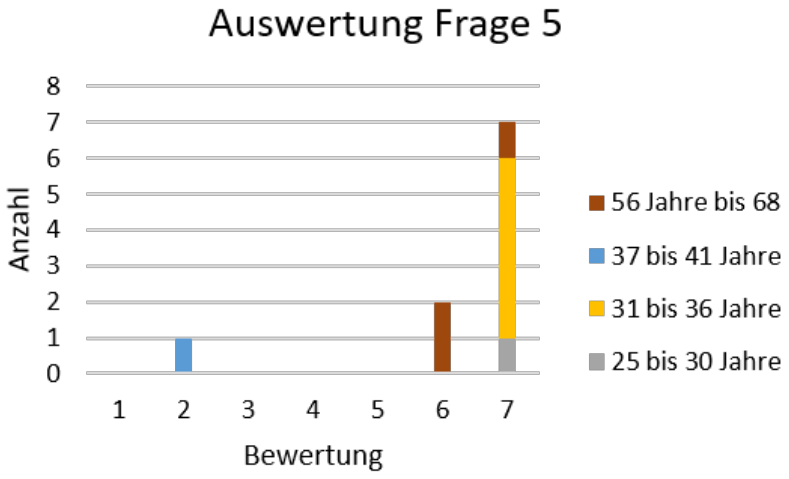


Abbildung 6.6: Auswertung von Frage 5

6. Frage: Die App hat eine übersichtliche Oberfläche.

- 3 von 10 Probanden antworteten mit 6.
- 7 von 10 Probanden antworteten mit 1.
- Das arithmetische Mittel beträgt 6,7.
- Der Median beträgt 7.
- Die Standardabweichung beträgt 0,5.

Abbildung 6.7 zeigt das Diagramm.

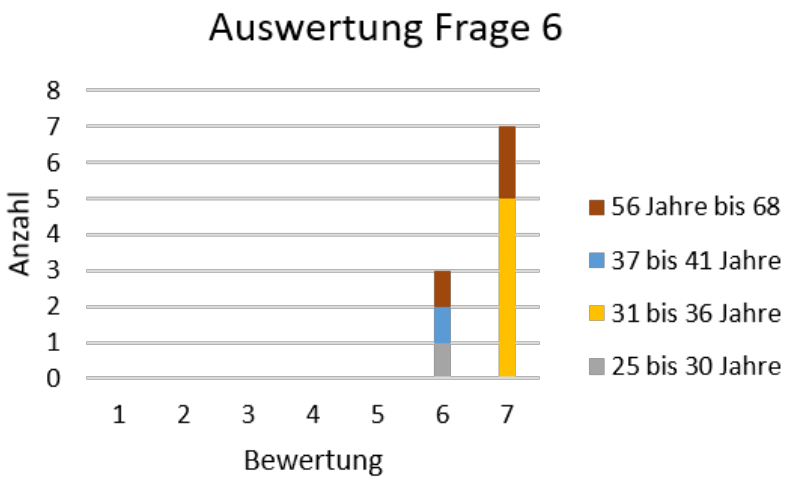


Abbildung 6.7: Auswertung von Frage 6

7. Frage: Die App hat mir gut in der Erste-Hilfe-Situation geholfen.

- 2 von 10 Probanden antworteten mit 4.
- 4 von 10 Probanden antworteten mit 5.
- 4 von 10 Probanden antworteten mit 6.
- Das arithmetische Mittel beträgt 5,2.
- Der Median beträgt 5.
- Die Standardabweichung beträgt 0,8.

Abbildung 6.8 zeigt das Diagramm.

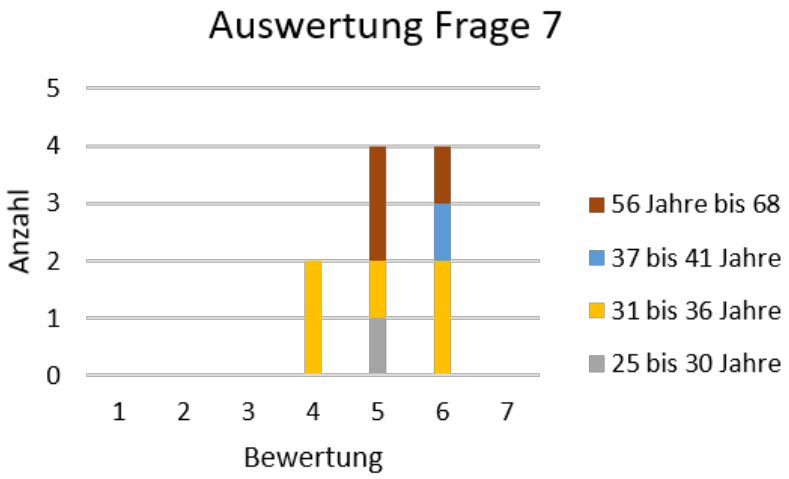


Abbildung 6.8: Auswertung von Frage 7

8. Frage: Die App hat es mir leicht gemacht, den Wundverlauf zu erstellen.

- 1 von 10 Probanden antworteten mit 4.
- 3 von 10 Probanden antworteten mit 6.
- 6 von 10 Probanden antworteten mit 7.
- Das arithmetische Mittel beträgt 6,4.
- Der Median beträgt 7.
- Die Standardabweichung beträgt 0,9.

Abbildung 6.9 zeigt das Diagramm.

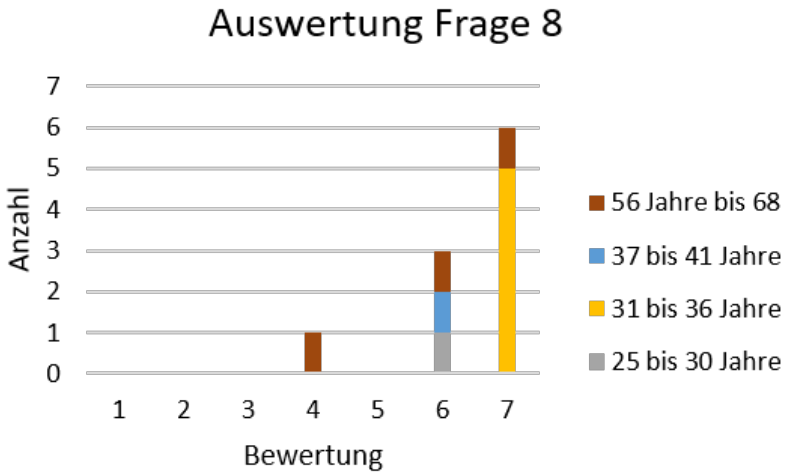


Abbildung 6.9: Auswertung von Frage 8

9. Frage: Die App hat das Körperteil richtig erkannt.

- 1 von 10 Probanden antworteten mit 3.
- 1 von 10 Probanden antworteten mit 5.
- 2 von 10 Probanden antworteten mit 6.
- 5 von 10 Probanden antworteten mit 7.
- Das arithmetische Mittel beträgt 6,2.
- Der Median beträgt 7.
- Die Standardabweichung beträgt 1,3.

Abbildung 6.10 zeigt das Diagramm.

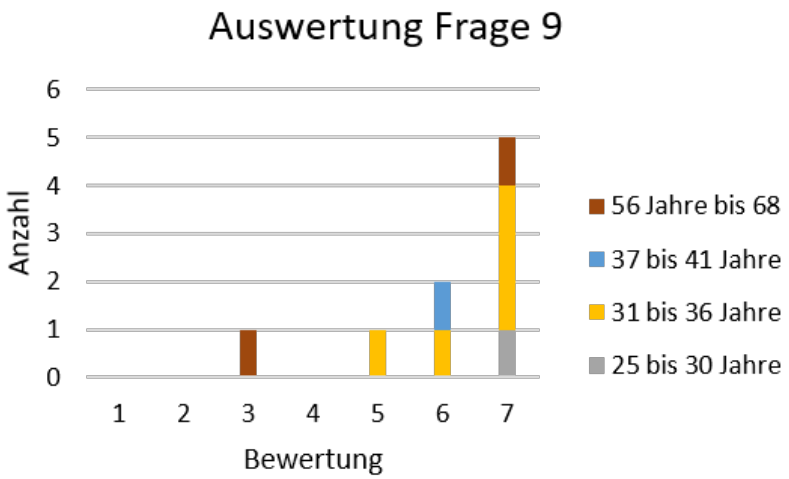


Abbildung 6.10: Auswertung von Frage 9

10. Frage: Die App hat die Wunde richtig zugeordnet.

- 1 von 10 Probanden antworteten mit 3.
- 5 von 10 Probanden antworteten mit 6.
- 7 von 10 Probanden antworteten mit 4.
- Das arithmetische Mittel beträgt 6,1.
- Der Median beträgt 6.
- Die Standardabweichung beträgt 1,2.

Abbildung 6.11 zeigt das Diagramm.

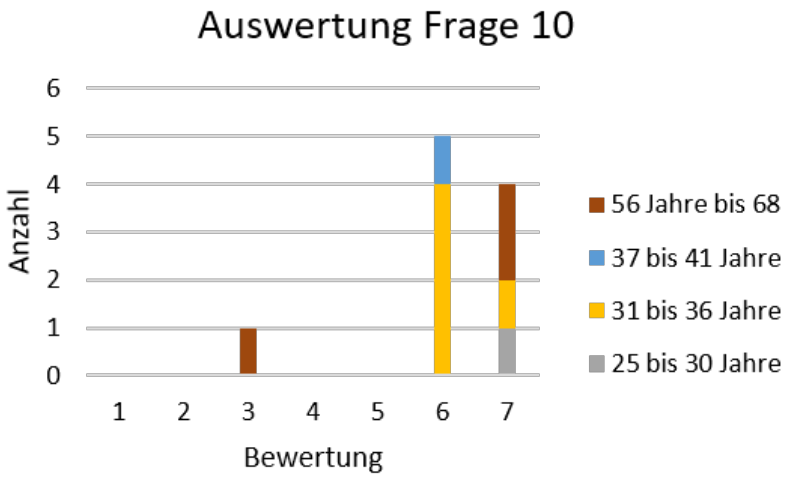


Abbildung 6.11: Auswertung von Frage 10

11. Frage: Die Handlungsempfehlungen sind klar verständlich.

- 2 von 10 Probanden antworteten mit 1.
- 6 von 10 Probanden antworteten mit 6.
- 2 von 10 Probanden antworteten mit 7.
- Das arithmetische Mittel beträgt 6,1.
- Der Median beträgt 6.
- Die Standardabweichung beträgt 1,2.

Abbildung 6.12 zeigt das Diagramm.

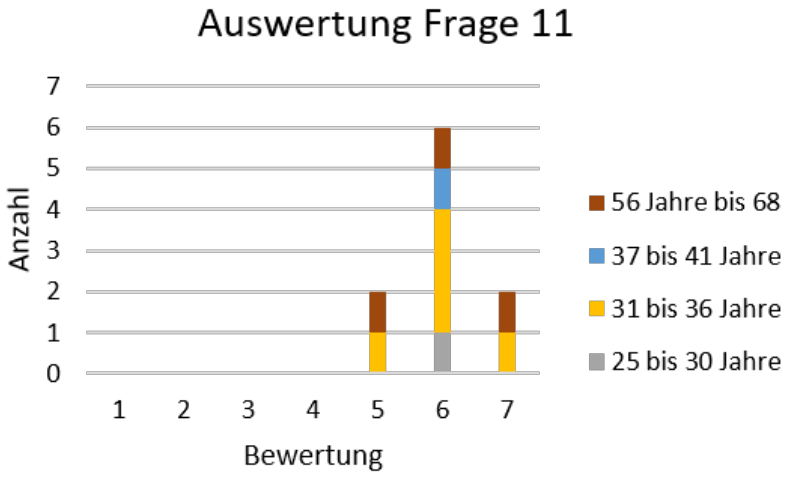


Abbildung 6.12: Auswertung von Frage 11

12. Frage: Die Handlungsempfehlungen haben mir geholfen.

- 1 von 10 Probanden antworteten mit 1.
- 1 von 10 Probanden antworteten mit 1.
- 1 von 10 Probanden antworteten mit 1.
- Das arithmetische Mittel beträgt 6.
- Der Median beträgt 6.
- Die Standardabweichung beträgt 0,7.

Abbildung 6.13 zeigt das Diagramm.

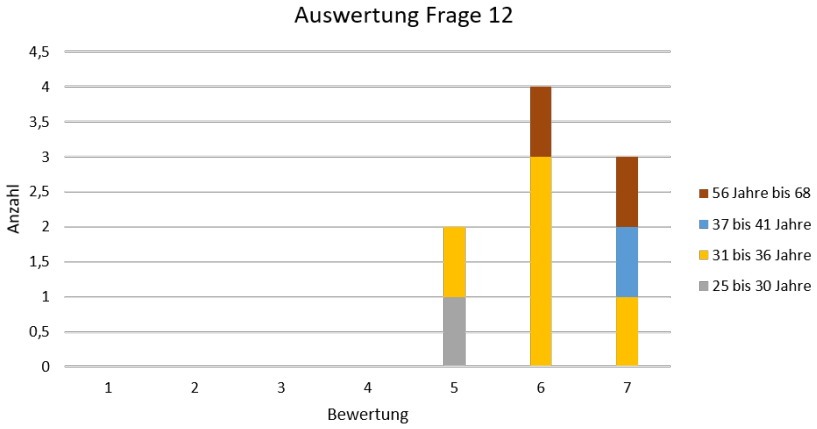


Abbildung 6.13: Auswertung von Frage 12

13. Frage: Die Handlungsempfehlungen enthalten genügend Informationen für die Behandlung.

- 2 von 10 Probanden antworteten mit 3.
- 5 von 10 Probanden antworteten mit 4.
- 1 von 10 Probanden antworteten mit 5.
- 2 von 10 Probanden antworteten mit 6.
- Das arithmetische Mittel beträgt 4,3.
- Der Median beträgt 4.
- Die Standardabweichung beträgt 1,0.

