
Yoga AI

Yoga Trainer mit Pose Estimation

Stephan Seliner
Alex Koller



Bachelorarbeit

Systemtechnik Fachrichtung Ingenieurinformatik
OST – Ostschweizer Fachhochschule

Referent: Prof. Dr. Christoph Würsch
Korreferent: Nicola Notari, M. Sc.

St. Gallen, den 26.08.2022

Zusammenfassung

Yoga ist eine uralte Übungsmethode, ursprünglich aus dem indischem Raum. Heute ist diese praktische Lebensphilosophie für ihre vielen spirituellen, körperlichen und geistigen Vorteile weltweit bekannt. Unser Ziel ist die Entwicklung eines virtuellen Yoga-Trainiers auf Basis der Schätzung der Körperhaltung mittels KI (Pose Estimation). Die Desktop-Applikation kann zuhause und zu jeder Zeit zum Erlernen, Trainieren, Auswerten und Aufzeichnen von Yoga-Asanas verwendet werden.

Bei der Pose Estimation geht es um die Lokalisierung menschlicher Gelenke in einem Bild oder Video, um eine Skelettdarstellung zu erstellen. Dies ist eine herausfordernde Aufgabe, da sie von einer Reihe von Aspekten wie Massstab und Auflösung des Bildes, Beleuchtungsvariationen, Hintergrundstörungen, Kleidungsvariationen und der Umgebung abhängt. Die Yoga-Posen beinhalten komplexe Körperhaltungen und sind somit bestens geeignet für diese Technologie.

Um eine Yoga Pose klassifizieren zu können, wird zuerst mittels MediaPipe eine Schätzung der Körperhaltung durchgeführt. MediaPipe ist eine Machine Learning Pipeline für die Posenschätzung, welche von Google entwickelt wurde. Es werden 33 Merkmale (Keypoints) erkannt und durch x, y und z Koordinate sowie einer Kennzahl für die Sichtbarkeit definiert.

Für die eigentliche Klassifizierung wird ein weiterer Machine Learning Algorithmus verwendet: der Random Forest Classifier. Random Forests sind überwachte Lernmethoden, welche für Klassifizierungsaufgaben eingesetzt werden. Ein Entscheidungsbaum benötigt nur einen Teil der Keypoints-Daten als Input, um die Wahrscheinlichkeit für die verschiedenen Posen vorauszusagen. Der Algorithmus an sich besteht aus mehreren solcher Entscheidungsbäumen und generiert die Vorhersage entsprechend der Vorhersagen der einzelnen Entscheidungsbäume.

In der Applikation kann ein Benutzer eine neue Yoga Pose erlernen, eine Trainingseinheit bestehend aus verschiedenen Posen absolvieren und die Performance absolvieter Trainings analysieren. Im Lernbereich wird zuerst eine Übersicht und Schritt für Schritt-Anleitung angezeigt bevor die Übung startet. Während der Übung wird ein Timer gestartet, sobald man sich in der richtigen Pose befindet. Wird die Pose vom Benutzer nicht exakt aufgeführt, wird ein visuelles Feedback generiert, welches anzeigt, in welche Richtung, die nicht perfekt positionierten Gelenke bewegt werden sollten. Am Ende jeder Übung erhält der Benutzer ein Feedback zur erzielten Performance anhand von drei verschiedenen Scores.

- Den Zeitscore, welcher die benötigte Zeit repräsentiert
- Den Genauigkeitscore, welcher aussagt wie exakt die Pose ausgeführt wurde
- Den Bewegungscore, welcher aussagt wie ruhig die Pose gehalten wurde

Im Trainingsmodus werden die Daten zur Performance aufgezeichnet und stehen im Statistikbereich zur weiteren Analyse bereit.

Abstract

Yoga is an ancient method of exercise, originally from the Indian region. Today, this practical philosophy of life is known worldwide for its many spiritual, physical and mental benefits. Our goal is the development of a virtual Yoga trainer based on human pose estimation that can be used at home at any time to learn and train new Yoga poses, record and evaluate them. The desktop application uses a camera and a flat screen.

Human pose estimation involves locating human joints in an image or video to create a skeletal representation. Automatic detection of a person's pose is a demanding task as it depends on a number of aspects such as scale and resolution of the image, illumination variations, background clutter, clothing variations, environment and interaction of the person with the environment. The yoga poses involve intricate postures, making them ideally suited for this technology.

In order to be able to classify a yoga pose, an estimation of the body pose is first performed using MediaPipe. MediaPipe is a machine learning pipeline for pose estimation developed by Google. It detects 33 features (keypoints) and defines them by x, y and z coordinate as well as a visibility metric.

For the actual classification, another machine learning algorithm is used: the Random Forest Classifier. Random forests are supervised learning methods which are used for classification tasks. Such a decision tree needs only a part of the keypoints data as input to predict the probability for the different poses. The algorithm itself consists of several such decision trees and generates the prediction according to the predictions of the individual decision trees as an ensemble.

In the application, a user can learn a new yoga pose, complete a training session consisting of different poses and analyze the performance of completed workouts. In the learning area, an overview and step-by-step instructions are first displayed before the exercise starts. During the exercise, a timer is started as soon as the user is in the correct pose. If the pose is not performed accurately by the user, visual feedback is generated indicating in which direction the imperfectly positioned joints should be moved. At the end of each exercise, the user receives feedback on the performance achieved based on three different scores.

- The time score, which represents the time taken
- The accuracy score, which tells how accurately the pose was executed
- The motion score, which represents how smoothly the pose was held

In training mode, the performance data is recorded and is available in the statistics area for further analysis.

Inhaltsverzeichnis

Abbildungsverzeichnis	V
Tabellenverzeichnis	VI
Listings	VI
1 Einleitung	1
1.1 Motivation	1
1.2 Aufgabenstellung und Zielsetzung	1
2 Posenschätzung	2
2.1 Klassische Ansätze	2
2.2 Ansätze mit Deep Learning	4
2.2.1 DeepPose	5
2.2.2 Coarse Heatmap Regression Model	6
2.2.3 Convolutional Pose Machine	6
3 Framework	8
3.1 MediaPipe	8
3.1.1 BazePose	9
3.2 TensorFlow	11
3.2.1 MoveNet	11
3.3 TensorFlowJS Pose-Detection API	12
3.4 Entscheid	13
4 Posen Klassifikation	14
4.1 Ansätze zur Klassifizierung	14
4.2 Vergleich verschiedener Klassifizierer	15
5 Anforderungsanalyse	17
5.1 Funktionale Anforderungen	17
5.2 Benutzerfunktionen der Anwendung	17
5.2.1 Lernen neuer Posen	18
5.2.2 Trainieren mit mehreren Posen	18
5.2.3 Darstellen des Trainingsverlaufs	18
5.3 Erkennung der Yoga Asanas	19
6 Konzeption und Design von Yoga AI	20
6.1 Use Cases	20
6.2 Klassendiagramm	21
6.3 User Interface Konzept	22
6.4 Posenerkennung und Erzeugung von Feedback	24

7 Umsetzung und Implementierung	26
7.1 Modelldaten Beschaffung	26
7.2 Erzeugung eines Klassifizierers	27
7.2.1 Hyperparameter-Tuning	28
7.3 Entwicklung des User Interfaces	30
7.3.1 PySide	31
7.3.2 Qt Designer	31
7.3.3 Layout	31
7.3.4 QSettings	32
7.3.5 Qt Ressourcen System	33
7.4 Implementierung der Posenerkennung	33
7.4.1 Implementierung des Feedbacks	35
7.5 Implementierung des Trainingssets	37
7.6 Speicherung und Auswertung der Daten	38
7.6.1 Berechnung der Scores	38
7.6.2 Datenbetrachtung in der Applikation	39
7.7 Bereitstellung der Applikation	40
7.7.1 Setup und Kompilieren	40
7.7.2 Erstellung eines Installationsprogramms	41
7.8 Übersicht über die Module	42
7.8.1 GUI Package	42
7.8.2 Pose Package	42
7.8.3 Statistics Package	42
7.8.4 Trainer Package	42
8 Evaluation	43
8.1 Quantitative Auswertung	43
9 Fazit und Ausblick	45
Literaturverzeichnis	48
Abkürzungsverzeichnis	49
Anhang A: Klassendiagramm	50
Anhang B: Code Ausschnitte	51
Anhang C: Fragebogen	55
Anhang D: Auswertung Fragebogen	56
Eidesstattliche Erklärung	57

Abbildungsverzeichnis

1	Zwei Beispiele mit klassischem Ansatz, Strukturmodelle [5]	3
2	Heatmap-Regression, a) Originalbild b) erzeugte Heatmap c) Erkennungsergebnis [8]	5
3	Links: Darstellung der Deep Neural Network (DNN)-basierten Posenregression. Rechts: im Stadium wird ein verfeinernder Regressor auf ein Teilbild angewendet. [6]	5
4	Übersicht der kaskadierten Architektur des Coarse Heatmap Regression Models [9]	6
5	Architektur einer Convolutional Pose Machine (CPM) [10]	7
6	MediaPipe Komponenten, Beispielgrafik für Objekterkennung [15]	9
7	Übersicht der Pipeline [1]	9
8	Vitruvianischer Mann, der anhand von zwei virtuellen Schlüsselpunkten ausgerichtet wird. [1]	10
9	Die 33 Keypoints von BlazePose [1]	10
10	Prozessablauf von MoveNet [18]	12
11	Vergleich verschiedener Klassifizierer	16
12	Use-Case Diagramm zur Anforderungsanalyse	17
13	Asanas im Yoga82 Datensatz [24]	19
14	Use Case Diagramm	20
15	Reduziertes Klassendiagramm	21
16	Konzept der Startseite	22
17	Konzept der Yoga Seite	23
18	Konzept des Statistikbereichs	23
19	Verwendete Keypoints [1]	27
20	Classification Report	28
21	Vergleich Standard und Bestes Model des Random Forest Classifier, Linker Balken repräsentiert die Bearbeitungszeit und der Rechte die Genauigkeit	30
22	Lernbereich mit Erklärvideo und Schritt-für-Schritt Anleitung	32
23	Trainingsbereich	32
24	Ablauf von Detektion bis zum Feedback	33
25	Verteilung der hinterlegten Winkel für die ideal Pose, Die vertikalen Linien kennzeichnen die drei Genauigkeitsbereiche	35
26	Feedback	36
27	Ablaufschema beim Training	37
28	Statistikbereich	40
29	Auswertung Nutzerumfrage	44

Tabellenverzeichnis

1	Auflistung der bewerteten Frameworks	13
---	--	----

Listings

1	Hyperparameter Tuning	29
2	Beste Parameter für das Modell	30
3	Berechnung der Scores	38
4	Ausschnitt aus Setup Skript	41
5	Mainloop von Yoga AI	51
6	Funktion zur Klassifizierung einer Yoga-Pose	52
7	Erstellung des Säulendiagramms der Sternebewertung	53
8	Handling verschiedener Posen im Training	54

1 Einleitung

1.1 Motivation

Die Schätzung der menschlichen Pose ist ein anspruchsvolles Problem in der Disziplin der Computer Vision. Dabei geht es um die Lokalisierung menschlicher Gelenke in einem Bild oder Video, zur Erstellung einer Skelettdarstellung. Die automatische Erkennung der Pose einer Person in einem Bild ist eine schwierige Aufgabe, da sie von einer Reihe von Aspekten wie Massstab und Auflösung des Bildes, Beleuchtungsvariationen, Hintergrundstörungen, Kleidungsvariationen, Umgebung und Interaktion des Menschen mit der Umgebung abhängt. Eine Anwendung der Pose-Schätzung, auch als Pose Estimation bekannt, ist Sport und Fitness. Eine Form der Übung mit komplizierten Körperhaltungen ist Yoga, eine uralte Trainingsmethode, die in Indien ihren Ursprung hat, heute aber wegen ihrer vielen spirituellen, körperlichen und geistigen Vorteile weltweit bekannt ist.

Das Problem beim Yoga ist jedoch, dass es, wie bei jeder anderen Sportart auch, von grösster Wichtigkeit ist, sie richtig zu praktizieren, da jede falsche Haltung während einer Yoga-Sitzung unproduktiv und möglicherweise schädlich sein kann. Daher ist es notwendig, dass ein Lehrer die Sitzung überwacht und die Haltung des Einzelnen korrigiert.

Da nicht alle Zugang zu einem Yogalehrer haben oder über entsprechende Ressourcen verfügen, könnte eine auf künstlicher Intelligenz basierende Anwendung eingesetzt werden, um Yogastellungen zu erkennen und personalisiertes Feedback zu geben, damit der Einzelne seine Form verbessern kann.

1.2 Aufgabenstellung und Zielsetzung

Ziel der Bachelorarbeit ist es, basierend auf der Google-Implementierung MediaPipe BlazePose [1], einen virtuellen Yoga-Trainer zu programmieren und zu testen.

Einleitend soll mit Hilfe einer Literaturrecherche der aktuelle Stand der Technik der Pose Estimation aufgezeigt werden. Mit dem Yoga-Trainer soll ein Yoga-Anfänger eine neue Asana, ruhende Körperstellung im Yoga, interaktiv erlernen können. Weiter sollen Trainings mit mehreren Übungen möglich sein, wobei der Benutzer ebenfalls in Echtzeit ein Feedback erhalten soll. Zusätzlich sollen die absolvierten Asanas passend beurteilt werden und diese Bewertung soll auch im Zeitverlauf analysiert werden können. Abschliessend soll die Anwendung in einem Selbstversuch getestet und analysiert werden.

2 Posenschätzung

2 Posenschätzung

Pose Estimation ist die Aufgabe, mithilfe eines Machine Learning (ML) Modells die Pose einer Person aus einem Bild oder einem Video zu schätzen, indem die räumlichen Positionen der wichtigsten Körpergelenke (Keypoints) geschätzt werden.

Pose Estimation bezieht sich auf Computer-Vision-Techniken, die menschliche Figuren in Bildern und Videos erkennen, um beispielsweise festzustellen, wo der Ellenbogen einer Person in einem Bild zu sehen ist. Es ist wichtig, sich der Tatsache bewusst zu sein, dass die Posenschätzung lediglich schätzt, wo sich die wichtigsten Körpergelenke befinden, und nicht erkennt, wer sich in einem Bild oder Video befindet.

Die Modelle zur Posenschätzung verwenden ein verarbeitetes Kamerabild als Eingabe und geben Informationen über Keypoints aus. Die erkannten Keypoints werden, abhängig ob 2D oder 3D Pose Estimation angewendet wird, mit den entsprechenden Koordinaten und mit einem Konfidenzwert zwischen 0 und 1 ausgegeben. Der Konfidenzwert gibt die Wahrscheinlichkeit an, dass an dieser Position der Keypoint vorhanden ist. So würde ein Konfidenzwert von 1 einer 100% Sicherheit entsprechen.

Diese Ansätze zur Schätzung der Körperhaltung lassen sich in Bottom-up- und Top-down-Methoden unterteilen [2].

Bei Bottom-up-Methoden wird zunächst jedes einzelne Körpergelenk geschätzt und dann zu einer eindeutigen Pose gruppiert. Top-down-Methoden führen zunächst einen Personendetektor aus und schätzen die Körpergelenke innerhalb der erkannten Bounding Boxes.

2.1 Klassische Ansätze

Klassische Ansätze beziehen sich in der Regel auf Techniken und Methoden, die Algorithmen des maschinellen Lernens verwenden. Die meisten dieser Modelle funktionieren gut, wenn das Eingabebild eindeutige und sichtbare Körperteile aufweist. Sie versagen jedoch bei der Erfassung und Modellierung von Körperteilen, die verborgen oder aus einem bestimmten Winkel nicht sichtbar sind. Um diese Probleme zu überwinden, wurden Methoden zur Merkmalsbildung wie Histogrammorientierte Gradienten (HOG), Konturen, Histogramme usw. verwendet [3]. Trotz des Einsatzes dieser Methoden mangelt es den klassischen Modellen an Genauigkeit, Korrelation und Verallgemeinerungsfähigkeit.

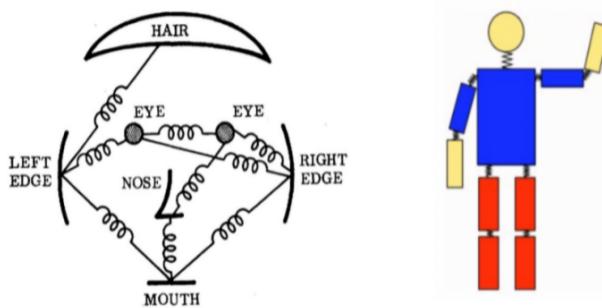
Die klassischen Ansätze lassen sich in die Kategorien Erscheinungs- und Strukturmodelle einteilen.

Bei den *Erscheinungsmodellen* spielen diese erwähnten Methoden zu Merkmalsbildung eine zentrale Rolle. Poselets nutzen genau diesen Ansatz [4]. Sie sind darauf trainiert, auf einen gewissen Ausschnitt des Objekts, unter einem bestimmten Blickwinkel und Pose zu reagieren. Es gibt eine Vielzahl von Poselets – ein Frontalgesicht, ein Profilgesicht, eine Kopf-Schulter-Konfiguration und viele Weitere.