Abbildung 6.14 zeigt das Diagramm.

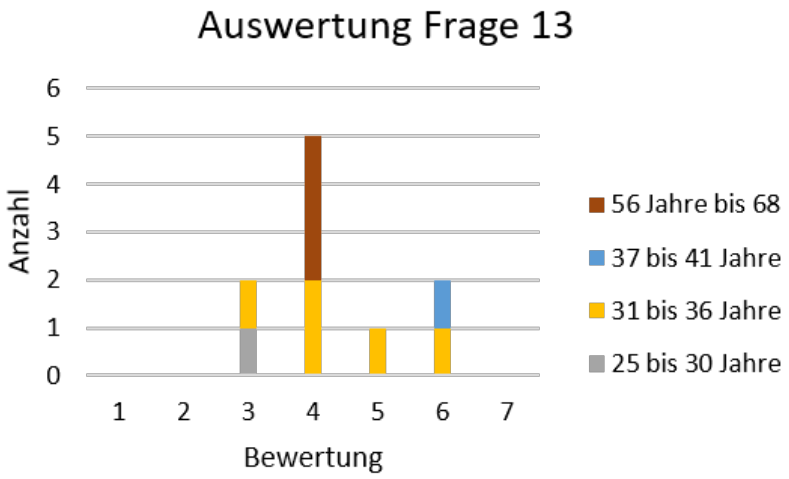


Abbildung 6.14: Auswertung von Frage 13

14. Frage: Das Pflaster ist sinnvoll.

- 1 von 10 Probanden antworteten mit 3.
- 5 von 10 Probanden antworteten mit 4.
- 3 von 10 Probanden antworteten mit 5.
- 1 von 10 Probanden antworteten mit 6.
- Das arithmetische Mittel beträgt 4,4.
- Der Median beträgt 4.
- Die Standardabweichung beträgt 0,8.

Abbildung 6.15 zeigt das Diagramm.

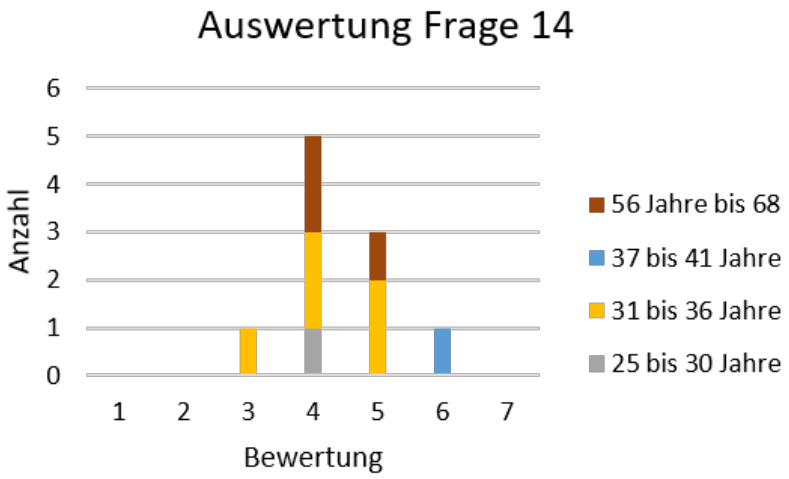


Abbildung 6.15: Auswertung von Frage 14

15. Frage: Das Pflaster ist leicht auf der Wunde anzubringen.

- 1 von 10 Probanden antworteten mit 1.
- 6 von 10 Probanden antworteten mit 4.
- 1 von 10 Probanden antworteten mit 5.
- 2 von 10 Probanden antworteten mit 7.
- Das arithmetische Mittel beträgt 4,4.
- Der Median beträgt 4.
- Die Standardabweichung beträgt 1,7.

Die Abbildung 6.16 zeigt das Diagramm.

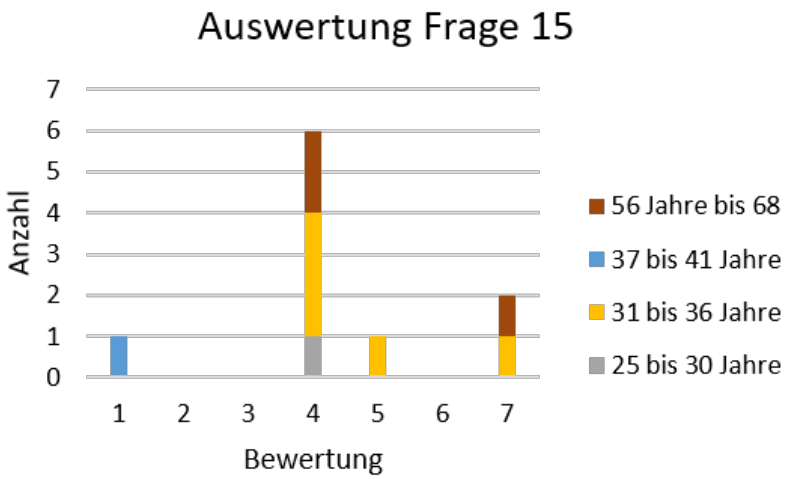


Abbildung 6.16: Auswertung von Frage 15

Der Freitext für Verbesserungen oder Erweiterungen wurden von einigen Probanden verwendet. Als Verbesserung wurde vorgeschlagen, ein Foto auch ohne Analyse machen zu können. Diese ist nicht notwendig, wenn die Wunde bereits versorgt ist, aber der Anwender ein weiteres Foto für den Wundverlauf machen will. Ein weiterer Vorschlag ist, dass die App die Möglichkeit bietet, direkt den Notruf anzurufen. Dies kann mit einer Handlungsanweisung verknüpft sein. Denkbar ist, dass sobald die Handlungsanweisung auf den Notruf verweist, der Anwender auf die ‚112' klickt und dadurch der Notruf ausgelöst wird.

Zusammenfassend kann gesagt werden, dass die App insgesamt positiv bewertet wurde. Den Probanden ist es leicht gefallen die App zu bedienen. Die Klassifizierung und Lokalisierung der Wunde wurde gut bewertet. Die Probanden hatten keine Probleme mit der Erstellung des Wundverlaufs. Die Frage 7 wurde im Schnitt mit 5,2 bewertet. Hier besteht noch Verbesserungspotential, da die App im Kern den Anwender genau solchen Situationen helfen soll. Die Fragen 13, 14 und 15 müssen kritisch betrachtet werden. Alle drei Fragen werden nur durchschnittlich bewertet. Frage 13 bezieht sich auf den Informationsgehalt der Handlungsanweisungen. Diese sind bewusst so gestaltet, dass sie kurze verständliche Anweisungen enthalten, damit der Anwender auch unter Stress die beschriebenen Handlungen ausführen kann. Die Anforderungen im Hinblick auf

die Handlungsanweisungen sind daher nur teilweise erfüllt. Ebenfalls kritisch muss die Frage 14 bewerten werden. Die Frage 14 soll die Sinnhaftigkeit des Pflasters beantworten. Die Frage wurde im Schnitt mit 4,4 bewertet. Daraus kann geschlussfolgert werden, dass das Pflaster neutral betrachtet wird, aber keinen wirklichen Mehrwert durch die Anbringung auf die Wunde erreicht wird. Die Probanden sollen mit der Frage 15 die Benutzerfreundlichkeit bezüglich des Pflasters bewerten. Die Bewertung ist auch hier nur durchschnittlich. Zu beachten ist, dass die Standardabweichung bei dieser Frage am größten ist. Zwei der Probanden haben die Fragen mit 7 bewertet und ein Proband hat die Frage mit 1 bewertet. Diese Probanden unterscheiden sich im Alter und Geschlecht. Die Bewertung kann ggf. auf unterschiedliche Fähigkeiten im Umgang mit einer Touch-Steuerung zurückgeführt werden.

7 Zusammenfassung und Ausblick

In diesem abschließenden Kapitel sollen die Ergebnisse der vorliegenden Masterarbeit zusammengefasst werden.Anschließend wird ein Ausblick gegeben, wie die App weiterentwickelt werden kann.

7.1 Zusammenfassung

Das Ziel der Arbeit war es, eine App zu entwickeln, die dem Anwender in einer besonderen medizinischen Situation hilft, diese zu bewältigen, und die bei der Versorgung der Wunde assistiert. Die App besteht im Wesentlichen aus den folgenden drei Bausteinen: Unity, einem Modell für die Wundlokalisierung basierend auf einem angepassten MobileNet sowie einem MobileNet mit SSD , das mithilfe der TensorFlow-Object-API erstellt wurde. Die für diese Arbeit benötigten Grundlagen wurden in Kapitel 2 vermittelt. In Kapitel 3 erfolgte die Analyse, dabei wurden unter anderem die Anforderungen festgelegt und der Stand der Technik wurde beschrieben. Ebenfalls wurden verschiedene Erste-Hilfe-Apps untersucht. Keine der untersuchten Apps verwendet KI zum Assistieren des Anwenders. Es wurden für die Arbeit zwei KI-Modelle entwickelt. Diese wurden neben der Unity-App in Kapitel 4 entworfen. Die Klassen wurden mit UML unter der Verwendung von zwei Entwurfsmustern modelliert. Die gesamte App wurde unter dem Aspekt der Modularisierung mit einer möglichst losen Koppelung der Komponenten entworfen, damit die App sich einfach erweitern lässt.Um die Anwender situationsabhängig unterstützen zu können, kommen zwei KI-Modelle zum Einsatz.

Im Kapitel 5 wurde die App entwickelt. Zuerst wurde ein Datensatz für die Erkennung der Körperteile und ein Datensatz für die Wundklassifikation erstellt. Das Empfehlungssystem ist die Schnittstelle zwischen Unity, dem Wundlokalisator, dem Wundklassifikator und dem Anwender. Es erstellt die Handlungsanweisungen. Durch dem Wundklassifikator kann gezielt eine bestimmte Wunde behandelt werden, da die Handlungsanweisung auf diesen Typ zugeschnitten ist. Der Anwender muss somit auch nicht die Merkmale zur Unterscheidung kennen und er erhält eine optimale Unterstützung bei der Versorgung der Wunde. Eine weitere Unterstützung ist die Möglichkeit, bestimmte Symbole, wie das Pflaster, bei jeder Handlungsanweisung anzuzeigen. Das Pflaster kann in der App über die Wunde angebracht werden, was dem Anwender hilft, ein Pflaster auf die reale Wunde anzubringen. Ein weiterer Aspekt, der dem Anwender hilft, ist der Wundverlauf. Er hat damit die Möglichkeit, die Fotos von Wunden zu speichern und in einem Verlauf zusammenzufassen. Er dokumentiert dadurch die Heilung und kann Abweichungen feststellen.

Nach dem Abschluss der Entwicklung wurde die App im Rahmen einer Einzelfallstudie bewertet. Die App wurde grundsätzlich gut bewertet. Kritisch muss die Verwendung des Pflasters stellvertretend für ein Symbol hinterfragt werden. Das Pflaster ist dazu gedacht, dem Anwender bei der Anbringung behilflich zu sein. Einen wirklichen Mehrwert für den Anwender wurde jedoch nicht erreicht. Die Handlungsanweisungen können bezüglich der Informationen für die Behandlung verbessert werden.

Ein beschränkender Faktor stellt der Datensatz für die Wundklassifizierung dar. Wunden, die nicht unbedingt medizinisch versorgt werden müssen oder die ohne weitere Komplikationen heilen, werden in der Regel zwar über die Wunddokumentation erfasst,

aber nicht fotografiert. Andere wissenschaftliche Arbeiten, die sich mit einer Wundklassifizierung oder mit einer Wundsegmentierung beschäftigen, betrachten entweder andere Wundtypen, wie chronische Wunden, oder haben eine Kooperation mit einer medizinischen Einrichtung. Diese Datensätze werden aber nicht veröffentlicht.