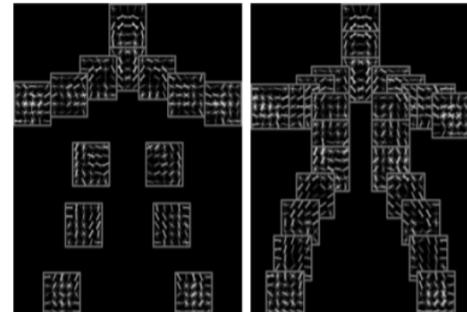
2 Posenschätzung

Bei den *Strukturmodellen* ist das „Pictorial Structures Model“ ein weit verbreitetes Modell [5]. Dieses modelliert die räumlichen Korrelationen starrer Körperteile, indem diese in Form eines baumstrukturierten grafischen Modells ausgedrückt werden, um die Lage der Körpergelenke vorherzusagen. Diese räumlichen Verbindungen werden durch die Verwendung von Federn dargestellt, und die Körperteile sind Erscheinungsbildvorlagen, die auf dem Bild basieren. Durch die Parametrisierung der Körperteile mittels Pixelposition und -ausrichtung, kann die resultierende Struktur die Artikulation modellieren.

Ein weiteres Modell dieser Kategorie ist das „Flexible Mixture-of Parts“. Bei diesem Ansatz werden verformbare Teilmodelle verwendet, die eine Sammlung von Vorlagen sind, welche in einem Bild gefunden werden und in einer verformbaren Konfiguration angeordnet sind. Außerdem verfügt jedes Modell über globale und partielle Vorlagen. Die Hauptidee ist, die Verwendung einer Mischung aus kleinen, nicht orientierten Körperteilen zu verwenden, anstelle von verformbaren Vorlagen. Die Gründe dafür haben mit der unterschiedlichen Darstellung von Gliedmassen und Änderungen des Blickwinkels zu tun.



(a) Pictorial Structures Model



(b) Flexible Mixture-of Parts

Abbildung 1: Zwei Beispiele mit klassischem Ansatz, Strukturmodelle [5]

2 Posenschätzung

2.2 Ansätze mit Deep Learning

Durch die enorme Entwicklung von Deep Learning Lösungsansätzen in den letzten Jahren wurden die klassischen Methoden in verschiedenen Gebieten, darunter auch Objekterkennung oder Bildersegmentierung, übertroffen [2]. Diese Deep Learning Techniken brachten signifikante Vorteile und Performancesteigerung in Pose Estimation Aufgaben.

Die klassische Pipeline hat ihre Grenzen, und die Posenschätzung wurde durch Convolutional Neural Networks (CNN) stark verändert. Mit der Einführung von DeepPose von Toshev et al. [6], begann sich die Forschung auf dem Gebiet der menschlichen Posenschätzung von klassischen Ansätzen auf Deep Learning zu verlagern. Die meisten neueren Systeme zur Posenschätzung haben durchgängig CNN als Hauptbaustein verwendet und damit handgefertigte Merkmale und grafische Modelle weitgehend ersetzt. Diese Strategie hat zu drastischen Verbesserungen bei Standard-Benchmarks geführt [7].

Bevor einige dieser Deep-Learning Modelle etwas genauer betrachtet werden, wird nachfolgend das Prinzip der Keypoint- und Heatmap-Regression erläutert [8].

Bei der *Keypoint-Regression* extrahiert das Modell die Keypoints direkt aus den feature maps (gefiltertes Eingangsbild durch CNN), weshalb sie in einigen Referenzen als direkte Regression bezeichnet wird. Wenn mit dieser Methode 17 Schlüsselpunkte in 2D für eine Person geschätzt werden möchten, ist die Ausgabe des Modells ein 17×2 -Vektor, der die x- und y-Koordinaten jedes vorhergesagten Keypoints enthält. Viele verschiedene Modelle wurden auf der Grundlage des Keypoint-Regressionsansatzes vorgeschlagen. Eine grosse Schwierigkeit ist jedoch die Empfindlichkeit des Modells, welches zu Instabilität beim Training führt. Diese entsteht, wenn im Modell ein bestimmter Keypoint mit einer Abweichung von wenigen Pixeln von der Grundwahrheit vorhergesagt wird. Dies stört den Trainingsprozess und hindert die Konvergenz des Modells zu einer optimalen Lösung.

Um das Problem der Empfindlichkeit und Instabilität zu lösen, wurde die *Heatmap-Regression* als alternativer Ansatz eingeführt. Im Gegensatz zur vorherigen Methode, bei der die genaue Position jedes Keypoints direkt ermittelt wird, schätzt dieser Ansatz die Wahrscheinlichkeit der Existenz eines Keypoints in jedem Pixel des Bildes. Dieses Modell ist sehr erfolgreich, und viele Arbeiten verwenden Heatmaps anstelle einer direkten Regression. Bei der Umsetzung dieser Methode werden geringfügige Unterschiede bei der Vorhersage von Keypoints ausser Acht gelassen und das Modell kann entspannter trainiert werden.

2 Posenschätzung

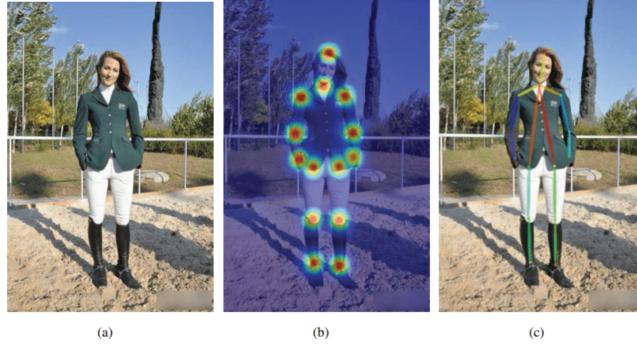


Abbildung 2: Heatmap-Regression, a) Originalbild b) erzeugte Heatmap c)
Erkennungsergebnis [8]

2.2.1 DeepPose

DeepPose [6] war die erste grössere Arbeit, in der Deep Learning auf die Schätzung der menschlichen Körperhaltung angewendet wurde. Es erreichte eine solide Performance und übertraf bestehende Modelle. Bei diesem Ansatz wird die Posenschätzung als CNN-basiertes Regressionsproblem für Körpergelenke formuliert. Es wird eine Kaskade solcher Regressoren verwendet, um die Posenschätzungen zu verfeinern und bessere Schätzungen zu erhalten. Ein wichtiger Aspekt dieses Ansatzes ist die ganzheitliche Betrachtung der Pose. Dies ermöglicht es selbst verborgene Gelenke zu schätzen. Die Verwendung von CNNs erlaubt es diese Art von Schlussfolgerungen auf natürliche Weise zu erhalten und überzeugende Ergebnisse zeigen.

Eine interessante Idee, die dieses Modell umsetzt, ist die Verfeinerung der Vorhersagen durch *kaskadierte Regressoren*. Die anfängliche grobe Pose wird verfeinert und eine bessere Schätzung wird erreicht. Die Bilder werden um das vorhergesagte Gelenk herum beschnitten und der nächsten Stufe zugeführt. Auf diese Weise sehen die nachfolgenden Posenregressoren Bilder mit höherer Auflösung und lernen so Merkmale für feinere Massstäbe, was letztlich zu einer höheren Präzision führt. Dieses Modell wird mit der Verlustfunktion $L2$ für die Regression, also dem quadratischen Fehler, trainiert.



Abbildung 3: Links: Darstellung der DNN-basierten Posenregression. Rechts: im
Stadium s wird ein verfeinernder Regressor auf ein Teilbild angewendet. [6]

2 Posenschätzung

2.2.2 Coarse Heatmap Regression Model

Beim Coarse Heatmap Regression Model [9] werden Heatmaps erzeugt, indem ein Bild parallel durch mehrere Auflösungsblocks läuft, um gleichzeitig Merkmale in verschiedenen Massstäben zu erfassen. Das Ergebnis ist eine diskrete Heatmap anstelle einer kontinuierlichen Regression.

Bei diesem Modell wird eine CNN-Architektur mit mehreren Auflösungen zur Implementierung eines Schiebefenster-Detektors verwendet, um eine grobe Heatmap-Ausgabe zu erzeugen. Die Hauptmotivation besteht darin, die räumliche Genauigkeit wiederherzustellen, die durch das Pooling im Ausgangsmodell verloren gegangen ist. Zu diesem Zweck wird ein zusätzliches CNN zur Verfeinerung der Pose verwendet, dass das Lokalisierungsergebnis der groben Heatmap verfeinert. Im Gegensatz zu einer Standardkaskade von Modellen verwenden diese Modelle jedoch vorhandene Faltungsmerkmale wieder. Dies reduziert nicht nur die Anzahl der trainierbaren Parameter in der Kaskade, sondern wirkt auch als Regularisierer für das grobe Heatmap-Modell, da das grobe und das feine Modell gemeinsam trainiert werden.

Im Wesentlichen besteht das Modell aus dem Heatmap basierten Teilmodell für die grobe Lokalisierung, einem Modul zum Abtasten und Zuschneiden der Faltungsmerkmale bei einem bestimmten (x, y) für jedes Gelenk, sowie ein zusätzliches Faltungsmodell für die Feinabstimmung. Ein entscheidendes Merkmal dieser Methode ist die gemeinsame Verwendung eines CNN und eines grafischen Modells. Das grafische Modell lernt typische räumliche Beziehungen zwischen Gelenken.

Dieses Modell wird trainiert, indem der Mean Squared Error (MSE) zwischen der vorhergesagten Heatmap und einer Ausgangs-Heatmap minimiert wird.

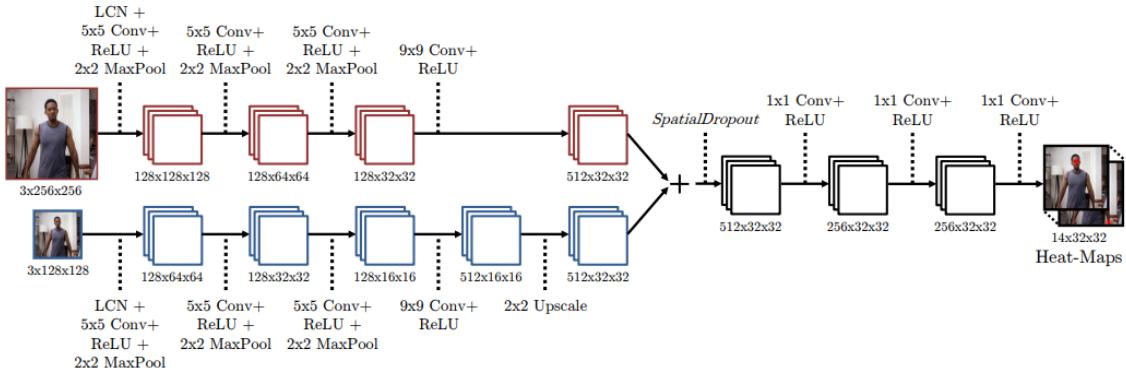


Abbildung 4: Übersicht der kaskadierten Architektur des Coarse Heatmap Regression Models [9]

2.2.3 Convolutional Pose Machine

Eine Pose-Maschine [10] besteht aus einer Folge von Mehrklassen-Prädiktoren, die trainiert werden, um die Lage jedes Körperteils in jeder Hierarchiestufe vorherzusagen.

2 Posenschätzung

Dieses Modell besteht aus einem Modul zur Berechnung von Bildmerkmalen, gefolgt von einem Vorhersagemodul. Diese CPMs sind vollständig differenzierbar und ihre mehrstufige Architektur kann durchgängig trainiert werden. Sie bieten einen sequenziellen Vorhersagerahmen für das Erlernen umfangreicher impliziter räumlicher Modelle und funktionieren sehr gut für die menschliche Pose.

Stufe 1 ist das Modul zur Berechnung von Bildmerkmalen, und Stufe 2 ist das Modul zur Vorhersage. Die Anzahl der Stufen ist ein Hyperparameter, normalerweise drei, und ab der zweiten Phase sind dies Wiederholungen von der Stufe zwei. Dieses sequentielle Vorhersagekonzept, das aus Faltungsnetzen besteht und implizite räumliche Modelle erlernt, nutzt grössere rezeptive Felder auf den belief maps (eine Art Heatmap) der vorherigen Phasen, was das Erlernen der räumlichen Beziehungen zwischen den Körperteilen unterstützt und zu einer verbesserten Genauigkeit durch zunehmend verfeinerte Schätzungen der Positionen in den späteren Phasen führt [5]. Nach jeder Stufe wird eine Zwischenüberwachung verwendet, um die Problematik der verschwindenden Gradienten zu vermeiden, was ein häufiges Problem bei tiefen mehrstufigen Netzen ist.

Die Abbildung 5 zeigt die Architektur einer CPM mit beliebigen T Stufen. Die Pose-Maschine ist in den Einschüben (a) und (b) dargestellt, und die entsprechenden Faltungsnetzwerke sind in den Einschüben (c) und (d) zu sehen. Einschübe (a) und (c) zeigen die Architektur, die in der ersten Stufe nur mit Bildinformationen arbeitet. Die Einschübe (b) und (d) zeigen die Architektur für die nachfolgenden Stufen, die sowohl auf Bildevidenz als auch auf belief maps aus den vorangegangenen Stufen arbeiten. Die Architekturen in (b) und (d) werden für alle nachfolgenden Stufen (2 bis T) wiederholt. Von den drei vorgestellten Modellen hat dieses die höchste Genauigkeit.

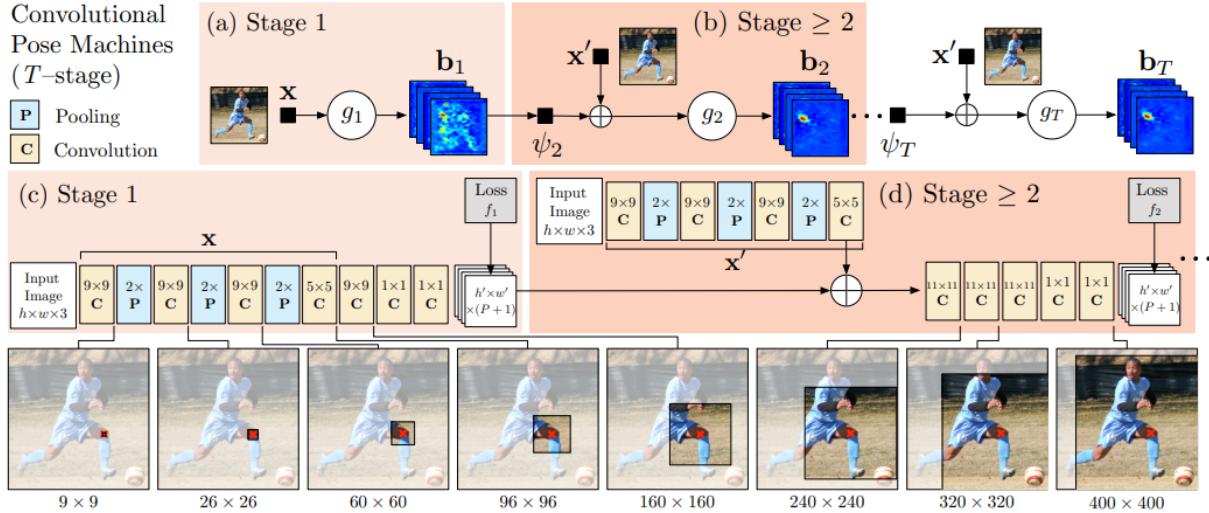


Abbildung 5: Architektur einer CPM [10]

3 Framework

Ein geeignetes Framework für die Pose Estimation zu finden war ein zentraler Aspekt dieser Arbeit. Zum einen sollte die Pipeline-Verarbeitung in Quasi-Echtzeit erfolgen können, und ausserdem muss dies auch auf einem gewöhnlichen Windows-Rechner ausführbar sein. Dafür wurden die Frameworks von Mediapipe und TensorFlow genauer untersucht.

3.1 MediaPipe

MediaPipe ist ein Open-Source-Framework von Google zum Aufbau von Anwendungen im Bereich der Pose Estimation. Der Code ist in C++ geschrieben und kann leicht auf den meisten Plattform eingesetzt werden [11]. Die grosse Stärke ist die Geschwindigkeit, welche durch Graphic Processing Unit (GPU)-Beschleunigung, jedoch nicht auf allen Plattformen unterstützt, und Multithreading erreicht wird [12]. Dies zahlt sich vor allem bei neuen Smartphones aus, welche hohe Frames Per Second (FPS)-Werte erreichen.

Jedoch kann der Mangel an Dokumentation oftmals viel Zeit in Anspruch nehmen. Die "Dokumentation" von MediaPipe besteht aus einer Website, auf der die Konzepte auf hohem Niveau erläutert werden, und einfachen Codebeispielen. Um MediaPipe wirklich zu verstehen, muss man schon fast in den Quellcode eintauchen. Dort hat es am Anfang vieler .cc-Dateien Kommentare die erklären was im Skript genau passiert. MediaPipe ist momentan noch in der Alpha (V0.7), was die fehlende Dokumentation erklärt. Viele Funktionen können noch geändert werden und deshalb ist wahrscheinlich auch noch keine endgültige Dokumentation vorhanden. Laut der GitHub-Seite [13] erwartet man noch Änderungen der API und versucht deshalb mit der v1.0 zu einer stabilen API zu kommen.

Die Blackbox MediaPipe nutzt die Komponenten Pakete, Graphen, Rechnermodule und Untergraphen zur Verarbeitung des Inputs [14]. Pakete sind einfach jede Datenstruktur mit einem Zeitstempel. In der folgenden Abbildung 6 ist „input_video“ das Paket, das in den Graphen eingespeist wird.

Ein Graph ist die Struktur des gesamten Programms. Das gesamte Diagramm selbst ist ein Graph namens „Object Detection“.

Rechnermodule können Ein- sowie Ausgaben haben und erledigen Rechenoperationen. Sie führen Code bei der Erstellung, pro Frame und beim Beenden aus. Ein Beispiel für einen Rechner könnte einer sein, der die Koordinaten von Objekten einliest und diese Koordinaten normalisiert ausgibt.

Ein Untergraph besteht aus einer Reihe von Rechnermodulen, die in einem Graphen gruppiert sind. Untergraphen haben definierte Eingänge und Ausgänge und helfen bei der Abstraktion dessen, was sonst ein zu kompliziertes Diagramm wäre. Die Untergraphen im Diagramm sind blau und lauten in diesem Fall: „ObjectDetection“, „Object-Tracking“ und „Renderer“.

3 Framework

MediaPipe stellt viele Lösungsansätze, sogenannte *Solutions* zur Verfügung, welche diese Komponenten nutzen. Bei intensiver Einarbeit, können dann auch individuelle Änderungen vorgenommen werden. MediaPipe empfiehlt die vorgefertigten Solutions zu verwenden und nur in Ausnahmefällen Änderungen selbst vorzunehmen.

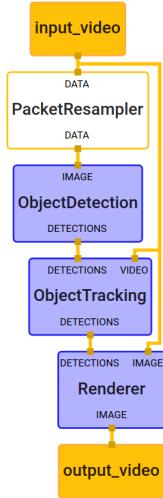


Abbildung 6: MediaPipe Komponenten, Beispielgrafik für Objekterkennung [15]

3.1.1 BlazePose

BlazePose ist eine performante Solution zur Pose-Estimation [1]. Es gibt ein heavy, full und lite Modell von welchen das heavy Model die höchste Genauigkeit erreicht. Mit BlazePose [1] verbessert MediaPipe den aktuellen Standard, die COCO-Topologie, welche aus 17 menschlichen Körper-Keypunkten besteht, auf 33. Diese Leistung ist gerade für praktische Anwendungen wie Fitness und Tanz von grosser Bedeutung.

Für die Pose Estimation wird eine zwei-Schritt Detector-Tracker ML-Pipeline verwendet. Der Detektor lokalisiert zuerst die Pose Region of Interest (ROI) innerhalb des Bildes. Anschliessend sagt der Tracker alle 33 Pose-Keypoints aus dieser ROI voraus. Bei Videoanwendungen wird der Detektor nur für das erste Bild ausgeführt. Für nachfolgende Frames werden die ROI aus den Pose-Keypoints des vorherigen Frames abgeleitet.

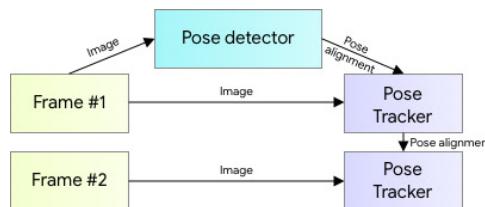


Abbildung 7: Übersicht der Pipeline [1]

3 Framework

Um die ML-Pipeline in Echtzeit zu durchlaufen, darf jede Komponente nur wenige Millisekunden pro Frame benötigen. Dies wird erreicht indem das Gesicht einer Person genutzt wird, um die Position des Oberkörpers zu bestimmen. Natürlich bedeutet dies, dass der Kopf einer Person sichtbar sein sollte. Folglich wurde ein Gesichtsdetektor trainiert, der sich am BlazeFace-Modell orientiert, dass im Sub-Millisekundenbereich liegt und als Stellvertreter für einen Pose-Detektor dient. Dieses Modell erkennt die Position einer Person innerhalb eines Bildes, kann aber nicht mehrere Personen identifizieren. Bei diesem Pose-Tracking werden explizit zwei zusätzliche virtuelle Keypoints vorausgesagt, die das Zentrum, die Rotation und den Massstab des menschlichen Körpers als Kreis fest beschreiben. Inspiriert von Leonardos Vitruvianischem Mann wird der Mittelpunkt der Hüfte einer Person, der Radius eines Kreises, der die gesamte Person umschreibt, und der Neigungswinkel der Linie, welche die Schulter- und Hüftmittelpunkte verbindet, vorausgesagt. Dies führt zu einer konsistenten Verfolgung, selbst bei sehr komplizierten Fällen, wie zum Beispiel bei bestimmten Yoga-Asanas.

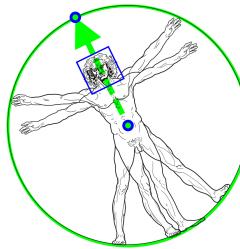


Abbildung 8: Vitruvianischer Mann, der anhand von zwei virtuellen Schlüsselpunkten ausgerichtet wird. [1]

Die Posenschätzungs komponente der Pipeline sagt die Position aller 33 Personen-Keypoints mit jeweils drei Freiheitsgraden (x-, y-Position und Sichtbarkeit), sowie der beiden oben beschriebenen virtuellen Ausrichtungs-Keypoints voraus [16]. Diese Daten sind normalisiert und liegen zwischen 0 und 1. Im Gegensatz zu aktuellen Ansätzen, die eine rechenintensive Heatmap-Vorhersage verwenden, nutzt dieses Modell einen kombinierten Ansatz aus Heatmap, Offset und Regression.

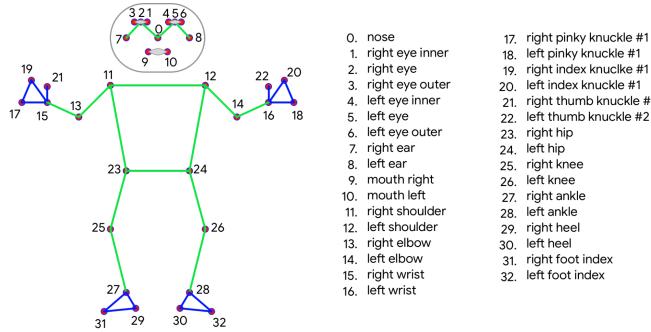


Abbildung 9: Die 33 Keypoints von BlazePose [1]

3.2 TensorFlow

TensorFlow ist ein maschinelles Lernsystem, das im grossem Massstab und in heterogenen Umgebungen arbeitet [17]. Mit TensorFlow können Entwickler Datenflussgraphen erstellen. Diese Strukturen beschreiben, wie sich Daten durch einen Graphen oder eine Reihe von Verarbeitungsknoten bewegen. Jeder Knoten im Graphen repräsentiert eine mathematische Operation, und jede Verbindung oder Kante zwischen Knoten ist ein mehrdimensionales Datenfeld, ein sogenannter Tensor.

TensorFlow stellt all dies dem Programmierer mit Hilfe der Sprache Python zur Verfügung. Python ist einfach zu erlernen und zu verwenden, ausserdem bietet es bequeme Wege, um auszudrücken, wie High-Level-Abstraktionen miteinander verbunden werden können. Knoten und Tensoren in TensorFlow sind Python-Objekte, und TensorFlow-Anwendungen sind selbst Python-Anwendungen.

Die eigentlichen mathematischen Operationen werden jedoch nicht in Python durchgeführt. Die Bibliotheken von Transformationen, die durch TensorFlow verfügbar sind, sind als hochleistungsfähige C++-Binärprogramme geschrieben. Python leitet nur den Verkehr zwischen den Teilen und bietet High-Level-Programmierabstraktionen, um sie miteinander zu verbinden.

TensorFlow-Anwendungen können auf fast jedem Ziel ausgeführt werden, das sich anbietet: ein lokaler Rechner, ein Cluster in der Cloud, iOS- und Android-Geräte, Central Processing Unit (CPU) oder GPUs. Auf Googles eigener Cloud, kann TensorFlow auf den benutzerdefiniertem TensorFlow Processing Unit (TPU) zur weiteren Beschleunigung ausgeführt werden.

3.2.1 MoveNet

Bei MoveNet handelt es sich um einen Hochgeschwindigkeits-Positionstracker von Google. Das Modell ist vor-trainiert und daher nach der Einrichtung sofort einsatzbereit. Es erfasst insgesamt 17 Keypoints. Diese Keypoints sind mit (x, y)-Koordinaten verknüpft und werden jedes Mal aktualisiert, wenn der Detektor aufgerufen wird. Jeder zurückgegebene (x, y) Keypoint ist mit einer Punktzahl verknüpft, die das Vertrauen von MoveNet in die Genauigkeit der Messung darstellt. Auch bei diesem Modell sind die Werte normalisiert und liegen zwischen 0 und 1.

MoveNet gibt es in zwei Versionen, die unterschiedliche Leistungsmerkmale aufweisen. Lightning ist schneller, kann aber weniger genaue Ergebnisse liefern. Thunder ist etwas langsamer, aber genauer. Laut TensorFlow können beide mit 30+ FPS laufen, welche wir in unseren Tests jedoch auf keinem Gerät erreicht haben. MoveNet ist ein Bottom-up-Schätzungsmodell, das Heatmaps zur genauen Lokalisierung menschlicher Schlüsselpunkte verwendet. Die Architektur besteht aus zwei Komponenten, einem Merkmalsextraktor und einem Satz von Vorhersagekomponenten. Obwohl diese Vorhersagen parallel berechnet werden, kann man einen Einblick in die Funktionsweise des Modells gewinnen, wenn man die folgende Abfolge von Vorgängen betrachtet [18]:

3 Framework

Schritt 1: Die Personenzentrum-Heatmap wird verwendet, um die Zentren aller Individuen im Frame zu identifizieren, definiert als das arithmetische Mittel aller zu einer Person gehörenden Keypoints. Der Standort mit der höchsten Punktzahl, gewichtet nach dem inversen Abstand zum Bildzentrum, wird ausgewählt.

Schritt 2: Ein anfänglicher Satz von Keypoints für die Person wird erstellt, indem die Keypoint-Regressionsausgabe von dem Pixel geschnitten wird, das dem Objektzentrum entspricht. Da es sich hierbei um eine Vorhersage aus der Mitte heraus handelt, die über verschiedene Skalen hinweg funktionieren muss, ist die Qualität der regressierten Keypoints nicht sehr genau.

Schritt 3: Jedes Pixel in der Keypoint-Heatmap wird mit einem Gewicht multipliziert, das umgekehrt proportional zum Abstand zum entsprechenden regressierten Keypoint ist. Auf diese Weise wird sichergestellt, dass keine Keypoints von Personen im Hintergrund akzeptiert werden, da sie sich in der Regel nicht in der Nähe der dieser Keypoints befinden und daher eine niedrige Punktzahl haben werden.

Schritt 4: Der endgültige Satz von Keypoint-Vorhersagen wird durch Abrufen der Koordinaten der maximalen Heatmap-Werte in jedem Keypoint-Kanal ausgewählt. Die lokalen 2D-Offset-Vorhersagen werden dann zu diesen Koordinaten hinzugefügt, um verfeinerte Schätzungen zu erhalten. Die folgende Abbildung veranschaulicht diese vier Schritte.

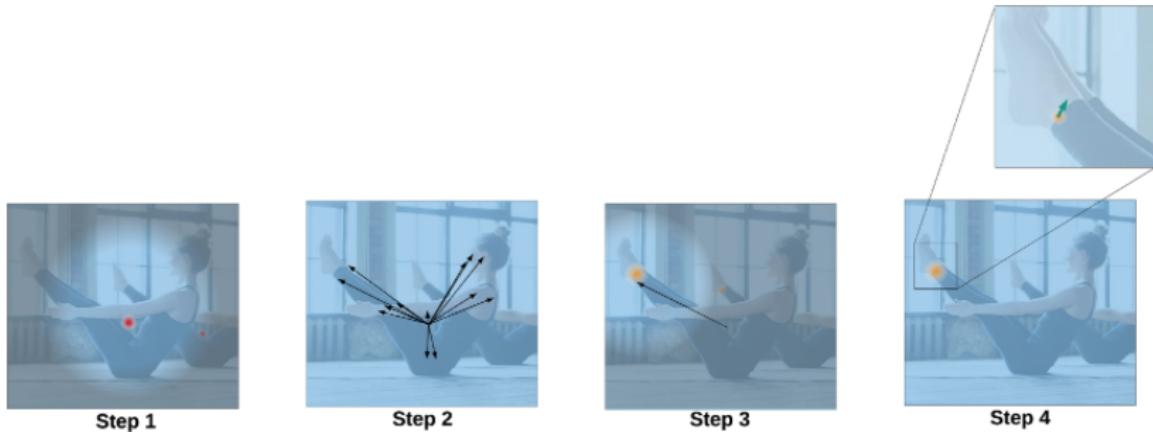


Abbildung 10: Prozessablauf von MoveNet [18]

3.3 TensorFlowJS Pose-Detection API

Ein entscheidender Faktor um möglichst viele FPS zu erreichen, ist die parallele Nutzung der CPU und GPU für die Abarbeitung einer Pipeline. Die Nutzung der GPU ist jedoch bei MediaPipe unter Verwendung von Windows eher experimentell und bei TensorFlow ist die Beschleunigung dadurch nur klein. Eine weitere Möglichkeit diese GPU-Beschleunigung zu erreichen, besteht durch das verschieben der Modelle in den

3 Framework

Browser. Somit kümmert sich dann der Browser um die richtige Konfiguration von Treibern und Bibliotheken. Da JavaScript auch Clientseitig läuft, können diese Modelle unabhängig der Bandbreite, also direkt beim Anwender laufen. Schliesslich verbleiben alle Daten auf dem Client, was die TensorFlowJS API für Inferenzen mit geringer Latenzzeit sowie für Anwendungen zur Wahrung der Privatsphäre nützlich macht.

Die neue TensorFlowJS Pose-Detection API macht dies möglich. Sie unterstützt die zwei Laufzeiten TensorFlowJS und MediaPipe sowie die beiden Modelle BlazePose und MoveNet. TensorFlowJS bietet die Flexibilität und eine breitere Akzeptanz von JavaScript, optimiert für mehrere Backends, einschliesslich Web Graphics Library (WebGL) (GPU), WebAssembly (WASM) (CPU) und Node. MediaPipe macht sich WASM mit GPU-beschleunigter Verarbeitung zunutze und bietet eine schnellere Inferenzgeschwindigkeit. Der MediaPipe-Laufzeitumgebung fehlt derzeit noch die Unterstützung für Node und iOS Safari [18].

Die Nutzung TensorFlowJS Pose-Detection Application Programming Interface (API) würde jedoch auch bedeuten das die gesamte Arbeit sich stark mit JavaScript beschäftigt. Weiter wird auch durch den Einsatz einer Webanwendung, das Vertrauen in den Datenschutz gefährdet.

3.4 Entscheid

Damit die geeignete Plattform für dieses Projekt gefunden werden kann, wurden die beiden Frameworks auf verschiedenen Betriebssystemen mit verschiedenen Backends und Programmiersprachen getestet. Die folgenden drei Punkte galten dabei als Bewertungskriterien:

- **Aufwand Einrichtung:** Aufwand für Endbenutzer bis Programm benutzbar ist
- **Frames Per Second (FPS):** Wie flüssig erscheint die Posenschätzung
- **Modell Schlüsselpunkte:** Wie viele Schlüsselpunkte werden vom Modell erkannt

Da der Einsatz von Linux unseren potentiellen Marktanteil auf nur 2.4% reduzieren würde [19] und uns die Verwendung von Python in der Datenanalyse vieles vereinfachen wird, fiel die Entscheidung auf MediaPipe mit dem Python Framework.

	Aufwand	FPS	Keypoints	Bemerkungen
MediaPipe Python Win10	gering	25	33	Einfachste Implementierung
MediaPipe Python Linux	sehr hoch	–	33	GPU Unterstützung schwierig
MediaPipe JavaScript	–	25	33	JavaScript Kenntnisse
TensorFlowJS	–	120	33	JavaScript Kenntnisse
TensorFlow Python CPU	gering	20	17	FPS identisch zu MediaPipe
TensorFlow Python GPU	mittel	20	17	GPU ohne Vorteil

Tabelle 1: Auflistung der bewerteten Frameworks

4 Posen Klassifikation

Ein zentraler Aspekt dieser Arbeit ist es verschiedene Yoga-Asanas zu erkennen, beziehungsweise zu unterscheiden. Für diese Aufgabe wird ein geeignetes ML-Modell benötigt, dass anhand von Daten, welche die Körperstellung einer Person definieren, die richtige Asana erkennt. Die verschiedenen Posen vorherzusagen ist eine klassische Klassifizierungsaufgabe. Deshalb kommen verschiedene ML-Modelle infrage.