7.2 Ausblick

Eine Weiterentwicklung der App kann sich auf verschiedene andere Bereiche konzentrieren. Dies kann z. B. die Verwendung von AR in der App sein, da Unity eine sehr gute Unterstützung von AR bietet. Die App kann um ein entsprechendes AR-Framework wie z. B. Vuforia, erweitert werden. Mithilfe von AR kann beispielsweise die Anbringung eines Druckverbands angezeigt und kontrolliert werden. Denkbar ist, dass der Anwender den Druckverband anbringt und dazu die einzelnen Schritte mittels AR direkt sehen kann. Die einzelnen Schritte und Aktionen müssen dabei mit verfolgt werden. Zuerst muss die Wunde auf dem Körperteil markiert werden, sodass auch beim Bewegen der Kamera die Wunde immer an derselben Stelle bleibt. Danach muss erkannt werden, ob das Körperteil hochgelagert wird, um die Blutzufuhr zu verringern. Beim Anbringen des Verbands ist darauf zu achten, dass die Wundauflage-das ist der Teil des Verbandes, der direkten Kontakt mit der Wunde hat- korrekt positioniert wird. Bei der Fixierung des Druckpolsters, das Druck auf die Wunde ausübt, um die Blutung zu stoppen, muss die Richtung, in der der Verband um den Arm gewickelt wird, erkannt werden. Mit der Integration von AR wird erreicht, dass der Anwender nicht nur eine textuelle Beschreibung der Handlungsanweisung bekommt, sondern auch eine optische.

Eine weitere Richtung, in die bei einer Weiterentwicklung gegangen werden kann, ist die Verbesserung und die Erweiterung des Wundklassifikators um weitere Verletzungen, wie zum Beispiel Blasen, Platz-, Riss-, Quetschwunden oder auch Pfählungsverletzungen. Auch andere Verletzungen der Haut sind interessante Anwendungen, wie z. B. Biss- und Stickverletzungen durch Insekten. Es kann einen Unterschied machen, ob der Anwender von einer Stechmücke, von Grasmilben oder von einer Zecke gebissen wurde. Dabei ist der Wundverlauf nützlich. Das Bild von Stichen kann als Ausgangspunkt für weitere Analysen benutzt werden. Während die Bisse von Grasmilben einfach heilen und ungefährlich sind, können vom Biss einer Zecke die gefährlichen Borrelien-Bakterien übertragen werden. Charakteristisch ist dabei die Wanderröte, eine ringförmige Hautrötung rund um die Einstichstelle, die nach einigen Tagen bis Wochen entstehen kann. Ein Modell kann trainiert werden, um dies zu erkennen und frühzeitig den Anwender zu warnen, um so früh wie möglich mit der Behandlung anzufangen.

Die Arbeit beschreibt auch das Problem, dass in diesem Bereich oft nicht genügend Daten vorhanden sind, um die Modelle gut zu trainieren. Eine Möglichkeit, dieses Problem zu beheben, ist die Verwendung des innovativen Ansatzes des Federated Machine (FL) Learnings. Im Kern beschreibt FL ein Verfahren, wie ein Modell dezentral trainiert werden kann. Zum Einsatz kommt es unter anderem bei mobilen Endgeräten. Die bisherigen Ansätze sind zentralisiert. Die Endgeräte übertragen ihre Daten an eine zentrale Komponente. Diese sammelt alle Daten und benutzt sie, um ein Modell zu trainieren. Nach dem Training wird das Modell für die Endgeräte wieder bereitgestellt, die das Modell dann herunterladen.

FL ermöglicht es, ein gemeinsames Modell zu erlernen und zu verwenden, wobei die Daten auf dem Endgerät bleiben und nicht an die zentrale Komponente geschickt werden müssen. Das spart zum einen viele Ressourcen und zum anderen besitzt die zentrale Komponente nicht die Rohdaten. Das ist besonders bei Anwendungen interessant, bei denen der Schutz der Daten im Vordergrund steht, was bei Gesundheitsdaten grundsätzlich der

Fall ist. FL verfolgt einen iterativen Ansatz. Das Training startet, in dem die Endgeräte das aktuelle Modell herunterladen. Danach beginnt jedes Gerät unabhängig von anderen mit dem Training des lokalen vorhandenen Datensatzes. Sobald das Training abgeschlossen ist, wird nur das trainierte Modell vom Endgerät an die zentrale Komponente zurückgeschickt. Die zentrale Komponente sammelt diese trainierten Modelle und führt sie mit der Bildung eines Mittelwerts zusammen. Das neue Modell wird daraufhin wieder dem Endgerät für die Vorhersagen bereitgestellt. Es sind mehrere Iterationen notwendig, da z. B. Endgeräte nicht erreicht werden können.

Literatur

[1] *10 Mythen der Wundheilung*. https://www.hansaplast.de/ratgeber/wundversorgung/10-mythen-der-wundheilung. Letzter Zugriff am 01.05.2021.

[2] *Actions as Space-Time Shapes*. http://www.wisdom.weizmann.ac.il/~vision/SpaceTimeActions.html. Letzer Zugriff am 01.05.2021.

[3] D. M. Anisuzzaman, Yash Patel, Jeffrey Niezgoda, Sandeep Gopalakrishnan und Zeyun Yu. „A Mobile App for Wound Localization using Deep Learning“. In: (Sep. 2020).

[4] *Anzahl der Unfallverletzten in Deutschland nach Unfallkategorie in den Jahren 2009 bis 2015*. https://de.statista.com/statistik/daten/studie/187486/umfrage/anzahl-der-unfallverletzten-in-deutschland-nach-unfallkategorie. Letzter Zugriff am 18.10.2021.

[5] Helmut Balzert. *Lehrbuch der Software-Technik: Basiskonzepte und Requirements Engineering*. 3. Aufl. Heidelberg: Spektrum, 2009. ISBN: 978-3-8274-1705-3. DOI: 10.1007/978-3-8274-2247-7.

[6] Nicolas Alejandro Borromeo. *Hands-On Unity 2020 Game Development - Build, customize, and optimize professional games using Unity 2020 and C#*. Packt Publishing Ltd, 2020. ISBN: 978-1-838-64760-5.

[7] Gary Bradski und Adrian Kaehler. *Learning OpenCV*. First Edition. O'Reilly Media, Inc., 2008. ISBN: 978-0-596-51613-0.

[8] Hartmuth Brandt und Rene Kerkmann. *Verbandstoffe für die Kitteltasche*. First Edition. Wissenschaftliche Verlagsgesellschaft Stuttgart, 2010. ISBN: 978-3-8047-2464-8.

[9] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff und Hartwig Adam. *Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation*. 2018. eprint: 1802.02611.

[10] Victor Williamson Chuanbo Wang D. M. Anisuzzaman. „Fully automatic wound segmentation with deep convolutional neural networks“. In: *Scientific Reports* (2020). DOI: https://doi.org/10.1038/s41598-020-78799-w.

[11] *Conversion of TensorFlow Classification Models and Launch with OpenCV Python*. https://docs.opencv.org/4.5.4/d1/d8f/tf_cls_tutorial_dnn_conversion.html. Letzter Zugriff am 03.11.2021.

[12] *Create TensorFlow Records*. https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/2.2.0/training.html#create-tensorflow-records. Letzter Zugriff am 17.10.2021.

[13] Aline Dresch, Daniel Pacheco Lacerda und José Antônio Valle Antunes. „Design Science Research“. In: *Design Science Research: A Method for Science and Technology Advancement*. Springer International Publishing, 2015. DOI: 10.1007/978-3-319-07374-3_4. URL: https://doi.org/10.1007/978-3-319-07374-3_4.

[14] *Eine kurze Einführung in den Unity Asset Store*. https://unity3d.com/de/quick-guide-to-unity-asset-store. Letzter Zugriff am 26.09.2021.

[15] Bundesministerium für Gesundheit. *Krankenhauszukunftsgesetz für die Digitalisierung von Krankenhäusern*. https://www.bundesgesundheitsministerium.de/krankenhauszukunftsgesetz.html. Letzter Zugriff am 18.09.2021.

[16] *Google Play - ASB App Erste Hilfe im Notfall*. https://play.google.com/store/apps/details?id=eu.asb.asb.app&hl=de&gl=US. Letzter Zugriff am 09.10.2021.

[17] *Google Play - Erste Hilfe Hand*. https://play.google.com/store/apps/details?id=net.atos.si.sol.ens.cv.handinjuryNative.Auva_Public_Key_signed&hl=de&gl=US. Letzter Zugriff am 09.10.2021.

[18] *Google Play - Erste-Hilfe-Techniken*. https://play.google.com/store/apps/details?id=com.devappsinc.FirstAidTechniques&hl=de&gl=US. Letzter Zugriff am 09.10.2021.

[19] *Google Play - Erste Hilfe Weisses Kreuz*. https://play.google.com/store/apps/details?id=my.company.iFirstAid&hl=de&gl=US. Letzter Zugriff am 09.10.2021.

[20] *Google Play - Erste Hilfe des SRK*. https://play.google.com/store/apps/details?id=com.cube.gdpc.che&hl=de&gl=US. Letzter Zugriff am 09.10.2021.

[21] *Google Play - Erste Hilfe für Kinder*. https://play.google.com/store/apps/details?id=com.dance.child.first.aid&hl=de&gl=US. Letzter Zugriff am 09.10.2021.

[22] *Google Play - Erste Hilfe – IFRC*. https://play.google.com/store/apps/details?id=com.cube.gdpc.fa&hl=de&gl=US. Letzter Zugriff am 09.10.2021.

[23] *Google Play - First Aid For Cyclists*. https://play.google.com/store/apps/details?id=com.sja.stjohnsroadsafety&hl=de&gl=US. Letzter Zugriff am 09.10.2021.

[24] *Google Play - First Aid Kit: First Aid and Emergency Techniques*. https://play.google.com/store/apps/details?id=com.fidel.omolo.smartfirstaid&hl=de&gl=US. Letzter Zugriff am 09.10.2021.

[25] *Google Play - First Aid and Emergency Techniques*. https://play.google.com/store/apps/details?id=com.fatbelly.firstaidsandemergencytechniques&hl=de&gl=US. Letzter Zugriff am 09.10.2021.

[26] *Google Play - First Aid for Emergency & Disaster Preparedness*. https://tinyurl.com/2bwksk7z. Letzter Zugriff am 09.10.2021.

[27] *Google Play - First Aid*. https://play.google.com/store/apps/details?id=com.viveogroup.firstaidkitselectronicguide&hl=de&gl=US. Letzter Zugriff am 09.10.2021.

[28] *Google Play - Instant Aid - Erste Hilfe App*. https://play.google.com/store/apps/details?id=finn.prietzel.instant_aid&hl=de&gl=US. Letzter Zugriff am 09.10.2021.

[29] *Google Play - Kindernotfall-App*. https://play.google.com/store/apps/details?id=de.juh.barmer.kindernotfall&hl=de&gl=US. Letzter Zugriff am 09.10.2021.

[30] *Google Play - St John Ambulance First Aid*. https://play.google.com/store/apps/details?id=com.sja.firstaid&hl=de&gl=US. Letzter Zugriff am 09.10.2021.

[31] Alan R Hevner, March Alan und T Salvatore. „Design Science in Information Systems Research“. In: *Management Information Systems Quarterly* (März 2004).

[32] Alan Hevner. „A Three Cycle View of Design Science Research“. In: *Scandinavian Journal of Information Systems* 19 (Jan. 2007).

[33] Van-Thanh Hoang und Kang-Hyun Jo. „Multi-Person Pose Estimation with Human Detection: A Parallel Approach“. In: *IECON 2018 - 44th Annual Conference of the IEEE Industrial Electronics Society*. 2018, S. 3269–3272. DOI: 10.1109/IECON.2018.8591434.

[34] Andrew G. Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le und Hartwig Adam. „Searching for MobileNetV3“. In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)* (2019), S. 1314–1324.

[35] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto und Hartwig Adam. *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. 2017. eprint: 1704.04861.

[36] Ahmad Jalal, Amir Nadeem und Satoshi Bobasu. „Human Body Parts Estimation and Detection for Physical Sports Movements“. In: *2019 2nd International Conference on Communication, Computing and Digital systems (C-CODE)*. 2019, S. 104–109. DOI: 10.1109/C-CODE.2019.8680993.

[37] Standford Vision Lab. *ImageNet*. . Letzter Zugriff am 15.10.2021.

[38] Robert Laganiere. *Opencv 3 Computer Vision Application Programming Cookbook*. Third Edition. Packt Publishing Ltd, 2017. ISBN: 978-0-596-51613-0.

[39] Maria Mahmood, Ahmad Jalal und M. A. Sidduqi. „Robust Spatio-Temporal Features for Human Interaction Recognition Via Artificial Neural Network“. In: *2018 International Conference on Frontiers of Information Technology (FIT)*. 2018, S. 218–223. DOI: 10.1109/FIT.2018.00045.

[40] Amir Nadeem, Ahmad Jalal und Kibum Kim. „Human Actions Tracking and Recognition Based on Body Parts Detection via Artificial Neural Network“. In: Jan. 2020. DOI: 10.1109/ICACS47775.2020.9055951.

[41] *OpenCV for Unity*. https://assetstore.unity.com/packages/tools/integration/opencv-for-unity-21088. Letzter Zugriff am 07.10.2021.

[42] Rafael Padilla, Sergio Netto und Eduardo da Silva. „A Survey on Performance Metrics for Object-Detection Algorithms“. In: Juli 2020. DOI: 10.1109/IWSSIP48289.2020.

[43] Benjamin Planche und Eliot Andreas. *Hands-On Computer Vision with TensorFlow 2*. First Edition. Packt Publishing Ltd, 2019. ISBN: 978-1-78883-064-5.

[44] *Protocol Buffers*. https://developers.google.com/protocol-buffers/?hl=en. Letzter Zugriff am 09.10.2021.

[45] *Protocol Buffers*. https://github.com/protocolbuffers/protobuf. Letzter Zugriff am 17.10.2021.

[46] Abhijeet Pujara. *Image classification with MobileNet*. 2020. URL: https://medium.com/analytics-vidhya/image-classification-with-mobilenet-cc6fbb2cd470.

[47] Sebastian Raschka und Vahid Mirjalili. *Machine Learning mit Python und Scikit-Learn und TensorFlow*. Heidelberg: MITP-Verlags GmbH & Co. KG, 2017. ISBN: 978-3-95845-733-1.

[48] *Recognition of human actions*. https://www.csc.kth.se/cvap/actions. Letzter Zugriff am 01.05.2021.

[49] M. S. Ryoo und J. K. Aggarwal. „Spatio-temporal relationship match: Video structure comparison for recognition of complex human activities“. In: *2009 IEEE 12th International Conference on Computer Vision*. 2009, S. 1593–1600. DOI: 10.1109/ICCV.2009.5459361.

[50] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov und Liang-Chieh Chen. „MobileNetV2: Inverted Residuals and Linear Bottlenecks“. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2018, S. 4510–4520. DOI: 10.1109/CVPR.2018.00474.

[51] *Save and load models*. https://www.tensorflow.org/tutorials/keras/save_and_load. Letzter Zugriff am 09.10.2021.

[52] Dr. med. Sonja Kempinski. *Erste Hilfe bei Verbrennungen*. https://www.apotheken.de/gesundheit/13046-erste-hilfe-bei-verbrennungen. Letzter Zugriff am 27.10.2021.

[53] Unity Technologies. *Unity Documentation - MonoBehaviour.Awake()*. https://docs.unity3d.com/ScriptReference/MonoBehaviour.Awake.html. Letzter Zugriff am 27.09.2021.

[54] Unity Technologies. *Unity Documentation - Prefabs*. https://docs.unity3d.com/Manual/Prefabs.html. Letzter Zugriff am 27.09.2021.

[55] Unity Technologies. *Unity Documentation - WebCamTexture*. https://docs.unity3d.com/ScriptReference/WebCamTexture-ctor.htmll. Letzter Zugriff am 06.10.2021.

[56] Silesian University of Technology. *WoundsDB*. https://chronicwounddatabase.eu. Letzter Zugriff am 08.05.2021.

[57] *TensorFlow 1 Detection Model Zoo*. https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf1_detection_zoo.md#coco-trained-models. Letzter Zugriff am 17.10.2021.

[58] *TensorFlow 1 Models*. https://github.com/tensorflow/models/tree/r1.13.0l. Letzter Zugriff am 17.10.2021.

[59] *TensorFlow 2 Detection Model Zoo*. https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md. Letzter Zugriff am 17.10.2021.

[60] *TensorFlow 2 Models*. https://github.com/tensorflow/models. Letzter Zugriff am 17.10.2021.

[61] *TensorFlow Object Detection API*. https://github.com/tensorflow/models/tree/master/research/object_detection. Letzter Zugriff am 17.10.2021.

[62] Rainer Thiel, Lucas Deimel, Daniel Schmidtmann, Klaus Piesche, Tobias Hüsing, Jonas Rennoch, Veli Stroetmann und Karl Stroetmann. „#SmartHealthSystems, Digitalisierungsstrategien im internationalen Vergleich“. In: (Okt. 2018).

[63] Stephen Thomas. *Medetec Wound Database*. http://www.medetec.co.uk/files/medetec-image-databases.html. Letzter Zugriff am 07.05.2021.

[64] Matteo R. Ronchi Tsung-Yi Lin Genevieve Patterson. *Common Objects in Context*. https://cocodataset.org. Letzter Zugriff am 16.06.2021.

[65] *Unity Documentation - TextureFormat*. https://docs.unity3d.com/ScriptReference/TextureFormat.html. Letzter Zugriff am 09.10.2021.

[66] *Vergleich der Marktanteile von Android und iOS am Absatz von Smartphones in Deutschland von Januar 2012 bis September 2021*. https://de.statista.com/statistik/daten/studie/256790/umfrage/marktanteile-von-android-und-ios-am-smartphone-absatz-in-deutschland/. Letzter Zugriff am 06.11.2021.

[67] Liu Wei, Anguelov Dragomir, Dumitru Erhan, Szegedy Christian, Reed Scott E., Fu Cheng-Yang und Berg Alexander C. „SSD: Single Shot MultiBox Detector“. In: *CoRR* abs/1512.02325 (2015). arXiv: 1512.02325. URL: http://arxiv.org/abs/1512.02325.

[68] Yi Yang und Deva Ramanan. „Articulated Human Detection with Flexible Mixtures of Parts“. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.12 (2013), S. 2878–2890. DOI: 10.1109/TPAMI.2012.261.

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Abkürzungsverzeichnis

ADB	Android Debug Bridge
APK	Android Application Package
AR	Augmented Reality
bat	Batch file
Blob	Binary Large Objects
CC0	Creative Commons Zero
CCL	Connected-component labeling
CNN	Convolutional Neural Network
COCO	Common Objects in Context
CPU	Central Processing Unit
CRUD	Create, Read, Update and Delete
CUDA	Compute Unified Device Architecture
cuDNN	CUDA Deep Neural Network Library
DAO	Data Access Object
DeptwiseSepConv	depthwise separable convolution
DI	Dependency Injection
DNN	Deep Neural Network
DSR	Design Science Research
ePA	elektronische Patientenakte
FC	fully-connected
FL	Federated Machine
FPS	Frames per second
GPU	Graphics Processing Unit
GUI	Graphical User Interface
hih	health innovation hub
IoU	Intersection Over Union
JSON	Java Script Object Notation
KHZG	Krankenhauszukunftsgesetz
KI	künstliche Intelligenz
mAP	Mean Average Precision
MEDETEC	Medical Device Technical Consultancy Service
MVC	Model View Controller
OpenCV	Open Source Computer Vision

PASCAL VOC	PASCAL Visual Object Classes
pb	Protobuf
PNG	Portable Network Graphic
Protobuf	Protocol Buffers
RCNN	Region Based Convolutional Neural Networks
ReLu	Rectified Linear Unit
ResNet	Residual Neural Network
RMSProp	Root Mean Square Propagation
SSD	Single Shot MultiBox Detector
TFOD	TensorFlow Object Detection API
UI	User Interface
UML	Unified Modeling Language
UWP	Universal Windows Platform
VGG16	Visual Geometry Group 16
VR	Virtual Reality
YOLO	You only look once
YOLOv3	You only look once Version 3

Anhang

A Lebenszyklus der Klasse MonoBehaviour

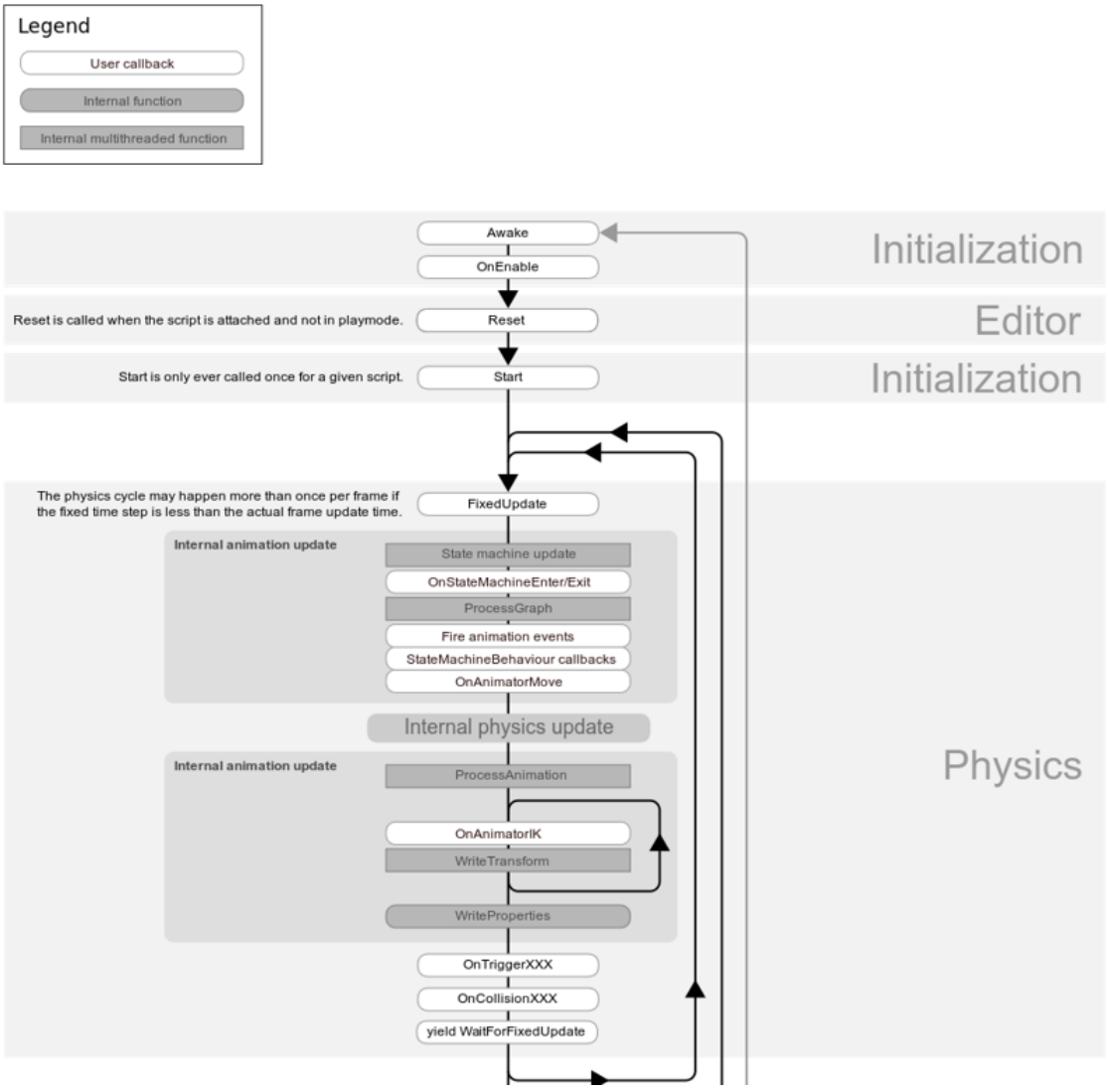


Abbildung A.1: Lebenszyklus der Klasse MonoBehaviour, Teil 1

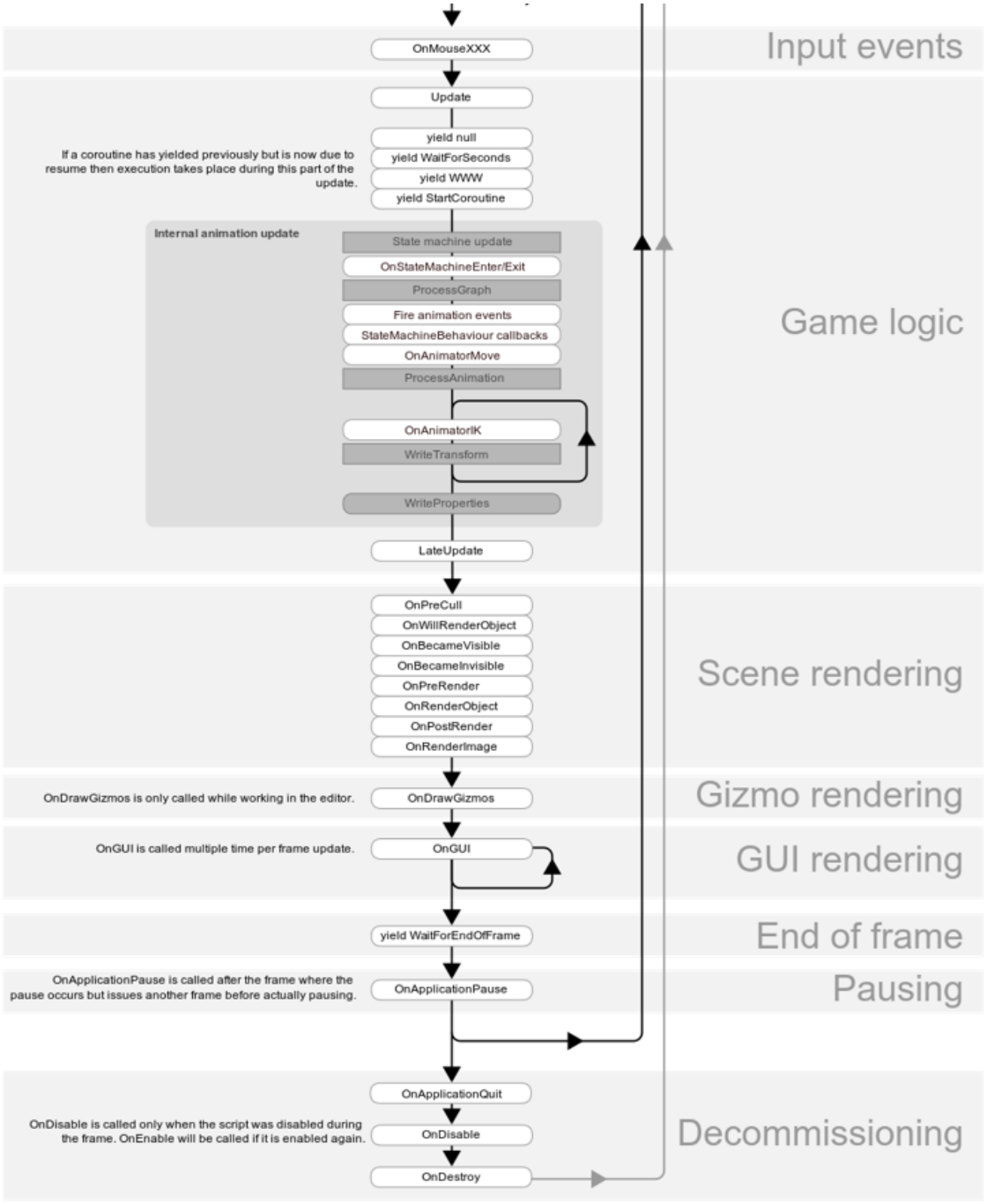


Abbildung A.2: Lebenszyklus der Klasse MonoBehaviour, Teil 2

B MobileNet

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Abbildung B.1: MobileNet Body Architektur [35, S. 4]