4.1 Ansätze zur Klassifizierung

Die Klassifizierungs-Modelle sind überwachte (supervised) Lernalgorithmen. Anhand von Inputdaten, auch features genannt, soll das Modell als Output eine definierte Zielvariable vorhersagen. Konkret bedeutet dies in dieser Anwendung, dass mit den generierten Keypoints-Daten von MediaPipe die korrekte Pose, also eine nominale Zielvariable, des Benutzers erkannt werden soll. Der Begriff überwacht, kommt daher, dass diese ML-Modelle zuerst mit Daten, welche den Input und Output definiert haben, trainiert werden, um die Parameter des jeweiligen Algorithmus passend einzustellen. Der Workflow zum Erstellen eines ML-Modells wird folgend erläutert [20].

1. Daten sammeln
2. Vorverarbeitung der Daten
3. Geeignete Modelle trainieren
4. Validieren der Modelle
5. Hyperparameter-Tuning und Modell festlegen

Die Performance eines Modells ist sehr stark mit den verwendeten Daten korreliert. Der erste Schritt ist also sehr entscheidend für den weiteren Ablauf. Die Daten sollten in den verschiedenen Kategorien ausgeglichen sein und müssen mit der korrekten Zielvariable gekennzeichnet (label) sein.

Bei der Vorverarbeitung der Daten geht es im Wesentlichen darum, die gesammelten Daten zu untersuchen und zu bereinigen. Dazu gehören Aufgaben wie das Analysieren der verschiedenen Datentypen, das Bereinigen von unerlaubten Werten oder die Auswahl der Prädiktoren. Ebenso werden die Daten in Trainings- und Testdaten unterteilt, um später die Performance eines Modells auf ungewohnten Daten zu testen.

Als nächstes sind geeignete ML-Modelle für die Aufgabe zu definieren und anschließend zu trainieren. Beim Trainieren lernt der Algorithmus eines Modells anhand der features und der Zielvariable, seine Parameter so einzustellen, dass eine möglichst hohe Trefferquote erreicht wird.

Anschließend wird die Performance der verschiedenen Modelle, anhand von geeigneten Klassifizierungsmetriken, auf den Testdaten verglichen.

Jedes Modell besitzt eigene Hyperparameter, welche vom Benutzer angepasst werden können, um eine bessere Performance zu erhalten. Zum Abschluss wird das zu verwendende Modell, anhand von Kriterien wie Performance und/oder Bearbeitungsgeschwindigkeit, festgelegt.

4.2 Vergleich verschiedener Klassifizierer

Für diese Aufgabe wurden fünf verschiedene Klassifizierer, mit den Daten aus Kapitel 7.1, trainiert und anschliessend ausgewertet. Als Input dienen die Koordinaten der Keypoints einer Person beim Ausführen einer Pose und als Output der Name dieser Pose. Dieses Kapitel dient der Übersicht über die berücksichtigten ML-Modelle und in Kapitel 7.2 wird auf die Erzeugung des verwendeten Klassifizierer genauer eingegangen [21].

- **Logistic Regression**

Die logistische Regression verwendet eine Sigmoid-Funktion um die Wahrscheinlichkeit einer Kennzeichnung zu ermitteln. Die Sigmoidfunktion erzeugt eine Wahrscheinlichkeitsausgabe. Durch den Vergleich der Wahrscheinlichkeit mit einem vordefinierten Schwellenwert wird das Objekt einem entsprechenden Label zugeordnet.

- **K-Nearest Neighbours Classifier**

Beim k-Nächster-Nachbar-Algorithmus (kNN) wird jeder Datenpunkt in einem n-dimensionalen Raum dargestellt, welcher durch n Merkmale definiert ist. Er berechnet den Abstand zwischen den Punkten und ordnet dann die Kennzeichnung der unbeobachteten Daten, auf der Grundlage der Kennzeichnungen der nächstgelegenen beobachteten Datenpunkte, zu.

- **Decision Tree Classifier**

Der Entscheidungsbaum baut Baumzweige in einem hierarchischen Ansatz auf, und jeder Zweig kann als eine wenn-dann-Anweisung betrachtet werden. Die Zweige entwickeln sich durch Aufteilung des Datensatzes in disjunkte Teilmengen auf der Grundlage der wichtigsten Merkmale. Die endgültige Klassifizierung erfolgt an den Blättern des Entscheidungsbaums.

- **Random Forest Classifier**

Wie der Name schon sagt, ist der Random Forest eine Sammlung von Entscheidungsbäumen. Es handelt sich dabei um eine gängige Art von Ensemble-Methoden, die Ergebnisse von mehreren Prädiktoren zusammenfassen. Random Forest nutzt zusätzlich die Bagging-Technik, bei der jeder Baum auf einer Zufallsstichprobe des Originaldatensatzes trainiert wird und die Mehrheitsentscheidung der Bäume berücksichtigt wird. Im Vergleich zu Entscheidungsbäumen bietet er eine bessere Verallgemeinerung ist aber weniger interpretierbar, da dem Modell mehr Schichten hinzugefügt werden.

- **Voting Classifier**

Der Voting Classifier gehört ebenfalls zu den Ensemble-Methoden [22]. Er trainiert verschiedene Basismodelle, wie z.B. die oben genannten, und trifft Vorhersagen auf Grundlage der Aggregation der Ergebnisse von den einzelnen Basis-schätzern. Grundsätzlich kann dabei zwischen hard- und soft-voting unterschieden werden. Beim hard-voting entscheidet man sich für die Klasse, die am häufigsten von den einzelnen Klassifikatoren gewählt wird. Beim soft-voting sind die summierten Wahrscheinlichkeiten der einzelnen Klassen ausschlaggebend.

4 Posen Klassifikation

Um aus dieser Auswahl an Modellen das am Besten geeignete zu finden, wurden die Kriterien Genauigkeit und Geschwindigkeit verwendet. Nach erfolgreichem Training der Klassifizierer, wurde nun die Performance auf den ungesiehenen Testdaten verglichen. Als Genauigkeitsmaß wurde der F1-Score verwendet. Dieser eignet sich, wenn ein Ungleichgewicht zwischen den Klassen/Kategorien besteht. Als zweites Kriterium wurde die Zeit, für das verarbeiten der Testdaten, in Millisekunden gemessen.

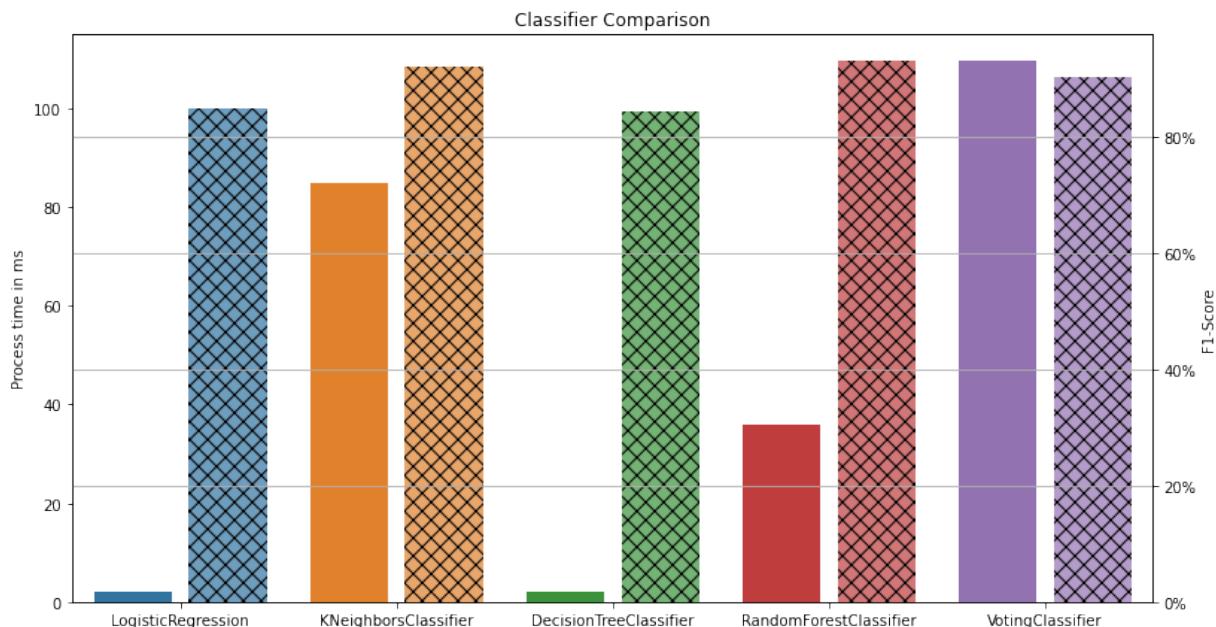


Abbildung 11: Vergleich verschiedener Klassifizierer

In der Graphik 11 gibt es zu jedem Klassifizierer zwei Balken, welche für die Kriterien Genauigkeit und Bearbeitungszeit stehen. Der linke Balken referenziert zur linken Skala, welche die benötigte Bearbeitungszeit in ms repräsentiert. Der rechte Balken hingegen bezieht sich auf die rechte Skala, welche die Genauigkeit der Vorhersagen in Prozent angibt. Der passende Klassifizierer sollte also ein niedrigen linken und hohen rechten Balken vorweisen.

Aus dem Vergleich ist zu erkennen, dass der Random Forest Classifier die besten Vorhersagen trifft und auch bei der Bearbeitungszeit etwa im Mittelfeld liegt. Die Logistic Regression und der Decision Tree Classifier sind zwar mit Abstand am effizientesten, müssen dafür bei der Genauigkeit etwas einbüßen. Der K-Nearest Neighbours Classifier, sowie der Voting Classifier haben eine vergleichsweise lange Bearbeitungszeit und sind deshalb weniger geeignet.

Der verwendete Classifier in der Anwendung ist deswegen der Random Forest. Alle diese Classifier wurden mit den Grundeinstellungen trainiert und es wurde auch noch kein Hyperparameter-Tuning vorgenommen. Aus Zeitgründen wurden diese Optimierung zu einem späteren Zeitpunkt nur für den ausgewählten Klassifizierer vorgenommen.

5 Anforderungsanalyse

Unser Ansatz zielt darauf ab, die Yogastellungen des Benutzers automatisch aus Echtzeitvideos, die von einer Rot-Grün-Blau (RGB)-Kamera aufgezeichnet werden, zu erkennen. Der Ablauf kann in vier Schritte unterteilt werden. Zunächst werden Daten gesammelt, die parallel zur Erkennung in Echtzeit stammen. Zweitens wird MediaPipe Pose verwendet, um alle 33 Schlüsselpunkte zu identifizieren. Die erkannten Schlüsselpunkte werden an unser Modell weitergeleitet, in dem unser ML-Modell Muster erkennt und diese analysiert. Abschliessend wird das Bild bearbeitet, so dass die Schlüsselpunkte für den Benutzer ersichtlich sind und an das Graphical User Interface (GUI) als Feedback ausgegeben.

5.1 Funktionale Anforderungen

Der Yoga-Trainer soll als GUI dem Benutzer zur Verfügung stehen und dabei vollständig in der Python Programmiersprache entwickelt werden. Während der Übung soll in Quasi-Echtzeit, die Yogastellung erkannt, analysiert und mit Feedback an den Benutzer zurückgegeben werden. Nebst den ganzen Funktionen während des Trainings, soll auch eine voll automatische statistische Auswertung der Performance möglich sein. Alles in allem gelten folgende Anforderungen an das Benutzersystem:

- Computer mit Windows 10
- Internetzugang (nur zur Installation)
- Python (Version: 3.9.0)
- RGB-Kamera

5.2 Benutzerfunktionen der Anwendung

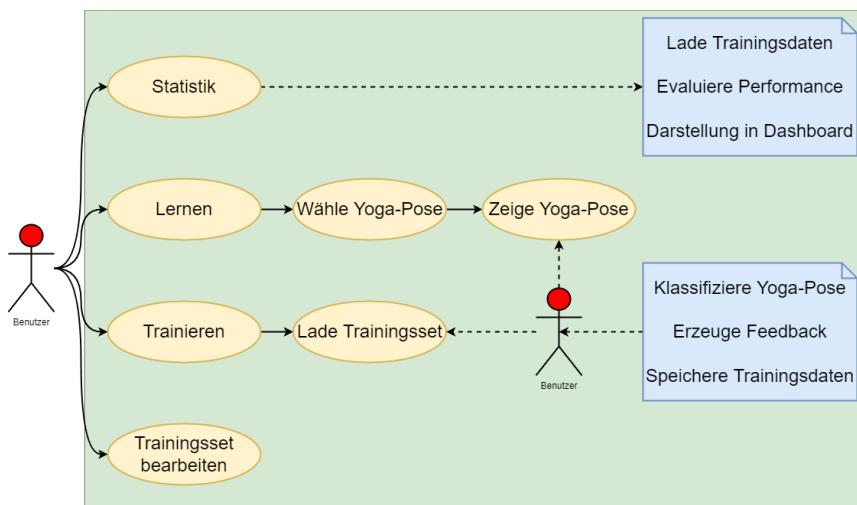


Abbildung 12: Use-Case Diagramm zur Anforderungsanalyse

5 Anforderungsanalyse

Wie in Abbildung 12 ersichtlich ist, wird die Anwendung in vier Hauptfunktionen unterteilt. In der ersten Hauptfunktion, Statistik 5.2.3, kann der Benutzer seine bisher geleisteten Trainings anschauen. Es werden ihm verschiedene Performancemetriken, mit Hilfe von Boxplots und Histogrammen dargestellt. In einer weiteren Hauptfunktion, Anpassen von Trainingssets 5.2.2 kann der Benutzer seine Trainings selber zusammenstellen. So kann jeder Benutzer sein Training nach seinen Ansprüchen gestalten. Die letzten beiden Hauptfunktionen Lernen und Trainieren, kann der Benutzer die verschiedenen Asanas trainieren. Der Lernmodus 5.2.1 wird dafür benutzt um die Yogastellungen zu verinnerlichen, ohne das sich mögliche Fehler auf die Statistik auswirken.

5.2.1 Lernen neuer Posen

In diesen beiden Funktionen wird nun tatsächlich Yoga angewandt, dabei stehen dem Benutzer zwei Möglichkeiten zur Verfügung. Als erstes empfiehlt sich der Lernmodus. In diesem Bereich kann eine Yoga Pose frei gewählt werden um diese zu verinnerlichen. Dazu wird dem Benutzer die Yoga Pose neben dem Kamerabild von sich selbst angezeigt und auf Fehler hingewiesen. Der Trainer versucht dabei mit verschiedenen Hilfsmittel zum Beispiel optisch oder akustisch die Yoga-Schülerin in die richtige Stellung zu führen. Da diese Methode zum Lernen dient, werden weder Erfolge noch Fehler aufgezeichnet.

Im Trainingsmodus kann der Benutzer ein vorgefertigtes Trainingsset auswählen und dieses schrittweise durcharbeiten. Die Yogastellungen müssen dabei für eine gewisse Zeit korrekt gehalten werden um das Training erfolgreich abzuschliessen. Die aufgezeichneten Daten, werden verarbeitet und zum Teil direkt während dem Training angezeigt, oder sie fließen in die Statistik nach dem Training und den Statistikbereich.

5.2.2 Trainieren mit mehreren Posen

Zu Beginn sollen einige Trainingssets bereits verfügbar sein. Damit der Benutzer sein Training auf seine Bedürfnisse anpassen kann, gibt es hier die Möglichkeit die bestehenden Sets anzupassen oder neue anzulegen. Die Trainingssets bestehen aus Yogastellungen und einer Zeit, welche angibt wie lange diese Pose gehalten werden soll, sowie optional die Anzahl Wiederholungen und Ruheintervalle.

5.2.3 Darstellen des Trainingsverlaufs

Damit der Benutzer einen Überblick über seine vergangenen Trainings bekommt, kann in diesem Bereich der gesamte Verlauf betrachtet werden. Es können die Trainings der vergangenen Tage betrachtet werden, sowie verschiedene Performance Auswertungen, welche mit Hilfe von statistischen Mitteln einfach verständlich dargestellt werden. Als Motivation für weitere erfolgreiche Trainings, werden auch einige Achievements zur Verfügung stehen.

5 Anforderungsanalyse

5.3 Erkennung der Yoga Asanas

Zu diesem Zeitpunkt sind nur wenige öffentliche Datensätze für Yogastellungen, sogenannte Asanas, verfügbar. Mit dem Datensatz *Yoga Poses Dataset* von Niharika Pandit [23] können die wohl am bekanntesten fünf abgedeckt werden. Zu den bekanntesten Asanas gehören:

- dog - „Adho Mukha Svanasana“
- goddess - „Utkata Konasana“
- tree - „Vrikshasana“
- plank - „Kumbhakasana“
- warrior - „Virabhadrasana“

Zur Validierung, oder auch um weitere Asanas zu verwenden, kann auf den *Yoga82* Datensatz [24] zugegriffen werden. Da *Yoga82*, wie in Bild 13 zu erkennen, sehr viele Daten umfasst, müssen die Bilder per Web Scraping bezogen werden.