C Beispiel einer pipeline.config

Listing C.1: Beispiel einer pipeline.config vor dem Konfigurieren

```
model {
  ssd {
    num_classes: 90  # Set this to the number of different label classes
    image_resizer {
      fixed_shape_resizer {
        height: 320
        width: 320
      }
    }
    feature_extractor {
      type: "ssd_mobilenet_v2_fpn_keras"
      depth_multiplier: 1.0
      min_depth: 16
      conv_hyperparams {
        regularizer {
          l2_regularizer {
            weight: 3.9999998989515007e-05
          }
        }
        initializer {
          random_normal_initializer {
            mean: 0.0
            stddev: 0.009999999776482582
          }
        }
        activation: RELU_6
        batch_norm {
          decay: 0.996999979019165
          scale: true
          epsilon: 0.0010000000474974513
        }
      }
      use_depthwise: true
      override_base_feature_extractor_hyperparams: true
      fpn {
        min_level: 3
        max_level: 7
        additional_layer_depth: 128
      }
    }
    box_coder {
      faster_rcnn_box_coder {
        y_scale: 10.0
        x_scale: 10.0
        height_scale: 5.0
        width_scale: 5.0
      }
    }
    matcher {
      argmax_matcher {
        matched_threshold: 0.5
```

```
    unmatched_threshold: 0.5
    ignore_thresholds: false
    negatives_lower_than_unmatched: true
    force_match_for_each_row: true
    use_matmul_gather: true
  }
}
similarity_calculator {
  iou_similarity {
  }
}
box_predictor {
  weight_shared_convolutional_box_predictor {
    conv_hyperparams {
      regularizer {
        l2_regularizer {
          weight: 3.9999998989515007e-05
        }
      }
      initializer {
        random_normal_initializer {
          mean: 0.0
          stddev: 0.009999999776482582
        }
      }
      activation: RELU_6
      batch_norm {
        decay: 0.996999979019165
        scale: true
        epsilon: 0.0010000000474974513
      }
    }
    depth: 128
    num_layers_before_predictor: 4
    kernel_size: 3
    class_prediction_bias_init: -4.599999904632568
    share_prediction_tower: true
    use_depthwise: true
  }
}
anchor_generator {
  multiscale_anchor_generator {
    min_level: 3
    max_level: 7
    anchor_scale: 4.0
    aspect_ratios: 1.0
    aspect_ratios: 2.0
    aspect_ratios: 0.5
    scales_per_octave: 2
  }
}
post_processing {
  batch_non_max_suppression {
    score_threshold: 9.99999993922529e-09
    iou_threshold: 0.6000000238418579
    max_detections_per_class: 100
    max_total_detections: 100
    use_static_shapes: false
  }
  score_converter: SIGMOID
}
normalize_loss_by_num_matches: true
```

```
    loss {
      localization_loss {
        weighted_smooth_l1 {
        }
      }
      classification_loss {
        weighted_sigmoid_focal {
          gamma: 2.0
          alpha: 0.25
        }
      }
      classification_weight: 1.0
      localization_weight: 1.0
    }
    encode_background_as_zeros: true
    normalize_loc_loss_by_codesize: true
    inplace_batchnorm_update: true
    freeze_batchnorm: false
  }
}
train_config {
  batch_size: 128
  data_augmentation_options {
    random_horizontal_flip {
    }
  }
  data_augmentation_options {
    random_crop_image {
      min_object_covered: 0.0
      min_aspect_ratio: 0.75
      max_aspect_ratio: 3.0
      min_area: 0.75
      max_area: 1.0
      overlap_thresh: 0.0
    }
  }
  sync_replicas: true
  optimizer {
    momentum_optimizer {
      learning_rate {
        cosine_decay_learning_rate {
          learning_rate_base: 0.07999999821186066
          total_steps: 50000
          warmup_learning_rate: 0.026666000485420227
          warmup_steps: 1000
        }
      }
      momentum_optimizer_value: 0.8999999761581421
    }
    use_moving_average: false
  }
  fine_tune_checkpoint: "PATH_TO_BE_CONFIGURED"
  num_steps: 50000
  startup_delay_steps: 0.0
  replicas_to_aggregate: 8
  max_number_of_boxes: 100
  unpad_groundtruth_tensors: false
  fine_tune_checkpoint_type: "classification"
  fine_tune_checkpoint_version: V2
}
train_input_reader {
  label_map_path: "PATH_TO_BE_CONFIGURED"
```

```
  tf_record_input_reader {
    input_path: "PATH_TO_BE_CONFIGURED"
  }
}
eval_config {
  metrics_set: "coco_detection_metrics"
  use_moving_averages: false
}
eval_input_reader {
  label_map_path: "PATH_TO_BE_CONFIGURED"
  shuffle: false
  num_epochs: 1
  tf_record_input_reader {
    input_path: "PATH_TO_BE_CONFIGURED"
  }
}
```