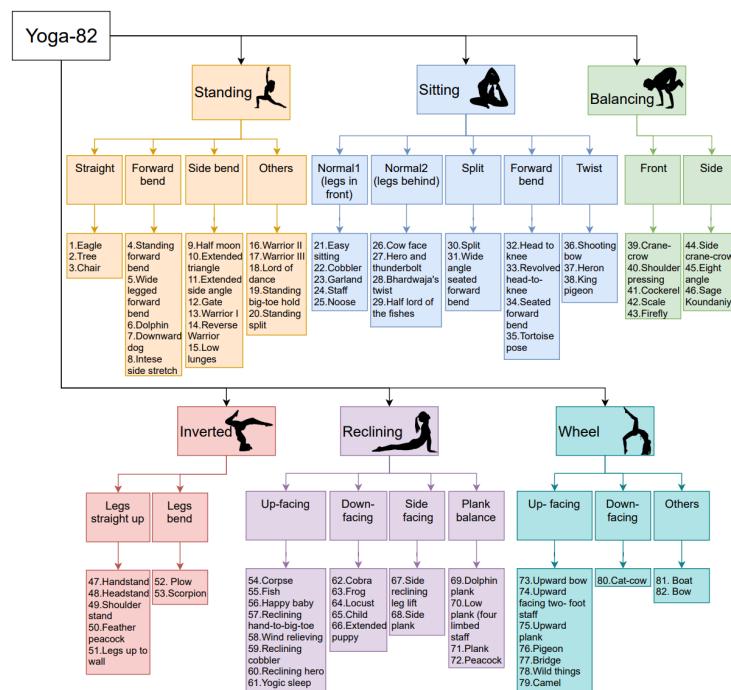


Abbildung 13: Asanas im Yoga82 Datensatz [24]

6 Konzeption und Design von Yoga AI

6 Konzeption und Design von Yoga AI

Aus den im Kapitel 5 beschriebenen Anforderungen wird nun ein Konzept vorgelegt. Es teilt die wichtigsten Funktionen in diverse Pakete auf und erleichtert damit die agile Entwicklung. Beim Konzept wird viel Wert auf eine möglichst einfache Bedienbarkeit gelegt, so dass die Applikation nach einer kurzen Demonstration dem Benutzer bereits vertraut erscheint.

6.1 Use Cases

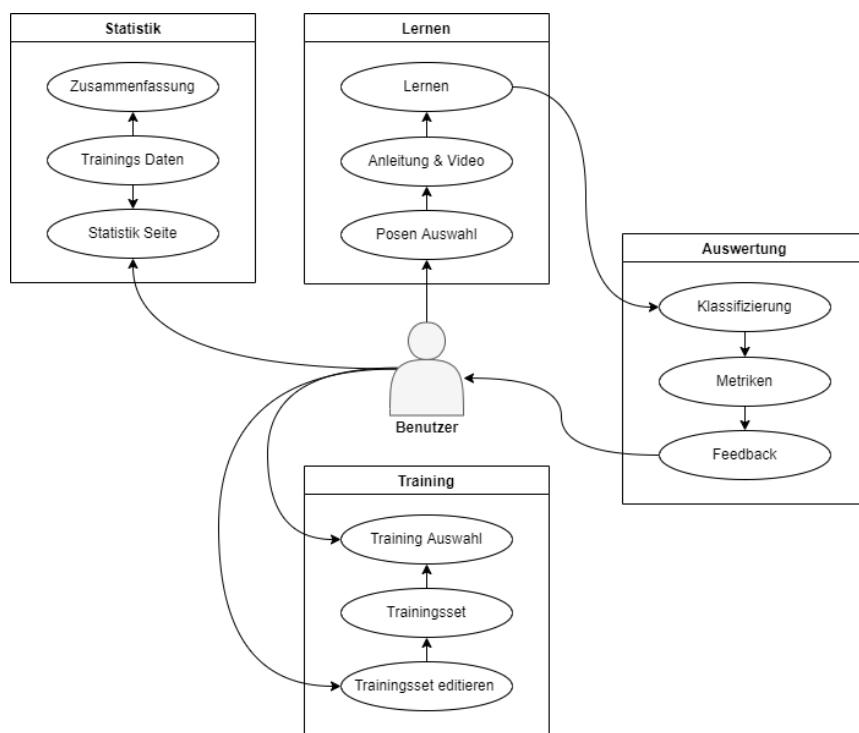


Abbildung 14: Use Case Diagramm

Es gibt vier Use Cases welche für die Entwicklung ausschlaggebend sind.

Lernen

Dieser Use Case befasst sich mit der Lernfunktion von Yoga AI. Es wird dem Benutzer ermöglicht eine Pose auszuwählen, welche er lernen möchte. Bevor mit der eigentlichen Yoga Übung gestartet wird, erhält der Benutzer ein kurze Einführung. Diese beinhaltet ein Video, welches die wichtigsten Aspekte der Pose aufzeigt, eine Schritt für Schritt Anleitung zur korrekter Durchführung sowie einigen Hinweisen zur möglichen Modifikation der Pose.

Training

Das Training umfasst zwei wichtige Funktionen von Yoga AI. Als erstes soll es dem Be-

6 Konzeption und Design von Yoga AI

nutzer ermöglicht werden, ein eigenes Trainingsset aus verschiedenen Yogastellungen zu erstellen. Aus dieser Funktion entsteht der zweite Teil dieses Use Cases, das Trainieren. Es soll die Möglichkeit bestehen, ein solches Trainingsset abzuarbeiten. Dabei werden die Koordinaten der Keypoints, sowie die am Ende der Trainingseinheit berechneten Scores gespeichert. Dies wird im dritten Use Case genauer betrachtet. Das Training ist eine Erweiterung der Funktionen, welche auch im Lernen verwendet werden. Dadurch entsteht auch der nachfolgende Use Case.

Auswertung

Die Auswertung bildet das Standbein von Yoga AI und beinhaltet die eigentliche Hauptaufgabe der vorherigen Use Cases. Hier wird die Posenerkennung von MediaPipe, die Klassifizierung der Yoga Pose mittels eines ML Algorithmus, die Bestimmung der Metriken und Rückgabe des Feedbacks, durchgeführt.

Statistik

Die ausgewerteten Daten der Trainings, sollen dem Benutzer eine Möglichkeit bieten sich zu verbessern. So soll es in einem separatem Bereich möglich sein, die durchgeföhrten Trainings miteinander zu vergleichen. Dank einer Sternebewertung, soll die Qualität des jeweiligen Trainings auf einen Blick ersichtlich sein.

6.2 Klassendiagramm

Mit den verschiedenen funktionellen Anforderungen an Yoga AI und dem Use-Case-Diagramm kann ein Klassendiagramm erstellt werden. Es ist zu beachten, dass hier aus Platzgründen ein kompaktes und nicht vollständiges Diagramm dargestellt wird. Das Klassendiagramm mit allen Datenfeldern und Methoden ist im Anhang (Anhang A: Klassendiagramm) ersichtlich.

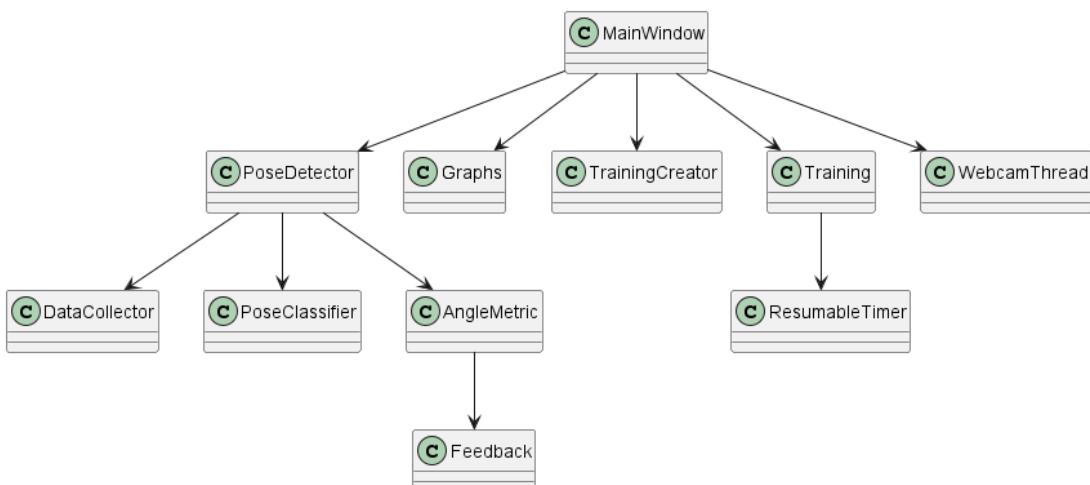


Abbildung 15: Reduziertes Klassendiagramm

6 Konzeption und Design von Yoga AI

6.3 User Interface Konzept

Eine leicht verständliche und intuitive Benutzeroberfläche trägt wesentlich dazu bei, dass die Bedienung von Yoga AI von den Benutzern als angenehm empfunden wird und ohne grossen Schulungsaufwand erlernt werden kann. Die Benutzeroberfläche hat eine einheitliche visuelle Struktur, die eine schnelle Orientierung ermöglicht. Am linken

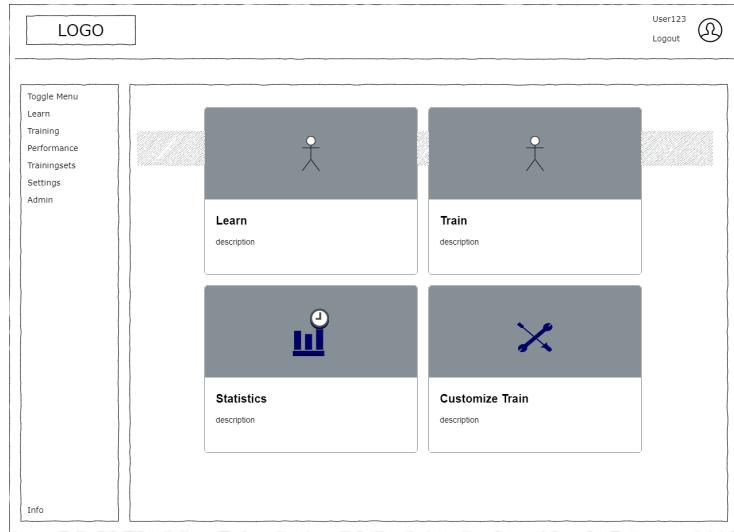


Abbildung 16: Konzept der Startseite

Rand des Fensters befindet sich eine einklappbare vertikale Menüleiste, die dem Inhaltsbereich mehr Platz gibt, wenn sie minimiert ist. Der grosse Inhaltsbereich erstreckt sich dynamisch über den Rest des Bildschirms. Die zielgerichtete Farbgestaltung unterstützt effizientes Arbeiten und lässt das Design modern erscheinen. Der Nutzer kann problemlos zwischen allen wichtigen Bereichen der App hin- und herwechseln. Deshalb sind in der Menüleiste alle wesentlichen Funktionen in einer sinnvollen Reihenfolge angeordnet:

- Startseite
- Yoga Seite
- Trainingsset Bearbeitung
- Statistik
- Lernbereich
- Einstellungen

Die Funktionen sind zugunsten einer optimalen Benutzerfreundlichkeit nicht in Untermenüs aufgeteilt. Weniger relevante Funktionen, wie die Einstellungen oder ein Kontaktbutton, sind in ein schlankes Menü innerhalb der Menüleiste ausgelagert, das nur bei Bedarf angezeigt wird.

Die Startseite (Abbildung 16) zeigt die vier wichtigsten Fähigkeiten von Yoga AI an und biete mit einem Klick auf die jeweilige Kachel, eine Erklärung wie jeder Bereich funk-

6 Konzeption und Design von Yoga AI

tioniert. Wird vom Benutzer, eine Pose zum lernen oder ein Trainingsset ausgewählt, wechselt die Anwendung direkt zum dementsprechenden Bereich.

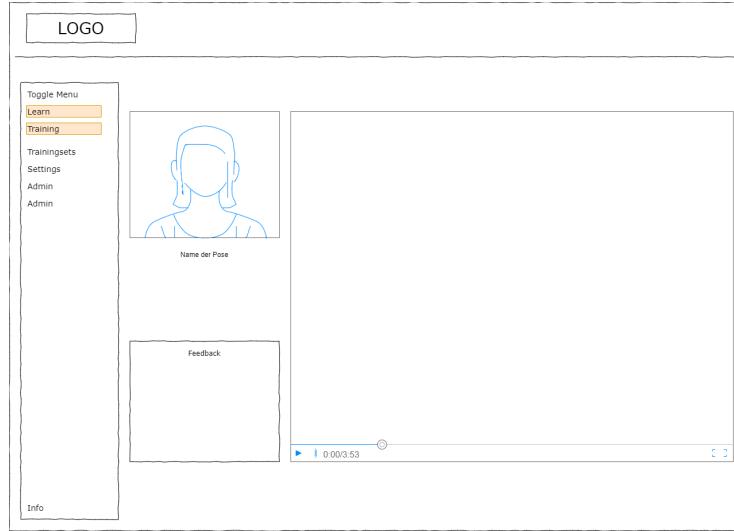


Abbildung 17: Konzept der Yoga Seite

Die eigentliche Trainingsseite, soll hauptsächlich mit dem Kamerabild des Benutzers gefüllt sein. Im Infobereich links, soll die aktuell geforderte Pose mit einem Bild und deren Namen erkennbar sein. Darunter wird das Feedback an den Benutzer zurückgegeben.

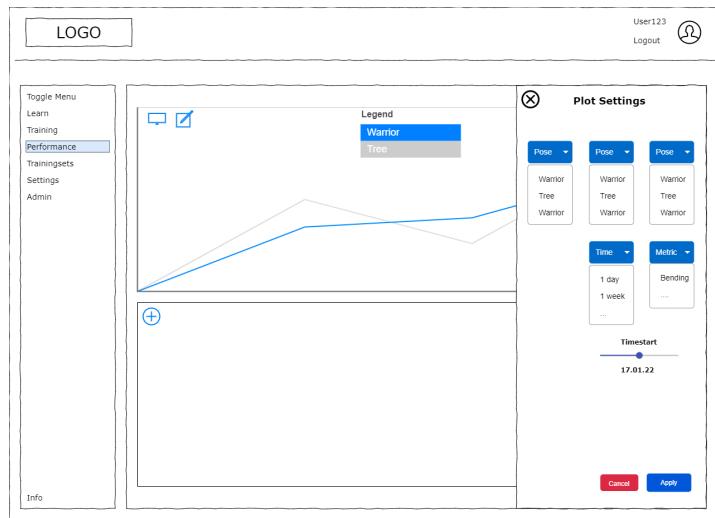


Abbildung 18: Konzept des Statistikbereichs

Im Statistikbereich soll der Benutzer die Möglichkeit haben seine vergangenen Trainings zu analysieren. Es werden Liniengraphen eingesetzt, welche je nach Auswahl, ein bestimmtes Training, eine bestimmte Pose oder einen definierten Zeitabschnitt anzeigen.

6.4 Posenerkennung und Erzeugung von Feedback

Eine Anforderung an die App ist, dass ein Benutzer eine Pose interaktiv erlernen und verbessern kann. Eine dynamische Bewegung kann allerdings kaum latenzfrei erkannt und verarbeitet werden. Deshalb wird die Bewegung von der jeweiligen Grundstellung einer Asana, hin zu Endstellung nicht weiter beurteilt. Ist der Benutzer in der Endstellung, wird anhand von den Winkeln zwischen definierten Gelenken bestimmt, ob die Pose korrekt ist. Dafür werden diese Winkel mit berechneten Werten, welche durch die Ausführung der ideal Pose definiert sind, verglichen. Liegt der Winkel an einem bestimmten Gelenk ausserhalb des definierten Genauigkeitsbereichs, soll dies dem Benutzer signalisiert und eine visuelle Hilfe erstellt werden. Dieser gesamte Ablauf vom Input des Kamerabildes bis zum Output des generierten Feedbacks soll als Pipeline in einer Klasse kontrolliert werden. Aus dieser Klasse können dann weiter Klassen aufgerufen werden, welche einzelne Teilschritte dieser Verarbeitung übernehmen. Der Ablauf, welches jedes Frame des Kamera-Inputs durchläuft ist wie folgt:

1. Posenerkennung durch ML-Modell
2. Winkelberechnung von definierten Gelenken
3. Vergleich berechneter Winkel mit Idealposition
4. Generieren eines visuellen Feedbacks, um Benutzer in richtige Postion zu helfen

Für die Posenerkennung wird das ML-Modell aus Kapitel 4 eingesetzt, welches fünf verschiedenen Posen erkennen kann [25].

- **bridge:** Schulterbrücke - Setu Bandha Sarvagasana (Rückbeuge)
- **cobra:** Kobra - Bhujangasana (Rückbeuge)
- **downdog:** Herabschauender Hund – Adho Mukha Svanasana (Umkehrhaltung)
- **tree:** Baum - Vrksasana (Standhaltung)
- **triangle:** Dreieck - Utthita Trikonasana (Standhaltung)

Möchten in einer späteren Version noch weitere Asanas hinzugefügt werden, müssen die Input-Daten für den Klassifizierer angepasst werden. Bei den praktischen Tests wurde nämlich festgestellt, dass diese fünf Asanas vom ML-Modell zwar erkannt werden, aber teils starke Unterschiede, mit welcher Wahrscheinlichkeit eine Pose erkannt wird, bestehen. Diese Unterschiede können durch Faktoren wie Abstand zur Kamera, Beleuchtung, Kleidung oder auch Hintergrund zustande kommen. Um nicht das gesamte ML-Modell nochmals zu überarbeiten, soll der Benutzer in der Anwendung diese Posenerkennungs-Schwelle, für eine individuelle Trainingsumgebung, selbst einstellen können.