D Entwurf GUI

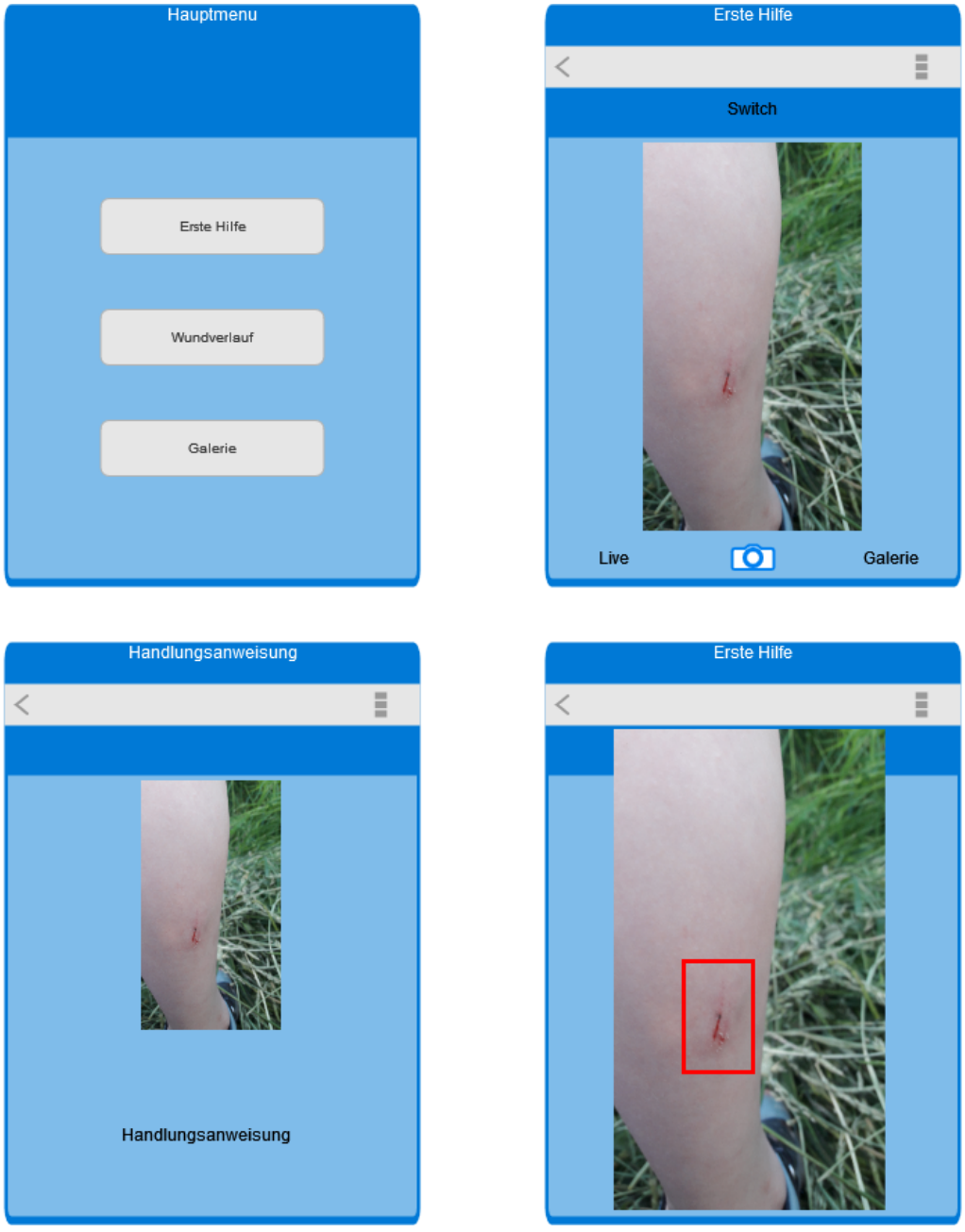


Abbildung D.1: GUI, Teil 1

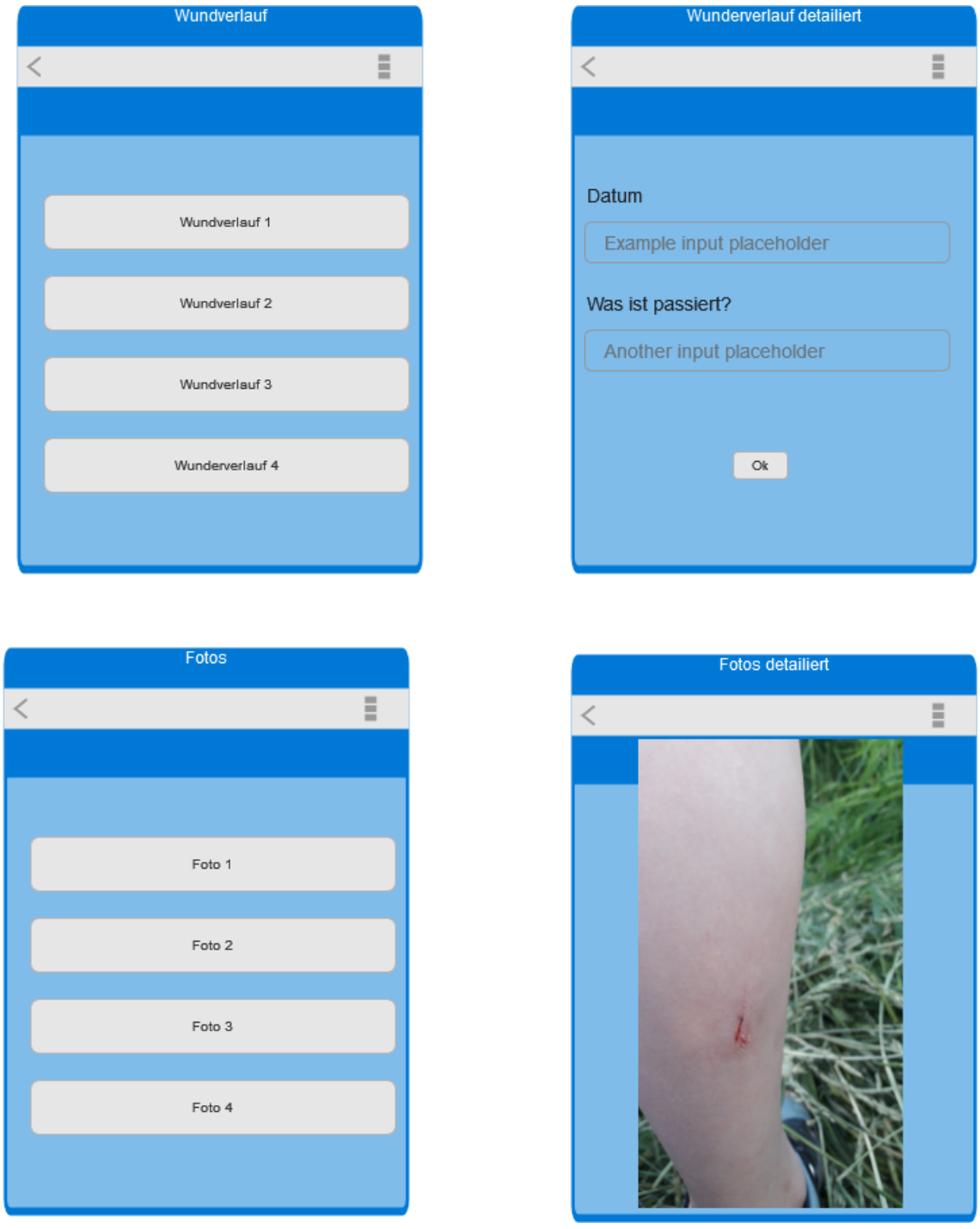


Abbildung D.2: GUI, Teil 2

E Klassendiagramm

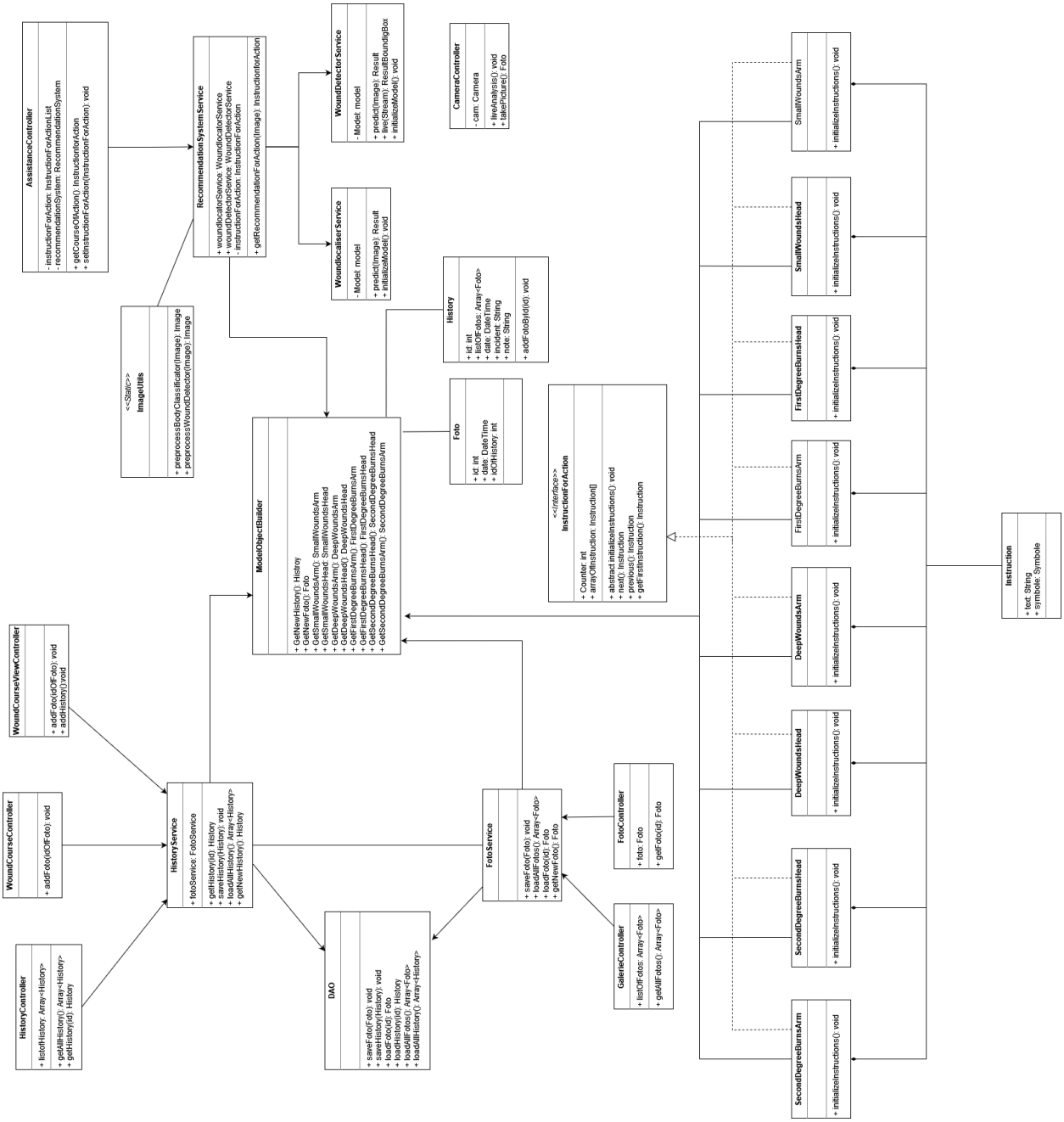


Abbildung E.1: Klassendiagramm mit den wichtigsten Klassen

F Handlungsanweisungen

Tabelle F.1: Handlungsanweisungen, Teil 1 [vgl. 8, S. 9 bis 11]

Wunde	**Körperteil**	**Schritt**	**Beschreibung**	Symbole
Tiefe Schnittverletzung	Kopf			
		1	Setzen Sie den Verletzen hin.	
		2	Lagern Sie die Schnittverletzung hoch.	
		3	Bringen Sie bei starkem Bluten einen Druckverband an.	
		4	Rufen die den Notruf 112 an oder begeben Sie sich in ärtzliche Behandlung.	
Tiefe Schnittverletzung	Arm			
		1	Setzen Sie den Verletzen hin.	
		2	Lagern Sie die Schnittverletzung hoch.	
		3	Bringen Sie bei starkem Bluten einen Druckverband an.	
Verbrennung 1. Grades	Arm			
		1	Kühlen Sie die Verbrennung unter fließendem Wasser. Wichtig kühlen Sie nicht die Wunde mit Eiswürfeln oder kalten Kompressen. Kühlen Sie maximal 10 Minuten.	
		2	Die Verbrennung heilt in der Regel ohne Probleme ab. Zur Unterstützung der Wundheilung können Sie Brand- und Wundgel benutzen.	

Tabelle F.2: Handlungsanweisungen, Teil 2

Wunde	**Körperteil**	**Schritt**	**Beschreibung**	Symbole
Verbrennung 1. Grades	Kopf			
		1	Kühlen Sie die Verbrennung unter fließendem Wasser. Wichtig kühlen Sie nicht die Wunde mit Eiswürfeln oder kalten Kompressen. Kühlen Sie maximal 10 Minuten.	
		2	Die Verbrennung heilt in der Regel ohne Probleme ab. Zur Unterstützung der Wundheilung können Sie Brand- und Wundgel benutzen.	
Verbrennung 2. Grades	Arm			
		1	Kühlen Sie nicht die Wunde.	
		2	Nehmen Sie ein steriles Verbandtuch und legen es auf die Wunde.	
		3	Nehmen Sie eine Fixierbinde und wickeln Sie die Binde um die Wunde.	
		4	Öffnen Sie unter keinen Umständen die vorhanden Blasen. Es besteht Infektionsgefahr.	
Verbrennung 2. Grades	Kopf			
		1	Kühlen Sie nicht die Wunde.	
		2	Nehmen Sie ein steriles Verbandtuch und legen es auf die Wunde.	
		3	Nehmen Sie eine Fixierbinde und wickeln Sie die Binde um die Wunde.	
		4	Öffnen Sie unter keinen Umständen die vorhanden Blasen. Es besteht Infektionsgefahr.	
		5	Rufen Sie den Notruf 112 an oder begeben Sie sich in ärtzliche Behandlung.	

Tabelle F.3: Handlungsanweisungen, Teil 3

Wunde	**Körperteil**	**Schritt**	**Beschreibung**	Symbole
Kleine Schnittwunde	Arm			
		1	Kleine Schnittverletzungen müssen in der Regel nicht durch einen Arzt behandelt werden. Lassen Sie die Wunde etwas ausbluten. Es werden so kleine Fremdkörper und Keime aus der Wunde geschwemmt.	
		2	Sprühen Sie die Wunde mit einem Desinfektionsspray ein und lassen Sie es abdunsten.	
		3	Bringen Sie ein Pflaster an.	
Kleine Schnittwunde	Kopf			
		1	Kleine Schnittverletzungen müssen in der Regel nicht durch einen Arzt behandelt werden. Lassen Sie die Wunde etwas aubluten. Es werden so kleine Fremdkörper und Keime aus der Wunde geschemmt.	
		2	Bringen Sie ein Pflaster an.	

G Quellcode für den Wundlokalisator

Listing G.1: Jupyter Notebook

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.layers import Dense, Activation
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.metrics import categorical_crossentropy
from tensorflow.keras.models import Model
from sklearn.metrics import confusion_matrix
from sklearn.metrics import plot_confusion_matrix
from tensorflow.keras import preprocessing
from tensorflow.keras.preprocessing import image
from tensorflow.python.keras import activations
from tensorflow.python.keras.applications import imagenet_utils
from tensorflow.python.keras.preprocessing.image import
    ImageDataGenerator
from tensorflow.keras.preprocessing.image import DirectoryIterator
from tensorflow.keras.layers import GlobalAveragePooling2D
import cv2 as cv
import matplotlib.pyplot as plt



physical_devices = tf.config.experimental.list_physical_devices('GPU')
print("Num GPUs Avail: ", len(physical_devices))
tf.config.experimental.set_memory_growth(physical_devices[0], True)

mobile = tf.keras.applications.mobilenet.MobileNet(classes=1000,
    classifier_activation='softmax',weights='imagenet')

x= mobile.layers[-6].output
x=Dense(1024,activation='relu')(x) #we add dense layers so that the model can
    learn more complex functions and classify for better results.
x=Dense(1024,activation='relu')(x) #dense layer 2
x=Dense(512,activation='relu')(x) #dense layer 3
output=Dense(3,activation='softmax')(x)
model = Model(inputs=mobile.input, outputs=output)

data_dir = 'F:\\First-Aid-App\\BodyPartRecognition\\Images\\DatasetCNN'
batch_size= 10

train_set = tf.keras.utils.image_dataset_from_directory(
  data_dir,
  validation_split=0.2,
  subset="training",
  seed=123,
  image_size=(224, 224),
  batch_size=batch_size,
  label_mode = "categorical")
```

```
vaild_set = tf.keras.utils.image_dataset_from_directory(
  data_dir,
  validation_split=0.2,
  subset="validation",
  seed=123,
  image_size=(224, 224),
  batch_size=batch_size,
  label_mode = "categorical")



model.compile(optimizer=tf.keras.optimizers.RMSprop(learning_rate=0.0001)
    , loss='categorical_crossentropy', metrics=['accuracy'])

history = model.fit(x=train_set, validation_data=vaild_set, epochs=12)

#Plot loss and accuracy
history_dict = history.history
train_loss = history_dict['loss']
val_loss = history_dict['val_loss']
train_acc = history_dict['accuracy']
val_acc = history_dict['val_accuracy']

epochs = range(1, len(train_loss) + 1)

plt.title('Training and validation loss')
plt.plot(epochs, train_loss, label='Training loss')
plt.plot(epochs, val_loss, label='Validation loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.grid()
plt.show()


plt.title('Training and validation accuracy')
plt.plot(epochs, train_acc, label='Training accuracy')
plt.plot(epochs, val_acc, label='Validation accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.grid()
plt.show()


#Freeze Graph
from tensorflow.python.framework.convert_to_constants import
    convert_variables_to_constants_v2

path_to_save_file = 'mobilenet_graph'
model.save(path_to_save_file)


loaded = tf.saved_model.load('mobilenet_graph')
infer = loaded.signatures['serving_default']

f = tf.function(infer).get_concrete_function(input_1=tf.TensorSpec(shape
    =[None, 224, 224, 3], dtype=tf.float32))
f2 = convert_variables_to_constants_v2(f)
graph_def = f2.graph.as_graph_def()
```

```
# Export frozen graph
with tf.io.gfile.GFile('frozen_graph.pb', 'wb') as f:
   f.write(graph_def.SerializeToString())
```

Listing G.2: Python-Skript zur Augmentation

```
from imageio.core.util import Image
import numpy as np
import imgaug.augmenters as iaa
import imgaug as ia
import cv2
import numpy as np
import imageio
import glob

path = "G:\\wundlokalisator\\Images\\Raw\Dataset\\training\\0"

def augImage(img):
    flip = iaa.Fliplr(0.5)
    crop = iaa.Crop(percent=(0, 0.1))

    img_fliped = flip(image=img)
    img_crop = crop(image=img)

    return (img_fliped, img_crop )


def getFilenames():
    filenames = [img for img in glob.glob(path + "\*.jpg")]
    return filenames

filenames = getFilenames()

for index, filename in enumerate(filenames):
    print (str(index) + ": Augment file: "  + filename)
    img = cv2.imread(filename)

    img1, img2 = augImage(img)

    path1 = pathAugmentedImages + "Image_" + str(index) + ".jpg"

    print (str(index) + ": Write  "  + pathAugmentedImages + "Image_Aug_"
        + str(index) + ".jpg")
    cv2.imwrite(pathAugmentedImages + "Image_Aug_" + str(index) + ".jpg",
        img1)


    cv2.imwrite(pathAugmentedImages + "Image1_Aug_" + str(index) + ".jpg"
       , img2)
    print (str(index) + ": Write  "  + pathAugmentedImages + "Image_Aug1_
       " + str(index) + ".jpg")
```

H Quellcode für den Wundklassifikator

Listing H.1: pipline.config für den Wundklassifikator