Um weitere Asanas deutlich erkennen zu können, müssten die Input-Daten des ML-Modells erweitert oder angepasst werden. Diese erweiterten Daten könnten Winkelangaben an ausgewählten Gelenken sein. Diese Daten würden zu den bestehenden Daten, den Koordinaten an bestimmten Gelenken, hinzukommen. Dies bedeutet, das eingesetzte ML-Modell muss mit den zusätzlichen Daten der zur erweiternden Pose neu trainiert werden. Ebenfalls werden noch die erweiterten Daten, wie Winkel an ausge-

6 Konzeption und Design von Yoga AI

wählten Gelenken, zu den bestehenden Posen benötigt.

Nach dem Erkennen einer Pose werden die Winkel an definierten Gelenken berechnet und mit idealen Werten verglichen. Die Winkel dienen als Metrik um zu beurteilen, wie genau eine Person in der aktuellen Pose ist. Diese Metrik wird verwendet um Skalenninvarianz zu erreichen. Würde die Genauigkeit der Pose direkt über die Koordinaten beurteilt, müsste ein weiterer komplexer Schritt, für die Berechnung des Abstandes zur Kamera, hinzugefügt werden. Für den Vergleich der Winkel werden Bilder verwendet, bei welchen die Ausführung ähnlich wie die der idealen Pose aus der Literatur ist. Der Benutzer soll in der Anwendung dann selbst einstellen können, wie genau diese Winkel übereinstimmen müssen.

Das Erstellen eines visuellen Feedbacks ist ein zentraler Faktor, da es dem Benutzer helfen soll sich in die ideale Pose zu begeben. Eine erste Idee waren Schieberegler, welche die Abweichung zum idealen Winkel an einem bestimmten Gelenk signalisieren sollen. Jedoch wurde nach einer ersten Testphase erkannt, dass dies zu unübersichtlich für den Benutzer ist. Nach weiteren Ideen und Versuchen wurde schliesslich das Einblenden einer roten und grünen Zone direkt im Kamerabild als benutzerfreundlichste Variante gewählt. Die rote Zone markiert für den entsprechenden Keypoint, dass diese Position falsch ist und der Keypoint in den grünen Bereich verschoben werden soll.

Zusätzlich erhält der Benutzer auch ein akustisches Feedback, da der Bildschirm nicht permanent im Blickfeld des Benutzers ist. Mittels des Moduls pytsxs3 ist eine solche Sprachausgabe offline möglich. Jedoch ist diese Ausgabe nicht sehr ansprechend für den Benutzer und deshalb wird darauf verzichtet. Anstelle einer sprachlichen Ausgabe erhält der Benutzer Signaltöne, welche wichtige Ereignisse wie das Ende einer Pose signalisieren.

7 Umsetzung und Implementierung

Das Ziel war eine kontinuierliche Bereitstellung funktionsfähiger Software und deshalb wurde die agile Softwareentwicklung angewandt. Somit war nach dem Erstellen des GUI, die Grundlage erstellt um die weiteren Funktionen zu integrieren und direkt zu testen. Da diverse Funktionen (z.B. Feedback für Pose) erst nach dem Implementieren visuell auf die Effektivität und Benutzerfreundlichkeit getestet werden konnten, war es wichtig eine flexible Arbeitsmethode zu wählen. Solche Arbeitspakte wurden als entsprechend dem Konzept implementiert getestet und gegebenenfalls direkt angepasst, ohne nochmals eine komplettes Konzept dafür auszuarbeiten.

7.1 Modelldaten Beschaffung

Um die fünf Yoga-Asanas zu erkennen werden Bilder mit diesen Posen benötigt. Während der Auswertung der verschiedenen ML-Modelle wurde erkannt, dass diese bessere Ergebnisse erzielen, wenn noch weitere Posen hinzugefügt werden. Dies ist darauf zurückzuführen, dass für jede Klasse eine Wahrscheinlichkeit vorhergesagt wird. Ist der Benutzer nun in keiner der verwendeten fünf Posen, muss das Modell trotzdem die Wahrscheinlichkeit auf die verschiedenen Klassen aufteilen und dadurch kann es zu unerwünschten, sprich falschen Vorhersagen kommen. Aus diesem Grund wurden noch die folgenden Posen hinzugefügt:

- random, Pose bei verschiedensten Tätigkeiten
- sideangle
- warrior 1
- warrior 2

Diese Posen kommen lediglich während der Klassifikation zum Einsatz. Dies bedeutet, dass ein Benutzer diese Posen mit der Anwendung weder lernen, noch erkennen lassen kann.

Für die Beschaffung dieser Bilder, wurden zum einen der Datensatz *Yoga Poses Dataset* [23], welcher Bilder von Yoga-Asanas enthält, genutzt. Da jedoch in diesem Datensatz nicht alle von der Anwendung verwendeten Posen enthalten sind und auch weil dieser Datensatz eher klein ist, wurde noch auf den Datensatz *Yoga82* [24] zurückgegriffen. Dieser Datensatz enthält jedoch nur Internetadressen zu diesen Bildern. Mittels image scraping lassen sich daraus die Bilder extrahieren. Mithilfe eines Python-Script und des Packages BeatifulSoups [26], lassen sich aus diesen Textdateien mit den Links, die Bilder automatisch herunterladen.

Als nächster Schritt werden aus den Bildern die Keypoint-Daten, welche eine Körperstellung definieren, benötigt. Für diese Aufgabe kann MediaPipe BlazPose genutzt werden, welche sofern eine Person auf dem Bild erkannt wird, 33 Keypoints-Daten identifiziert. Da es zu jedem Keypoint x, y, z und ein Sichtbarkeitswert gibt, werden pro Bild 132 Werte generiert. Diese features dienen dem ML-Modell als Input-Daten. Wird also

7 Umsetzung und Implementierung

jedem Bild noch die Zielvariable, also die ausgeführte Pose, zugeordnet sind die Daten komplett für den nächsten Schritt, das Datenaufbereiten. Die Datenaufbereitung ist dank den normalisierten Werten schon erledigt. Es wurden jedoch zwei Varianten von Input-Daten getestet. Zum Einen die 132 Werte welche MediaPipe direkt liefert, zum Anderen eine Auswahl von 15 Keypoints, siehe Abbildung 19, und jeweils nur die x und y Koordinate davon. Die zweite Variante mit deutlich weniger Daten (30) erzielte bessere Ergebnisse und wurde darum verwendet.

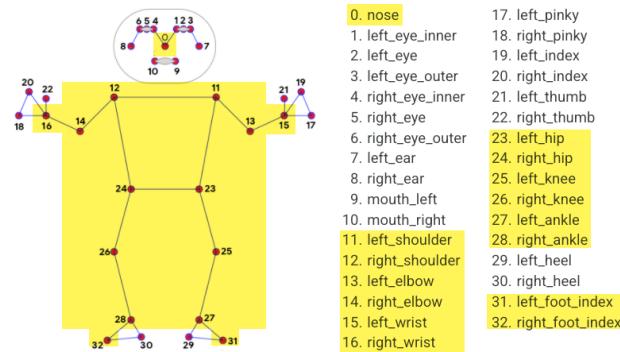


Abbildung 19: Verwendete Keypoints [1]

7.2 Erzeugung eines Klassifizierers

In Kapitel 4.2 sind die verschiedenen Klassifizierer einander gegenübergestellt und der Random Forest Classifier als zu verwendendes Modell ausgemacht. Alle Modelle sind mit den selben Daten (7.1) trainiert und getestet worden. Die Performance des verwendeten Random Forest Classifier wurde in einem Classificaton Report, siehe Abbildung 20, zusammengefasst.

Der Bericht zeigt die Klassifizierungsmetriken Präzision, Recall und F1-Score für jede Klasse [27]. Die Metriken werden unter Verwendung von True und False Positives sowie True und False Negatives berechnet. Positiv und negativ sind in diesem Fall generische Bezeichnungen für die vorhergesagten Klassen. Es gibt vier Möglichkeiten zu überprüfen, ob die Vorhersagen richtig oder falsch sind:

- **TN / True Negative:** wenn ein Fall negativ war und negativ vorhergesagt wurde
- **TP / True Positive:** wenn ein Fall positiv war und positiv vorhergesagt wurde
- **FN / Falsch Negativ:** wenn ein Fall positiv war, aber negativ vorhergesagt wurde
- **FP / Falsch Positiv:** wenn ein Fall negativ war, aber positiv vorhergesagt wurde

Die Precision definiert die Genauigkeit der positiven Vorhersagen.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

Der Recall ist der Anteil der Positivmeldungen, die korrekt identifiziert wurden.

7 Umsetzung und Implementierung

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

Der F1-Score ist ein gewichtetes harmonisches Mittel aus Precision und Recall, wobei die beste Punktzahl eins und die schlechteste null beträgt.

$$\text{F1Score} = 2 \cdot \frac{\text{Recall} \cdot \text{Precision}}{\text{Recall} + \text{Precision}}$$

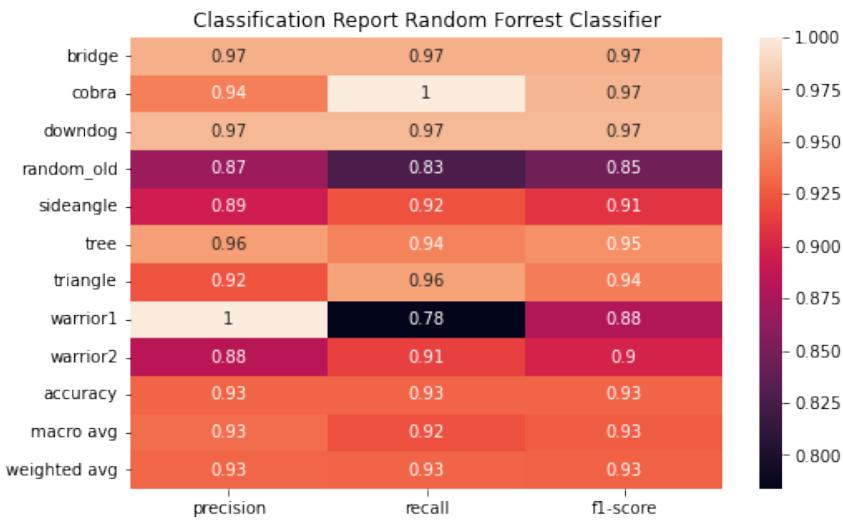


Abbildung 20: Classification Report

Entscheidend für die Anwendung, ist das Erkennen der fünf verwendeten Asanas. Für diese Klassen sind alle Klassifizierungsmaßnahmen deutlich über 90%. Bei den nicht verwendeten Klassen wie random oder warrior1 gibt es beim Recall deutlich schlechtere Resultate. Das bedeutet, dass eine höhere Anzahl an Bildern aus diesen Klassen, einer falschen Klasse zugewiesen wurde. Da diese Klassen nur für die allgemeine Verbesserung des ML-Modells zum Einsatz kommen, sind diese Werte vernachlässigbar.

7.2.1 Hyperparameter-Tuning

Mittels Hyperparameter-Tuning soll der Random Forest Classifier noch optimiert werden [28]. Da nur eine vage Vorstellung von den besten Hyperparametern vorhanden ist, ist ein guter Ansatz zur Eingrenzung der Suche die Auswertung einer grossen Bandbreite von Werten für jeden Hyperparameter. Mit der RandomizedSearchCV-Methode von Scikit-Learn [29] ist es möglich ein Raster von Hyperparameter-Bereichen zu definieren und nach dem Zufallsprinzip Stichproben aus dem Raster ziehen, wobei k-fache Kreuzvalidierung mit jeder Wertekombination durchgeführt wird. Die k-fache Kreuzvalidierung ist ein Verfahren, mit dem die Performance des Modells anhand neuer Daten geschätzt wird [30]. Die zu optimierenden Hyperparameter sind folgende:

- **n_estimators:** Anzahl der Bäume

7 Umsetzung und Implementierung

- **max_features:** Maximale Anzahl von Merkmalen, die für die Aufteilung eines Knotens berücksichtigt werden
- **maxdepth:** Maximale Anzahl der Ebenen in jedem Entscheidungsbaum
- **min_samples_split:** Mindestanzahl der Datenpunkte, die in einem Knoten platziert werden, bevor der Knoten geteilt wird
- **min_samples_leaf:** Mindestanzahl der in einem Blattknoten zulässigen Datenpunkte
- **bootstrap:** Methode zur Stichprobenziehung von Datenpunkten (mit oder ohne Ersetzung)

```
1 # Number of trees in random forest
2 n_estimators = [int(x) for x in np.linspace(start = 200,
3     stop = 2000, num = 10)]
4 # Number of features to consider at every split
5 max_features = ['auto', 'sqrt']
6 # Maximum number of levels in tree
7 max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
8 max_depth.append(None)
9 # Minimum number of samples required to split a node
10 min_samples_split = [2, 5, 10]
11 # Minimum number of samples required at each leaf node
12 min_samples_leaf = [1, 2, 4]
13 # Method of selecting samples for training each tree
14 bootstrap = [True, False]
15 # Create the random grid
16 random_grid = {'n_estimators': n_estimators,
17     'max_features': max_features,
18     'max_depth': max_depth,
19     'min_samples_split': min_samples_split,
20     'min_samples_leaf': min_samples_leaf,
21     'bootstrap': bootstrap}
22
23 # Use the random grid to search for best hyperparameters
24 # First create the base model to tune
25 rf = RandomForestClassifier()
26 # Random search of parameters, using 3 fold cross validation,
27 # search across 100 different combinations, and use all cores
28 rf_random = RandomizedSearchCV(estimator = rf,
29     param_distributions = random_grid, n_iter = 100,
30     cv = 3, verbose=2, random_state=42, n_jobs = -1)
31 # Fit the random search model
32 rf_random.fit(X_train, y_train)
```

Listing 1: Hyperparameter Tuning

Um RandomizedSearchCV zu verwenden, muss zunächst ein Parameterraster erstellt werden, aus diesem werden dann während der Anpassung Stichproben gezogen. Bei jeder Iteration wählt der Algorithmus eine andere Kombination von Merkmalen aus. Insgesamt gibt es $2 \times 12 \times 2 \times 3 \times 3 \times 10 = 4320$ Einstellungen! Der Vorteil einer Zufallssuche ist jedoch, dass nicht jede Kombination ausprobiert wird, sondern nach dem

7 Umsetzung und Implementierung

Zufallsprinzip ausgewählt wird, um eine breite Palette von Werten zu erfassen.

Die wichtigsten Argumente in `RandomizedSearchCV` sind `n_iter`, das die Anzahl der auszuprobierenden Kombinationen steuert und `cv`, das die Anzahl der für die Kreuzvalidierung zu verwendenden Faltungen angibt (in diesem Fall sind 100 bzw. 3 verwendet). Mehr Iterationen decken einen grösseren Suchraum ab, und mehr Kreuzvalidierungs Faltungen verringern die Wahrscheinlichkeit einer Überanpassung, aber eine Erhöhung beider Werte erhöht die Laufzeit. Die besten Parameter aus der Anpassung der Zufallssuche könne anschliessend analysiert werden:

```
1 rf_random.best_params_
2
3 {'n_estimators': 1800,
4  'min_samples_split': 2,
5  'min_samples_leaf': 1,
6  'max_features': 'auto',
7  'max_depth': 20,
8  'bootstrap': False}
```

Listing 2: Beste Parameter für das Modell

Werden diese Parameter angewandt, wird die Performance nur marginal besser und die Bearbeitungszeit steigt beträchtlich, siehe Abbildung 21. Der F1-Score steigt um ca. 0.33% während sich die Bearbeitungszeit mehr als verzehnfacht. Somit wird in der Applikation der Random Forest Classifier mit den Standardparametern verwendet.

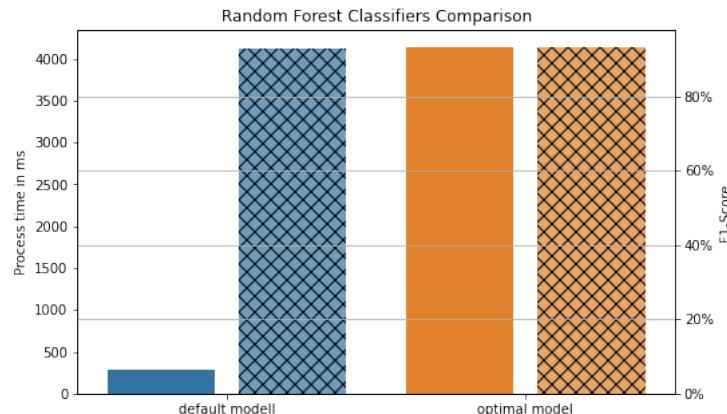


Abbildung 21: Vergleich Standard und Bestes Model des Random Forest Classifier, Linker Balken repräsentiert die Bearbeitungszeit und der Rechte die Genauigkeit

7.3 Entwicklung des User Interfaces

Das Hauptproblem bei grafischen Benutzeroberflächen ist, dass nicht dauernd kontrolliert werden kann, auf welche Schaltfläche der Benutzer zu einem bestimmten Zeitpunkt klickt. Deshalb verwendet man ereignisorientierte Programmierung für grafische

7 Umsetzung und Implementierung

Oberflächen. Das bedeutet, dass der Programmablauf durch bestimmte Ereignisse, in diesem Fall Klicks und Eingaben des Nutzers, beeinflusst wird. Für dieses Programm bedeutet das, dass es meistens darauf wartet, dass der Benutzer etwas anklickt. Sobald dies geschieht, tritt ein Ereignis auf, das wiederum eine bestimmte Methode auslöst, welche für die Ausführung der Aktion des Benutzers verantwortlich ist. Sobald das Programm diese Aufgabe erledigt hat, geht es zurück in eine Endlosschleife und wartet auf weitere Anweisungen.

7.3.1 PySide

Für die Entwicklung der Oberfläche wird PySide verwendet. PySide ist eine Python-Anbindung des plattformübergreifenden GUI-Entwicklungs-Toolkits Qt, das derzeit von der Firma Qt im Rahmen des Qt for Python-Projekts [31] entwickelt wird. PySide bietet Lesser General Public License (LGPL)-lizenzierte Python-Bindings für Qt 6 und enthält eine komplette Toolchain zur schnellen Generierung von Bindings für beliebige Qt-basierte C++-Klassenhierarchien. PySide Qt-Bindings ermöglichen sowohl freie Open-Source- als auch proprietäre Software-Entwicklung und zielen letztlich auf die Unterstützung von Qt-Plattformen ab. In dieser Arbeit wird PySide6, die aktuell neueste Version von PySide, verwendet.

7.3.2 Qt Designer

Qt Designer ist ein Qt-Werkzeug, welches eine What You See Is What You Get (WYSIWYG) Benutzeroberfläche zur Verfügung stellt, mit der produktiv und effizient GUIs für PyQt-Anwendungen erstellt werden können [32]. Oberflächen werden erstellt, indem man QWidget-Objekte per Drag and Drop auf ein leeres Formular ablegt. Danach kann man mit verschiedenen Layout-Managern, diese Objekte zu einem kohärenten GUI anordnen.

Qt Designer ist plattform- und programmiersprachenunabhängig. Er erzeugt keinen Code in einer bestimmten Programmiersprache, sondern erstellt .ui-Dateien. Diese Dateien sind XML-Dateien mit detaillierten Beschreibungen, wie Qt-basierte GUIs zu erzeugen sind. Der Inhalt solcher .ui-Dateien kann mit pyuic6, einem Befehlszeilenwerkzeug, das mit PyQt geliefert wird, in Python-Code übersetzt werden. Danach kann dieser generierte Python-Code in der GUI-Anwendungen verwendet werden.

7.3.3 Layout

Das Ziel beim Layout ist, die Applikation so schlank wie möglich zu gestalten. Umgesetzt heisst das, die Widgets mit denen der Benutzer interagiert, so gross wie möglich und alles andere nur so gross wie nötig, zu gestalten. So ist zum Beispiel das seitliche Menu ausklappbar, damit nebst dem Tooltip auch eine Beschriftung zum Icon sichtbar wird. Aber auch der Standard Windows Rahmen um die Applikation herum wird entfernt, um den verfügbaren Platz auf dem Bildschirm so gut wie möglich auszunutzen.

7 Umsetzung und Implementierung

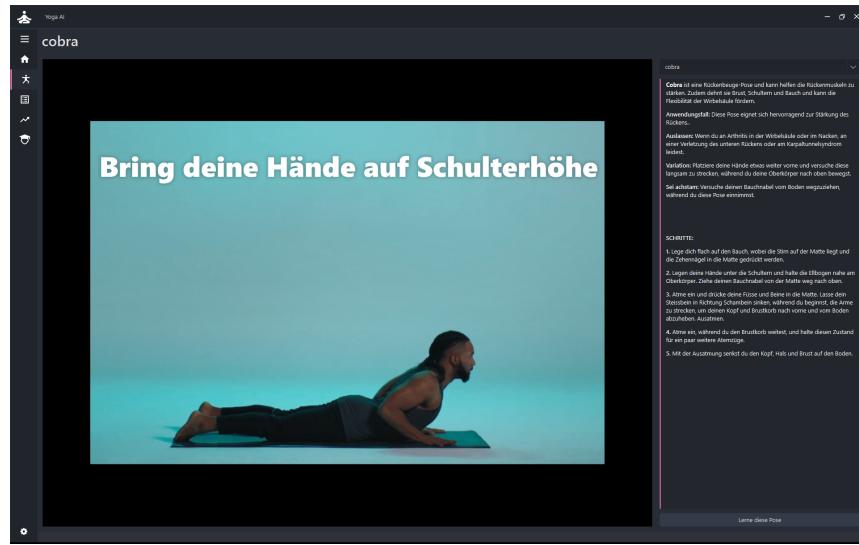


Abbildung 22: Lernbereich mit Erklärvideo und Schritt-für-Schritt Anleitung

Beim Farbdesign orientiert sich Yoga AI an dem Dracula Theme, eines der gängigsten Farbdesigns im Bereich der Software Entwicklung. Der dunkelblaue Hintergrund mit den pinken Akzenten, sowie den Icons von Google's Material Theme, lassen das Benutzerinterface sehr modern wirken.

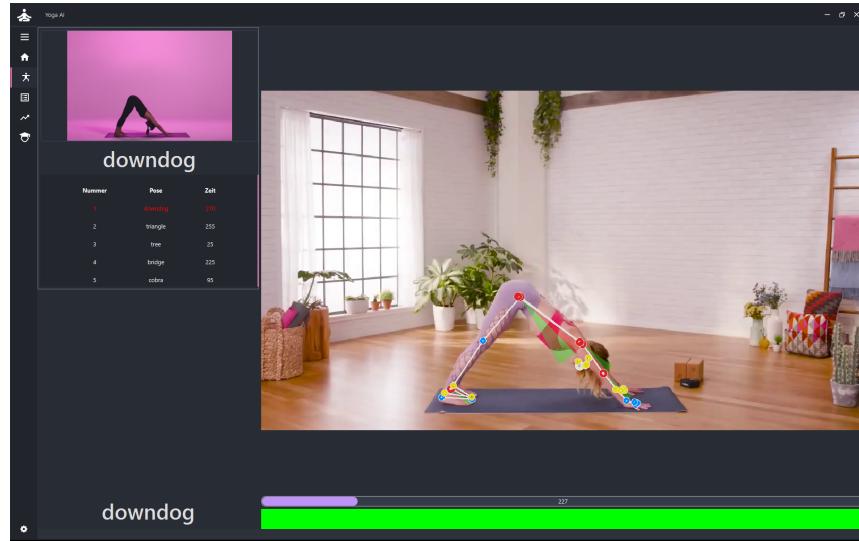


Abbildung 23: Trainingsbereich

7.3.4 QSettings

Benutzer erwarten, dass eine Anwendung ihre Einstellungen sitzungsübergreifend speichert. Mit QSettings ist es möglich Anwendungseinstellungen in der Systemregistrie-

7 Umsetzung und Implementierung

rung unter Windows zu speichern und wiederherzustellen. So ist zum Beispiel, die Einstellung der Farben im Liniendiagramm anpassbar, um Benutzern mit Dyschromatopie ein besseres Erlebnis zu gewährleisten. Aber auch die Optionen der Posenerkennung mit MediaPipe sind, je nach Trainingsumgebung oder Rechnerleistung, anpassbar.

7.3.5 Qt Ressourcen System

Das Qt Ressourcen System ist ein plattformunabhängiger Mechanismus für die Auslieferung von Ressourcendateien einer Anwendung. Es bringt die Vorteile, dass Übersetzungsdateien, Icons aber auch Videos ohne systemspezifische Mittel lokalisiert und in der Applikation geladen werden können. Die Resource Collection File (.qrc) Datei wird mit Qt Designer (Kapitel 7.3.2) erzeugt und anschliessend mit dem Qt Resource Compiler (rcc) zu einer Python Datei kompiliert.

7.4 Implementierung der Posenerkennung

Die Applikation kann fünf verschiedenen Asanas erkennen und ein Feedback für diese generieren. Der Prozess vom Kamera-Input bis zur Feedback-Ausgabe ist eine komplexe Aufgabe, welche in mehrere Teilschritte unterteilt und in entsprechende Klassen gegliedert ist. Die Schnittstelle zwischen User Interface (UI) und der Verarbeitung der Daten ist die Klasse Detection. Diese kommuniziert mit den Klassen Classifier, AngleMetric und Feedback, welche jeweils einzelne Teilschritte übernehmen.

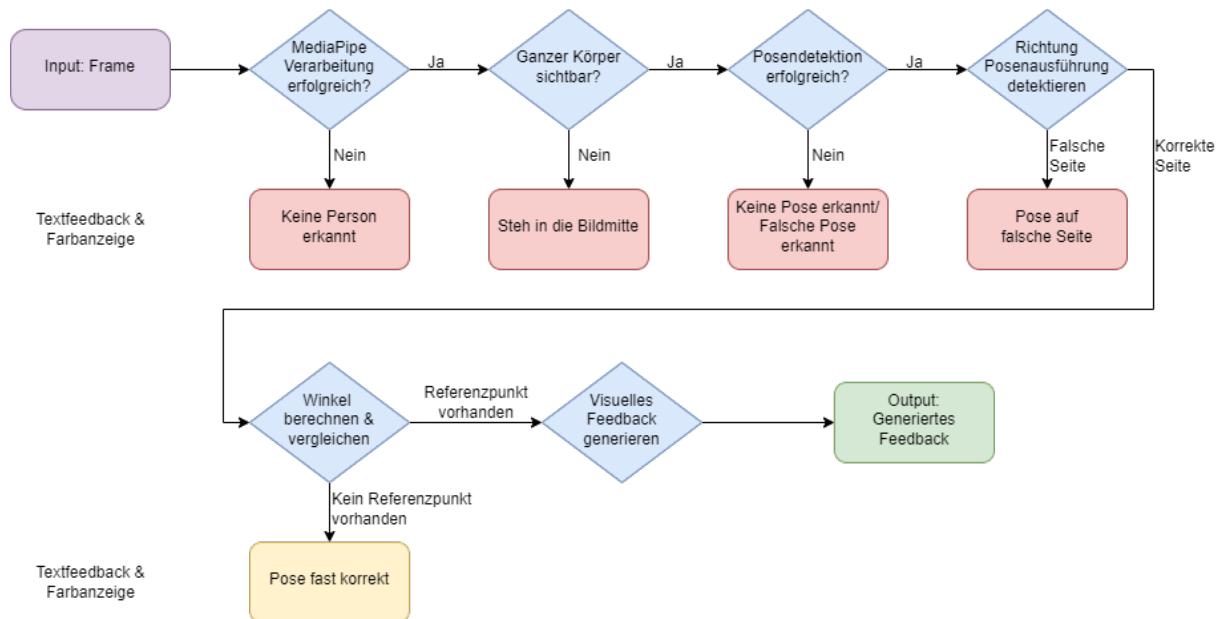


Abbildung 24: Ablauf von Detektion bis zum Feedback

Der Input dieser Pipeline ist ein Frame welches von der Kamera kommt. Dieses Bild wird nun mittels der Klasse Pose von MediaPipe verarbeitet und die Ausgabe sind die

7 Umsetzung und Implementierung

33 Keypoints, falls eine Person in diesem Bild erkannt wird. Anschliessend wird überprüft, ob sich die gesamte Person im Bild befindet, dies geschieht mittels den Sichtbarkeitswerten von Füssen, Schultern und Händen. Ist auch diese Kontrolle erfolgreich, werden die 33 Keypoints auf 15 reduziert (Abbildung 19) und die x- und y-Werte werden über eine definierte Filtergrösse gemittelt. Dieser Schritt soll kleine Störungen dieser Werte verhindern. Ab diesem Punkt, werden vom aktuellen Bild nur noch diese gemittelten Werte verwendet.

Als nächstes wird von der Klasse Classifier die Wahrscheinlichkeitsvorhersage gemacht. Ist diese in einer bestimmten Klasse grösser als die einstellbare Posenerkennungs-Schwelle, wurde eine Pose erkannt. Diese Schwelle kann vom Benutzer zwischen 0.6 bis 0.8 individuell eingestellt werden. Nun muss noch überprüft werden, ob eine geforderte Pose, durch Lern- oder Trainingsmodus, definiert ist. Ist dies der Fall, müssen erkannte und geforderte Pose übereinstimmen für eine erfolgreiche Detektion. Im freien Modus, muss dies Überprüfung nicht gemacht werden.

Anschliessend wird die Richtung/Seite, in welche der Benutzer die Pose ausführt, detektiert. Dies wird für Posen, welche auf beide Seiten ausgeführt werden sollten, benötigt. Beim nächsten Schritt werden von der Klasse AngleMetric die Winkel an 10 Gelenken, Ellenbogen, Schulter, Hüfte, Knie und Ferse, berechnet. Um den Winkel zu berechnen werden jeweils drei Punkte benötigt, also zusätzlich die benachbarten Keypoints eines Gelenkes.

Als nächstes werden die berechneten Winkel mit den Winkeln der idealen Pose verglichen. Diese ideale Pose ist durch Koordinaten von ca. 30 Bilder pro Pose definiert, in welchen die Ausführung der Pose als ideal betrachtet wird. Aus diesen Bildern sind die Winkel berechnet und es ist ein einstellbarer Genauigkeitsbereich hinterlegt. Dieser ist mittels der Funktion `percentile` von `numpy` definiert. Über diese Quantils-Einstellung kann z.B. gefordert werden, dass der Benutzer bei einer Pose mit den Winkeln nicht mehr als 20% vom Mittelwert der hinterlegten Winkel, der idealen Pose, abweichen darf. Ansonsten färbt sich dieser Keypoint rot, was dem Benutzer signalisiert, dass die lokale Körperstellung noch nicht ganz korrekt ist.

Folgende Genauigkeits-Einstellungen sind für den Benutzer möglich, siehe auch Abbildung 25.

- einfach, Abweichung vom Mittelwert maximal 45%
- mittel, Abweichung vom Mittelwert maximal 30%
- schwierig, Abweichung vom Mittelwert maximal 20%

Bevor nun die Generierung des visuellen Feedbacks möglich ist, muss auf jeder Seite ein Winkel von Schulter, Hüfte oder Knie im definierten Genauigkeitsbereich liegen. Dies wird als sogenannter Referenzpunkt benötigt. Ist ein solcher auf beiden Seiten gefunden, wird ein visuelles Feedback, ausgehend vom Referenzpunkt, generiert.

7 Umsetzung und Implementierung

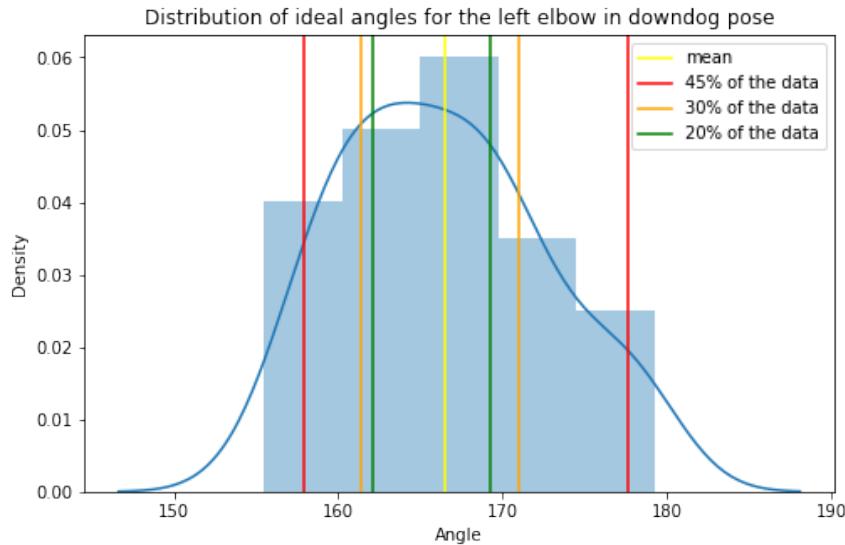


Abbildung 25: Verteilung der hinterlegten Winkel für die ideal Pose,
Die vertikalen Linien kennzeichnen die drei Genauigkeitsbereiche

7.4.1 Implementierung des Feedbacks

Der Benutzer erhält während einer Übung verschiedene Arten von Feedback. Eines davon in Textform und Farbanzeige, wie in Abbildung 24 ersichtlich, um eine grundlegende Rückmeldung zu liefern. Die Farbanzeige zeigt jeweils die Hintergrundfarbe, rot orange oder grün an. Zusätzlich ertönen Signaltöne welche, den Start des Timers, bei beidseitigen Asanas eine abgeschlossene Seite und noch das Ende einer Pose signalisieren. Ist der Benutzer in der richtigen Pose, aber es sind noch nicht alle Winkel zwischen den Gelenken korrekt, so wird ein visuelles Feedback direkt im Kamerabild eingeblendet. Dies soll dem Benutzer helfen, sich in die korrekte Körperstellung zu bringen. Das erzeugen dieses Feedback ist ein komplexer Ablauf und kann in folgende Teilschritte unterteilt werden.