```
model {
  ssd {
    inplace_batchnorm_update: true
    freeze_batchnorm: false
    num_classes: 4
    box_coder {
      faster_rcnn_box_coder {
        y_scale: 10.0
        x_scale: 10.0
        height_scale: 5.0
        width_scale: 5.0
      }
    }
    matcher {
      argmax_matcher {
        matched_threshold: 0.5
        unmatched_threshold: 0.5
        ignore_thresholds: false
        negatives_lower_than_unmatched: true
        force_match_for_each_row: true
        use_matmul_gather: true
      }
    }
    similarity_calculator {
      iou_similarity {
      }
    }
    encode_background_as_zeros: true
    anchor_generator {
      ssd_anchor_generator {
        num_layers: 6
        min_scale: 0.2
        max_scale: 0.95
        aspect_ratios: 1.0
        aspect_ratios: 2.0
        aspect_ratios: 0.5
        aspect_ratios: 3.0
        aspect_ratios: 0.3333
      }
    }
    image_resizer {
      fixed_shape_resizer {
        height: 320
        width: 320
      }
    }
    box_predictor {
      convolutional_box_predictor {
        min_depth: 0
        max_depth: 0
        num_layers_before_predictor: 0
```

```
        use_dropout: false
        dropout_keep_probability: 0.8
        kernel_size: 3
        use_depthwise: true
        box_code_size: 4
        apply_sigmoid_to_scores: false
        class_prediction_bias_init: -4.6
        conv_hyperparams {
          activation: RELU_6,
          regularizer {
            l2_regularizer {
              weight: 0.00004
            }
          }
          initializer {
            random_normal_initializer {
              stddev: 0.03
              mean: 0.0
            }
          }
          batch_norm {
            train: true,
            scale: true,
            center: true,
            decay: 0.97,
            epsilon: 0.001,
          }
        }
      }
    }
    feature_extractor {
      type: 'ssd_mobilenet_v3_large'
      min_depth: 16
      depth_multiplier: 1.0
      use_depthwise: true
      conv_hyperparams {
        activation: RELU_6,
        regularizer {
          l2_regularizer {
            weight: 0.00004
          }
        }
        initializer {
          truncated_normal_initializer {
            stddev: 0.03
            mean: 0.0
          }
        }
        batch_norm {
          train: true,
          scale: true,
          center: true,
          decay: 0.97,
          epsilon: 0.001,
        }
      }
      override_base_feature_extractor_hyperparams: true
    }
    loss {
      classification_loss {
        weighted_sigmoid_focal {
          alpha: 0.75,
```

```
          gamma: 2.0
        }
      }
      localization_loss {
        weighted_smooth_l1 {
          delta: 1.0
        }
      }
      classification_weight: 1.0
      localization_weight: 1.0
    }
    normalize_loss_by_num_matches: true
    normalize_loc_loss_by_codesize: true
    post_processing {
      batch_non_max_suppression {
        score_threshold: 1e-8
        iou_threshold: 0.58
        max_detections_per_class: 100
        max_total_detections: 100
        use_static_shapes: true
      }
      score_converter: SIGMOID
    }
  }
}

train_config: {
  batch_size: 32
  sync_replicas: true
  startup_delay_steps: 0
  replicas_to_aggregate: 32
  num_steps: 400000
  data_augmentation_options {
    random_horizontal_flip {
    }
  }
  data_augmentation_options {
    random_vertical_flip {
    }
  }
data_augmentation_options {
    ssd_random_crop {
    }
  }
  optimizer {
    momentum_optimizer: {
      learning_rate: {
        cosine_decay_learning_rate {
          learning_rate_base: 0.4
          total_steps: 400000
          warmup_learning_rate: 0.13333
          warmup_steps: 2000
        }
      }
      momentum_optimizer_value: 0.9
    }
    use_moving_average: false
  }
  max_number_of_boxes: 100
  unpad_groundtruth_tensors: false
}
```

```
train_input_reader: {
  tf_record_input_reader {
    input_path: "annotations/train.record"
  }
  label_map_path: "annotations/label_map.pbtxt"
}

eval_config: {
  num_examples: 8000
  # Note: The below line limits the evaluation process to 10 evaluations.
  # Remove the below line to evaluate indefinitely.
  max_evals: 10
}

eval_input_reader: {
  tf_record_input_reader {
    input_path: "annotations/test.record"
  }
  label_map_path: "annotations/label_map.pbtxt"
  shuffle: false
  num_readers: 1
}
```

Listing H.2: Klasse WoundDetectorService

```
public class WoundDetectorService : MonoBehaviour
{
    public double heightOfResize;
    public double widthOfResize;
    private Net model;
    private string pathToModel;
    private string pathTographPbtxt;
    public float confidenceThreshold;
    public DAO dao;
    public string[\] labels;
    public filenamePB;
    public filenamePBTXT

    public void Awake()
    {
        pathToModel = Application.dataPath + "/TensorflowModels/WoundSSD/
            " + filenamePB;
        pathTographPbtxt = Application.dataPath + "/TensorflowModels/
            WoundSSD/" + filenamePBTXT;

        if (Application.platform == RuntimePlatform.Android)
        {
            pathToModel = Utils.getFilePath(filenamePB);
            pathTographPbtxt = Utils.getFilePath(filenamePBTXT);
        }

        model = Dnn.readNetFromTensorflow(pathToModel,pathTographPbtxt);
    }


    public String[] getLabels()
    {
        return labels;
    }
```

```
public Mat detectWounds(Mat frame)
{
    Mat resizeimage;
    resizeimage = Dnn.blobFromImage(frame, 1.0, new Size(
        widthOfResize, heightOfResize));
    model.setInput(resizeimage);
    Mat output = model.forward();

    return output;
}

public Texture2D drawBox(Mat output, Mat frame)
{

    int sizeMaxBoxes = output.size(2);
    output = output.reshape(1, output.size(2));

    for (int i = 0; i < sizeMaxBoxes; i++)
    {
        float confidence;
        int class_id;
        string className;
        float[] data = new float[7];

        output.get(i, 0, data);

        confidence = data[2];

        if (confidence > confidenceThreshold)
        {
            class_id = (int)data[1];
            className = labels[class_id - 1];
            float box_x = data[3] * frame.width();
            float box_y = data[4] * frame.height();
            float box_width = data[5] * frame.width();
            float box_height = data[6] * frame.height();
            new Point(box_x, box_y);
            new Point(box_width, box_height);
            Imgproc.rectangle(frame, new Point(box_x, box_y), new
                Point(box_width, box_height), new Scalar(0, 0, 255),
                3);
            Imgproc.putText(frame, className + ": " + confidence, new
                 Point(box_x, box_y - 5), Imgproc.
                FONT_HERSHEY_SIMPLEX, 1, new Scalar(0, 0, 255), 3);
        }
    }
    Texture2D texture2D = new Texture2D(frame.width(), frame.height()
        , TextureFormat.RGBA32, false);
    Utils.matToTexture2D(frame, texture2D);

    return texture2D;
}


public float[] classifiesWounds(Mat frame)
{

    Mat resizeimage;
    float[] predictions = new float[2];
    float[] result = new float[7] { 0, 0, 0, 0, 0, 0, 0 };

    resizeimage = Dnn.blobFromImage(frame, 1.0, new Size(
```

```
            widthOfResize, heightOfResize));
        model.setInput(resizeimage);
        Mat output = model.forward();
        int sizeMaxBoxes = output.size(2);
        output = output.reshape(1, output.size(2));


        for (int i = 0; i < sizeMaxBoxes; i++)
        {
            float[] temp = new float[7];

            output.get(i, 0, temp);

            //Compare the probabilities
            if (temp[2] > result[2])
            {
                result = temp;
            }
        }
        //Class
        predictions[0] = result[1];

        //Probability
        predictions[1] = result[2];

        return predictions;
    }
}
```

I Quellcode für die wichtigsten Klassen der Unity-App

Listing I.1: getAllFotos-Methode der Klasse FotoRepository

```
public List<Foto> getAllFotos()
{
    List<Foto> arrayOfFotos = new List<Foto>();
    Regex regex;
    Match match;

        regex = new Regex("[0-9]*");


        DirectoryInfo directoryInfo = new DirectoryInfo(pathToFotos);
        FileInfo[] fileInfo = directoryInfo.GetFiles();

        foreach (FileInfo file in fileInfo)
        {
            Foto foto;
            match = regex.Match(file.Name);
            foto = LoadFoto(int.Parse(match.Groups[0].Value));

            arrayOfFotos.Add(foto);
        }

        return arrayOfFotos;
    }
```

Listing I.2: LoadPictureTexture2D-Methode der Klasse FotoRepository

```
public Texture2D LoadPictureTexture2D(int id)
{
    string path = pathToPicture + "/";

    Texture2D tex = null;
    byte[] fileData;

    if (File.Exists(path + id + ".png"))
    {
        fileData = File.ReadAllBytes(path + id + ".png");
        tex = new Texture2D(2, 2);
        tex.LoadImage(fileData);
    }

    if (Application.platform == RuntimePlatform.Android)
    {
        tex = ImageUtil.rotateTexture(tex, true);
    }

    return tex;
}
```

Listing I.3: Update-Methode der Klasse TouchControl

```
void Update()
{
    foreach (Touch touch in Input.touches)
    {
        if (touch.tapCount == 0)
        {
                isRectTransformEnabled = (isRectTransformEnabled == true)
                    ? false : true;
        }

    if (isRectTransformEnabled) {
        if (Input.touches.Length == 2)
        {
            Touch t1 = Input.touches[0];
            Touch t2 = Input.touches[1];

            if (t1.phase == TouchPhase.Began || t2.phase == TouchPhase.
                Began)
            {
                initialFingersDistance = Vector2.Distance(t1.position, t2
                    .position);
                initialScale = gameObject.transform.localScale;
            }
            else if (t1.phase == TouchPhase.Moved || t2.phase ==
                TouchPhase.Moved)
            {
                var currentFingersDistance = Vector2.Distance(t1.position
                    , t2.position);
                var scaleFactor = currentFingersDistance /
                    initialFingersDistance;

                gameObject.transform.localScale = initialScale *
                    scaleFactor;

                Vector3 diff = t1.position - t2.position;
                var angle = (Mathf.Atan2(diff.y, diff.x));
                transform.rotation = Quaternion.Euler(0f, 0f, Mathf.
                    Rad2Deg * angle);
            }
        }

        if (Input.touches.Length == 1)
        {
            Touch t1 = Input.touches[0];
            if (t1.phase == TouchPhase.Began || t1.phase == TouchPhase.
                Moved )
            {
                initialPosition = t1.position;
            }

        }
    }
}
```

Listing I.4: HistoryList Klasse

```
public class HistoryList : MonoBehaviour
{
    public DAO dao;
```

```
    public string textNoHistory;

    void Start()
    {
        List<History> arrayOfHistory;
        Text text;
        GameObject noHistory;
        GameObject template = transform.GetChild(0).gameObject;
        GameObject game;

        arrayOfHistory = dao.getAllHistories();

        if (arrayOfHistory.Count != 0)
        {
            foreach (History history in arrayOfHistory)
            {
                game = Instantiate(template, transform);
                text = game.GetComponentInChildren<Text>();
                text.text = history.DateAsString;
                game.GetComponentInChildren<HistoryManager>().HistoryId =
                    history.id;
            }
        }
        else
        {
            noHistory = GameObject.Find("NoHistory");
            noHistory.GetComponent<Text>().text = textNoHistory;
        }
        Destroy(template);
    }
}
```

Listing I.5: Klasse WoundCourseViewFotoList

```
public class WoundCourseViewFotoList : MonoBehaviour
{
    public DAO dao;
    public string textNoFoto;
    private GameObject template;
    public WoundCourseViewManager woundCourseViewManager;

    void Start()
    {
        ArrayList arrayOfFotos;
        Text text;
        GameObject noFotos;
        template = transform.GetChild(0).gameObject;
        GameObject game;

        arrayOfFotos = woundCourseViewManager.getArrayOfFotosId();

        arrayOfFotos.Sort();
        arrayOfFotos.Reverse();


        if (arrayOfFotos.Count != 0)
        {
            foreach (int fotoId in arrayOfFotos)
            {

                Foto foto = dao.LoadFoto(fotoId);
```

```
                game = Instantiate(template, transform);
                text = game.GetComponentInChildren<Text>();
                text.text = foto.DateAsString;

                game.GetComponentInChildren<FotoManager>().IdOfFoto =
                    foto.Id;

                Texture2D texture2D = new Texture2D(150, 150);
                texture2D = dao.LoadPictureTexture2D(foto.IdOfPicture);
                game.GetComponentInChildren<RawImage>().texture =
                    texture2D;
            }
        }
        else
        {
            noFotos = GameObject.Find("NoFotos");
            noFotos.GetComponent<Text>().text = textNoFoto;
        }
        template.SetActive(false);
    }

    public void updateList()
    {
        ArrayList arrayOfFotos;
        Text text;
        GameObject noFotos;
        GameObject game;
        GameObject[] currentFotos;

        //Clears the list
        currentFotos = GameObject.FindGameObjectsWithTag("FotosWound");
        foreach (GameObject gameObject in currentFotos)
        {
            Destroy(gameObject);

        }

        template.SetActive(true);
        arrayOfFotos = woundCourseViewManager.getArrayOfFotosId();
        arrayOfFotos.Sort();
        arrayOfFotos.Reverse();

        if (arrayOfFotos.Count != 0)
        {
            foreach (int fotoId in arrayOfFotos)
            {
                Foto foto = dao.LoadFoto(fotoId);
                game = Instantiate(template, transform);
                text = game.GetComponentInChildren<Text>();
                text.text = foto.DateAsString;
                game.GetComponentInChildren<FotoManager>().IdOfFoto =
                    foto.Id;

                //Load the thumbnail
                Texture2D texture2D = new Texture2D(150, 150);
                texture2D = dao.LoadPictureTexture2D(foto.IdOfPicture);
                game.GetComponentInChildren<RawImage>().texture =
                    texture2D;
            }
        }
        else
```

```
        {
            noFotos = GameObject.Find("NoFotos");
            noFotos.GetComponent<Text>().text = textNoFoto;
        }
        template.SetActive(false);
    }
}
```

Listing I.6: Klasse FotoManager

```
public class FotoManager : MonoBehaviour
{
    public int IdOfFoto;

    public void showPicture()
    {
        FotoStatic.fotoId = gameObject.GetComponentInChildren<FotoManager
            >().IdOfFoto;
        SceneManager.LoadScene("Foto");
    }

    public void showPictureOnWoundCourseDetailed()
    {
        FotoStatic.fotoId = gameObject.GetComponentInChildren<FotoManager
            >().IdOfFoto;
        SceneManager.LoadScene("FotoOfWoundCourse");
    }

    public void showFotoOfWoundCourseView()
    {
        FotoStatic.fotoId = gameObject.GetComponentInChildren<FotoManager
            >().IdOfFoto;
        SceneManager.LoadScene("FotoOfWoundCourseView");
    }
}
```

Listing I.7: Klasse FotoLoader

```
public class FotoLoader : MonoBehaviour
{
    public DAO dao;
    public RawImage background;

    void Start()
    {
        Mat imageAsMat;
        Foto foto;
        int fotoId;
        fotoId = FotoStatic.fotoId;
        Texture2D texture;

        foto = dao.LoadFoto(fotoId);
        imageAsMat = dao.LoadPicture(foto.IdOfPicture);

        texture = new Texture2D(imageAsMat.width(), imageAsMat.height(),
            TextureFormat.RGBA32, false);
        Utils.matToTexture2D(imageAsMat, texture);

        background.texture = texture;
    }
```

```
}
```

Listing I.8: Klasse Recommander

```
public class Recommander : MonoBehaviour
{
    private InstructionForAction InstructionForAction;
    public DAO dao;
    public Woundlocaliser woundlocator;
    public WoundDetectorService woundDetectorService;
    float[] predictionsWounds;
    public ModelObjectBuilder modelObjectBuilder;


    public Recommandation AnalyseImageByTexture2D(Texture2D frame)
    {
        int predictedWoundType;
        Mat img;
        float[] predictionsBodyParts;
        int predictedBodyPart;
        Recommandation recommandation;


        img = new Mat(frame.height, frame.width, CvType.CV_8UC3);
        recommandation = modelObjectBuilder.GetRecommandation();


        Utils.texture2DToMat(frame, img);

        predictionsBodyParts = woundlocator.Predict(img);
        predictionsWounds = woundDetectorService.classifiesWounds(img);
        predictedBodyPart = getIndexOfArrayByMaxValue(
            predictionsBodyParts);
        predictedWoundType = (int)predictionsWounds[0];

        recommandation.instructionForAction = LoadInstructionForAction(
            predictedBodyPart, predictedWoundType);
        recommandation.woundlocation = predictedBodyPart;
        recommandation.confidenceWoundlocation = predictionsBodyParts[
            predictedBodyPart];
        recommandation.woundclass = predictedWoundType;
        recommandation.confidenceWoundclassification = predictionsWounds
            [1];

        return recommandation;
    }

    private InstructionForAction LoadInstructionForAction(int
        predictedBodyPart, int predictedWoundType)
    {
        Recommandation recommandation = modelObjectBuilder.
            GetRecommandation();

        if (predictedBodyPart == 0 || predictedBodyPart == 2)
        {
            switch (predictedWoundType)
            {
                case 1:
                    InstructionForAction = modelObjectBuilder.
                        GetSmallWoundsArm();
                    break;
```

```
                case 2:
                    InstructionForAction = modelObjectBuilder.
                        GetDeepWoundsArm();
                    break;
                case 3:
                    InstructionForAction = modelObjectBuilder.
                        GetFirstDegreeBurnsArm();
                    break;
                case 4:
                    InstructionForAction = modelObjectBuilder.
                        GetSecondDegreeBurnsArm();
                    break;
            }
        }

        if (predictedBodyPart == 1)
        {
            switch (predictedWoundType)
            {
                case 1:
                    InstructionForAction = modelObjectBuilder.
                        GetSmallWoundsHead();
                    break;
                case 2:
                    InstructionForAction = modelObjectBuilder.
                        GetDeepWoundsHead();
                    break;
                case 3:
                    InstructionForAction = modelObjectBuilder.
                        GetFirstDegreeBurnsHead();
                    break;
                case 4:
                    InstructionForAction = modelObjectBuilder.
                        GetSecondDegreeBurnsHead();
                    break;
            }
        }
        return InstructionForAction;
    }

    private int getIndexOfArrayByMaxValue(float[] array)
    {
        float max = array[0];
        int position = 0;


        for (int i = 1; i < array.Length; i++)
        {
            if (max < array[i])
            {
                max = array[i];
                position = i;
            }
        }

        return position;
    }

}
```

Listing I.9: Methode initializeInstructions der Klasse SmallWoundsArm

```
public override void initializeInstructions()
{
    arrayOfInstruction = new Instruction[3];

    Instruction instruction_0 = new Instruction();
    instruction_0.text = "Kleine Schnittverletzungen muessen in der
        Regel nicht durch einen Arzt behandelt werden. Lassen Sie die
            Wunde etwas ausbluten. Es werden so
        kleine Fremdkoerper und Keime aus der Wunde geschemmt.";
    instruction_0.symbole = new string[1];
    instruction_0.symbole[0] = "pflaster";
    arrayOfInstruction[0] = instruction_0;

    Instruction instruction_1 = new Instruction();
    instruction_1.text = "Spruehen Sie die Wunde mit einem
       Desinfektionsspray ein und lassen Sie es abdunsten. ";
    instruction_1.symbole = new string[1];
    instruction_1.symbole[0] = "pflaster";
    arrayOfInstruction[1] = instruction_1;

    Instruction instruction_2 = new Instruction();
    instruction_2.text = "Bringen Sie ein Pflaster an.";
    instruction_2.symbole = new string[1];
    instruction_2.symbole[0] = "pflaster";
    arrayOfInstruction[2] = instruction_2;
}
```

Listing I.10: Klasse AssistanceList

```
public class AssistanceList : MonoBehaviour
{
    private InstructionForAction instructionForAction;
    private Instruction instruction;
    public Text instructionAsString;
    public RawImage aidSymbole;
    public GameObject aidSymboleGameObject;


    public void Start()
    {
        //get the text of the first instruction
        instruction = instructionForAction.getFirstInstruction();
        instructionAsString.text = instruction.text;
    }

    public void setInstructionForAction(InstructionForAction
       instructionForAction)
    {
        this.instructionForAction = instructionForAction;
    }

    public void next()
    {
        try
        {
            instruction = instructionForAction.next();
            instructionAsString.text = instruction.text;
            if(instruction.symbole != null)
            {

                aidSymboleGameObject.SetActive(true);
```

```
                aidSymbole.texture = Resources.Load<Texture2D>("Icons/" +
                    instruction.symbole[0]);
            }
        }
        catch (Exception e)
        {
          Debug.LogError(e);
        }

    }

    public void back()
    {
        try
        {
            instruction = instructionForAction.previous();
            instructionAsString.text = instruction.text;
        }
        catch (Exception e)
        {
          Debug.LogError(e);
        }
    }
}
```

J Fragebogen

Fragebogen zur Masterarbeit „Prototypische Entwicklung einer KI-basierten Erste-Hilfe-App“

Im Folgenden geht es um die Beurteilung der Erste-Hilfe-App anhand eines Fragebogens.

Bitte beachten Sie:

- Dieser Fragebogen dient dazu, die Gebrauchstauglichkeit zu bestimmen.
- Bei der Bewertung geht es um die objektive Beurteilung der App, nicht um eine persönliche Bewertung.
- Füllen Sie den Fragebogen sorgfältig aus und lassen Sie keine Frage aus.

Hinweis zur Beantwortung des Fragebogens:

Im Folgenden erfolgt ein Beispiel. Die Fragen weisen immer dieselbe Form aus.

Beispiel:

Die Software...	---	--	-	-/+	+	++	+++	Die Software...
ist schlecht.	○	○	○	○	○	○	○	ist gut.

In dem Beispiel wird gefragt, wie gut oder schlecht die Software ist. Die Bewertungsskalar reicht von sehr schlecht (---) bis sehr gut (+++). In dem Beispiel hat der Anwender die Software zwar mit gut bewertet, sieht jedoch noch Verbesserungspotential.

Abbildung J.1: Fragebogen, Seite 1

Bitte kreuzen Sie an, was auf Sie zutrifft			
Geschlecht	□ männlich	□ weiblich	□ divers
Alter	□ unter 18 Jahre □ 18 bis 24 Jahre □ 25 bis 30 Jahre □ 31 bis 36 Jahre □ 37 bis 41 Jahre	□ 42 bis 48 Jahre □ 49 bis 55 Jahre □ 56 bis 68 Jahre □ über 68 Jahre	
Waren Sie schon einmal in einer Ersten-Hilfe-Situation?	□ Ja	□ Nein	

	Stimme überhaupt nicht zu	Stimme nicht zu	Stimme weder zu noch lehne ich ab	Stimme zu	Stimme voll und ganz zu
Gegenüber digitalen Lösungen zur Unterstützung in der Ersten-Hilfe bin ich sehr offen.	○	○	○	○	○

Die App...	---	--	-	-/+	+	++	+++	Die App...
ist nicht sinnvoll.	○	○	○	○	○	○	○	ist sinnvoll.
werde ich nicht benutzen.	○	○	○	○	○	○	○	werde ich benutzen.
ist nicht leicht zu bedienen.	○	○	○	○	○	○	○	ist leicht zu bedienen.
bietet keine gute Performance.	○	○	○	○	○	○	○	bietet eine gute Performance.
ist schwer ohne fremde Hilfe oder Handbuch erlernbar.	○	○	○	○	○	○	○	ist leicht ohne fremde Hilfe oder Handbuch erlernbar.
hat keine übersichtliche Oberfläche.	○	○	○	○	○	○	○	hat eine übersichtliche Oberfläche.
hat mir nicht gut in der Ersten-Hilfe-Situation geholfen.	○	○	○	○	○	○	○	hat mir gut in der Ersten-Hilfe-Situation geholfen.
hat es mir nicht leicht gemacht, den Wundverlauf zu erstellen.	○	○	○	○	○	○	○	hat es mir leicht gemacht, den Wundverlauf zu erstellen.
hat das Körperteil nicht richtig erkannt.	○	○	○	○	○	○	○	hat das Körperteil richtig erkannt.
hat die Wunde nicht richtig zugeordnet.	○	○	○	○	○	○	○	hat die Wunde richtig zugeordnet.

Abbildung J.2: Fragebogen, Seite 2

Die Handlungs-empfehlungen...	---	--	-	-/+	+	++	+++	Die Handlungs-empfehlungen...
sind nicht klar verständlich.	○	○	○	○	○	○	○	sind klar verständlich.
haben mir nicht geholfen.	○	○	○	○	○	○	○	haben mir geholfen.
enthalten nicht genügend Informationen für die Behandlung .	○	○	○	○	○	○	○	enthalten genügend Informationen für die Behandlung.

Das Pflaster...	---	--	-	-/+	+	++	+++	Das Pflaster ...
ist nicht sinnvoll.	○	○	○	○	○	○	○	ist sinnvoll.
ist nicht leicht auf die Wunde anzubringen.	○	○	○	○	○	○	○	ist leicht auf die Wunde anzubringen.

Die folgenden Fragen können Sie optional ausfüllen:

Was fanden Sie besonders gut oder besonders schlecht?

Haben Sie Vorschläge für Verbesserungen oder Erweiterungen?

Abbildung J.3: Fragebogen, Seite 3

Kolophon

Dieses Dokument wurde mit der LaTeX-Vorlage für Abschlussarbeiten an der htw saar im Bereich Informatik/Mechatronik-Sensortechnik erstellt (Version 2.1). Die Vorlage wurde von Yves Hary und André Miede entwickelt (mit freundlicher Unterstützung von Thomas Kretschmer, Helmut G. Folz und Martina Lehser). Daten: (F)10.95 – (B)426.79135pt – (H)688.5567pt