1. Referenzpunkt auf jeder Seite finden (Schulter, Hüfte oder Knie)
2. Ausgehend von diesem Referenzpunkt, Gelenke mit falschen Winkel finden
3. Falsch positionierter Keypoint finden und dessen Drehrichtung identifizieren
4. Für jedes Gelenk mit falschem Winkel, zwei Kreissegmente generieren

Der genaue Ablauf wird anhand der Abbildung 24 und den darin falsch positionierten Schultern erläutert. Als erstes wird ein gültiger Referenzpunkt aus Schultern, Hüfte oder Knie für jede Seite gesucht. An diesem Punkt sind die Winkel im Genauigkeitsbereich. Dies bedeutet, dass die benachbarten Keypoints richtig positioniert sind. In diesem Fall ist dieser Punkt auf beiden Seiten die Hüfte, da bei den Schultern gestartet wird und dort der Winkel falsch ist.

Nun werden Listen mit falsch positionierten Keypoints, eine in Richtung Hand und eine

7 Umsetzung und Implementierung

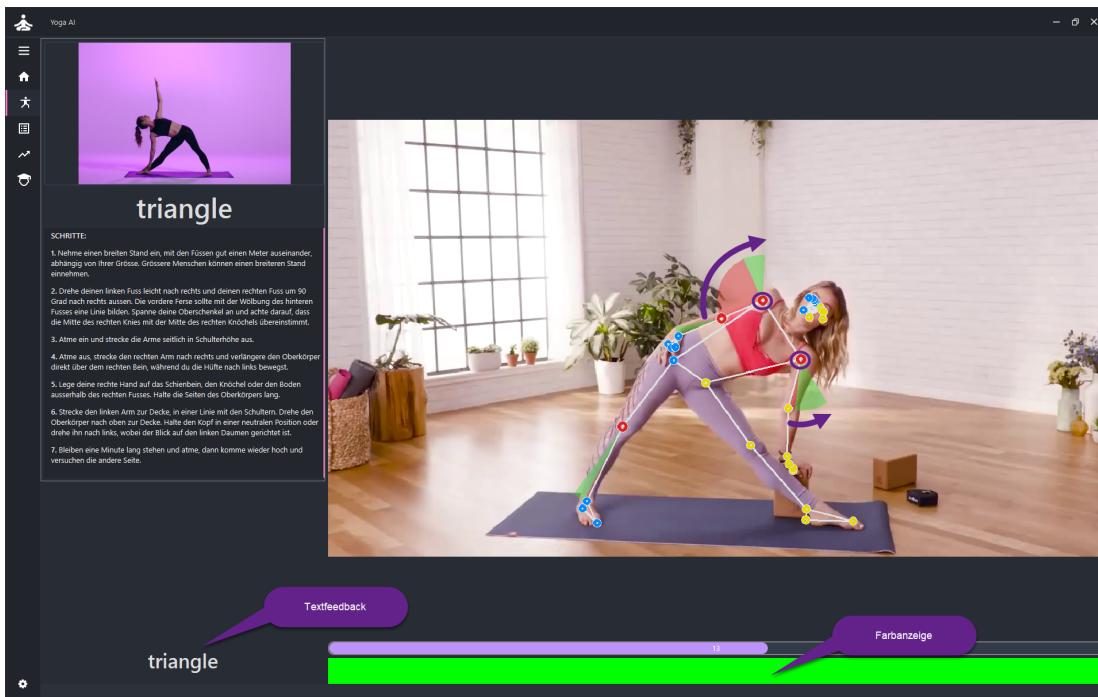


Abbildung 26: Feedback

in Richtung Fuss, erstellt. Auf der rechten Seite sind dies für die Liste Richtung Fuss das Knie und für die Richtung Hand die Schulter und der Ellenbogen. Für die linke Seite hat nur die Liste Richtung Hand den Keypoint Schulter eingetragen.

Danach wird jeder dieser Einträge abgearbeitet und für die rechte Schulter geht das wie folgt. Es werden die benachbarte Keypoints miteinbezogen. Der Keypoint Hüfte wird als lokaler Referenzpunkt definiert und der Keypoint Ellenbogen als falsch positionierter Keypoint festgelegt, dieser gilt es zu verschieben. Nun werden diese Koordinaten auf einer Kreisbahn, definiert durch den Abstand zur Schulter, im Uhrzeigersinn um ein Grad verschoben. Anschliessend wird der Winkel der Schulter nochmals berechnet und verglichen. Ist die Differenz kleiner geworden war dies die richtige Richtung, ansonsten muss im Gegenuhrzeigersinn verschoben werden.

Als nächstes werden mittels dem Package cv2 zwei Kreisbögen, einer rot für falschen Bereich und einer grün für den idealen Bereich, gezeichnet. Das Zentrum sind die Koordinaten der Schulter und der Radius ist gegeben durch den Abstand zum Ellenbogen. Der Startwinkel, zum zeichnen des roten Kreisbogens, ist definiert durch die Position des Ellenbogen zur X-Achse hin. Der Endwinkel für den roten und sogleich der Startwinkel des grünen Bogens, ist definiert durch den vorherigen Starwinkel plus der Winkeldifferenz zum idealen Winkel. Der Endwinkel des grünen Bogens ist gegeben durch den eingestellten Genauigkeitsbereich des idealen Winkels.

Die violetten Pfeile in der Graphik visualisieren wie die Ellenbogen bewegt werden

7 Umsetzung und Implementierung

müssen um diese in die richtige Position zu bringen. Ebenfalls zusätzlich markiert, sind die rot gefärbten Keypoints der Schultern, welche eine fehlerhafte Körperstellung signalisieren.

7.5 Implementierung des Trainingssets

Damit der Benutzer eine eigene Abfolge von verschiedenen Yogastellungen erstellen kann, wird ihm im Bereich des Trainingssets genau diese Möglichkeit geboten. Es können neue Trainingssets erstellt aber auch bestehende editiert werden. Dazu wählt der Benutzer ein bestehendes Set und wählt eine Yoga-Pose aus, bestimmt die Trainingsdauer für diese Pose mittels eines Sliders und fügt diese dann dem Trainingsset hinzu. Das aktuelle Set wird dabei in einer Tabelle dargestellt. Dort befindet sich auch die Möglichkeit ausgewählte Yogastellungen aus dem Training zu entfernen. Im Hintergrund werden diese Daten in eine JSON-Datei gespeichert, mit dem Vorteil, dass das Dateiformat ohne grossen Aufwand in ein Python Dictionary umgewandelt werden kann. Der Ablauf eines Trainingssets, ist in Abbildung 27 ersichtlich.

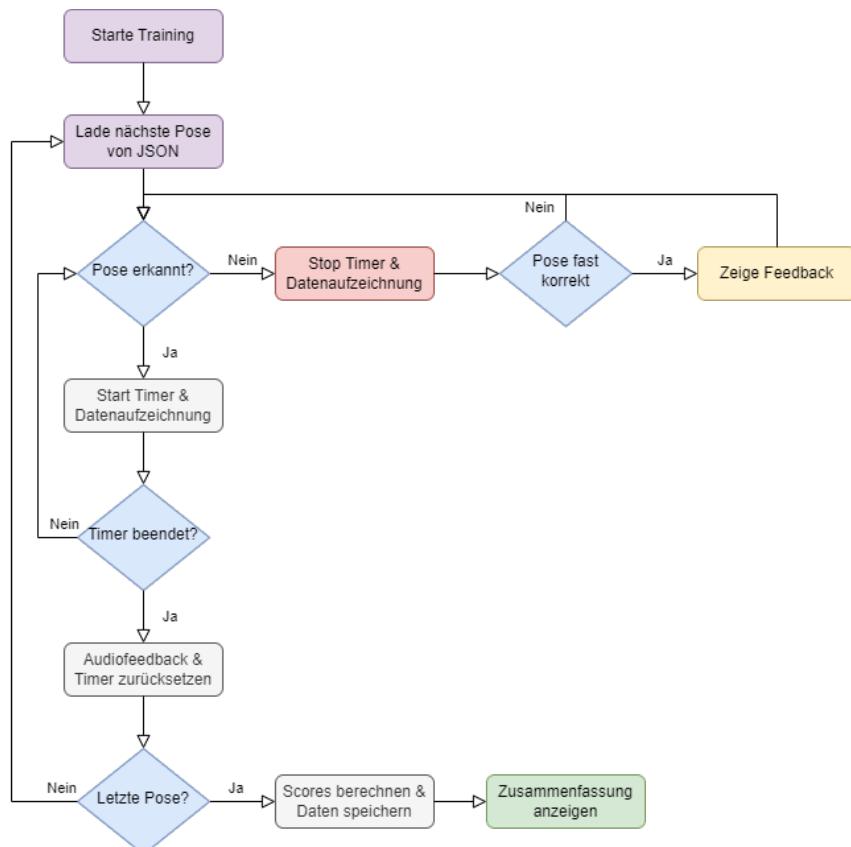


Abbildung 27: Ablaufschema beim Training

7 Umsetzung und Implementierung

Während dem Training, sieht der Benutzer dieses Trainingsset auf der linken Seite des Bildschirms, siehe Abbildung 23. Die aktuell geforderte Pose wird in der linken oberen Ecke mit einem Bild und Text angezeigt, ausserdem wird sie im Trainingset farblich hervorgehoben. An der unteren Bildschirmseite ist die aktuell von Yoga AI erkannte Pose zu sehen, sowie eine Fortschrittsanzeige welche den Timer repräsentiert. Als weiterer Indikator ist zudem ein Balken vorhanden, der die Farbe je nach aktueller Qualität der Yogastellung ändert.

7.6 Speicherung und Auswertung der Daten

Während des Trainings werden die gemittelten Daten jedes zehnten Bildes gespeichert. Da die Bildfrequenz bei etwa 25 FPS liegt, bedeutet dies, dass pro Sekunde ca. zwei Datenpakete hinzukommen. Diese Daten bestehen unter anderem aus den Koordinaten der Schlüsselpunkte, die benötigte Zeit für die aktuelle Pose, sowie drei Punktzahlen (Scores). In der Applikation wird pandas DataFrame für das Datahandling verwendet, dies bringt die Vorteile mit sich, dass pandas bereits für MediaPipe verwendet wird und somit kein zusätzliches Python Paket, sogenannte Packages, benötigt wird, aber auch zum Speichern und Laden der Daten im csv-Format.

7.6.1 Berechnung der Scores

Die Anwendung berechnet nach jeder Lern- oder Trainingseinheit vier verschiedene Scores, welche im Statistikbereich auch nochmals weiter analysiert werden können. Alle diese Scores liegen im Wertebereich von 0 bis 100.

- Genauigkeitsscore (angle_livescore)
- Bewegungsscore (motion_livescore)
- Zeitscore (time_score)
- Gesamtscore

```
1 # angle and motion score, sum over all angles on every timestep
2 pose_data['angle_livescore'] =\
3     round((PoseConstants.SCORE_MAX - (pose_data[angle_columns].sum(1) / 
4 PoseConstants.ANGLE_ERROR_STRETCH) ** 2).clip(lower=0))
5
6 pose_data['motion_livescore'] =\
7     round((PoseConstants.SCORE_MAX - PoseConstants.MOTION_ERROR_FACTOR * 
8 pose_data[motion_columns].sum(1)).clip(lower=0))
9
10 # time score, is per pose unchanged
11 pose_data['time_score'] =\
12     round(PoseConstants.SCORE_MAX * (pose_data['requested_duration'] / 
13 pose_data['actual_duration']))
```

Listing 3: Berechnung der Scores

7 Umsetzung und Implementierung

Der Genauigkeitscore repräsentiert wie exakt die Pose ausgeführt wurde und wird in Listing 3 auf den Zeilen 2-4 berechnet. Für diesen Score werden an jedem Zeitschritt die Winkelabweichungen der einzelnen Gelenke summiert. Dieser Fehler wird, nachdem er durch den Faktor ANGLE_ERROR_STRETCH dividiert wurde, quadriert. Diese Division bewirkt eine Streckung der Parabel an der X-Achse. Dadurch werden kleine Winkelabweichungen schwächer gewichtet als grosse Winkelabweichung. Dieser Fehler wird vom Score-Maximum abgezogen. Ab einer summierten Winkelabweichung von 150 Grad beträgt dieser Score 0.

Der Bewegungsscore sagt aus, wie ruhig der Benutzer die Pose gehalten hat und wird in Listing 3 auf den Zeilen 6-8 berechnet. Für diesen Score werden die x und y Koordinaten der 15 (Abbildung 19) benutzten Keypoints verwendet. An jedem Zeitschritt wird die Differenz zum Vorherigen berechnet und über alle diese Keypoints summiert. Anschliessend wird dieser Wert mit dem MOTION_ERROR_FACTOR multipliziert und vom Score-Maximum abgezogen.

Der Zeitscore, welcher in Listing 3 auf den Zeilen 11-13 berechnet wird, repräsentiert die benötigte Zeit, um eine Pose erfolgreich abzuschliessen. Dieser wird nicht wie die vorherigen Scores pro Zeitschritt berechnet, sondern ist über die gesamte Zeit einer Pose unverändert. Für die Berechnung wird die geforderte Zeit durch die effektive Zeit für die Ausführung der Pose dividiert und mit dem Score-Maximum multipliziert. Benötigt der Benutzer für die Ausführung einer Pose doppelt so lange wie die geforderte Zeit, so ist dieser Score 50.

Der Gesamtscore unterscheidet sich für Lern- und Trainingsmodus. Beim Lernmodus wird der Durchschnitt der drei vorherigen Scores, von der erlernte Pose, verwendet. Beim Trainingsmodus wird auch der Durchschnitt der drei Scores verwendet, jedoch über alle Posen des gesamten Trainings. Entsprechend dieses Scores erhält der Benutzer auch noch eine Sternebewertung zwischen null und fünf.

7.6.2 Datenbetrachtung in der Applikation

Zur Darstellung und Analyse der aufgezeichneten Daten, werden zwei verschiedene Python Pakete verwendet. Die statischen Balken- und Liniendiagramme, wie zum Beispiel auf der Übersichtsseite nach dem Training, geben dem Nutzer Informationen über das gerade absolvierte Training. Sie sollen einfach zu verstehen und trotzdem genügend Mehrwert bieten. Da diese Graphiken keine Interaktion anbieten müssen, werden ausschliesslich PySides QChart Widgets verwendet.

Der Statistikbereich hingegen bietet es an sämtliche Daten genauer zu inspizieren. Dafür wird PyQtGraph [33], eine Grafikbibliothek, welche häufig in technischen Anwendungen benötigt wird, benutzt. Diese Bibliothek erlaubt es dem Benutzer, mit den Plots zu interagieren, um jedes Detail der Daten zu betrachten. In diesem Bereich können Trainings, aber auch eine Zusammenfassung über einen bestimmten Zeitverlauf, besichtigt werden. Die Standardeinstellung zeigt den Verlauf der verschiedenen Scores, in Abhän-

7 Umsetzung und Implementierung

gigkeit der ausgewählten Posen. Werden alle Yoga-Posen ausgewählt, hilft eine farbliche Kennzeichnung zur Unterscheidung zwischen den Asanas. Im Erweiterten Modus können zusätzliche die Koordinaten der verschiedenen Keypoints geplottet werden.



Abbildung 28: Statistikkbereich

7.7 Bereitstellung der Applikation

Die Bereitstellung (freezing) einer Anwendung ist ein wichtiger Teil eines Python-Projekts. Dabei werden alle erforderlichen Ressourcen gebündelt, damit sie auf dem Rechner eines Benutzers zur Verfügung stehen, auch wenn auf dem Zielrechner keine Python Installation vorliegt. Damit wird sichergestellt, dass keine zusätzlichen Programme oder Bibliotheken vom Benutzer installiert werden müssen.

7.7.1 Setup und Kompilieren

Um eigenständige ausführbare Dateien aus Python Skripten zu erzeugen, wird `cx_Freeze` [34] verwendet. Mit diesem Paket ist es möglich plattformübergreifende Executables ohne Leistungsverlust zu erstellen. Um solche Dateien zu erzeugen, wird ein Setup-Script geschrieben, welches die nötigen Parameter, wie Icondatei, Name und Version von Yoga AI, sowie Einstiegspunkt in das Programm, setzt. `cx_Freeze` exkludiert automatisch, alle nicht benötigten Python Pakete. Damit die Applikation aber noch weniger Speicherplatz benötigt, werden die Pakete `PySide6` und `pyarrow` komprimiert und bei `opencv` wird eine headless Version verwendet. Bei anderen grossen Paketen wie `sklearn` und `scipy` ist es leider unumgänglich diese in voller Ausführung zu integrieren.

7 Umsetzung und Implementierung

```
1 from cx_Freeze import Executable, setup
2 # Setupscript to build an executable with following options
3
4 options = {
5     "build_exe": { [...],
6         "zip_include_packages": ["PySide6", "pyarrow"],
7         "includes": [...],
8         "packages": ["sklearn", "scipy"],
9         "include_files": ["./resources", "./platforms"]
10    }
11 }
12
13 executables = [
14 Executable(
15     script="main.py",
16     base=base,
17     target_name=QSettings.value('app/name'),
18     icon="./app/icon.ico")
19 ]
20
21 setup(
22     name=QSettings.value('app/name'),
23     version=QSettings.value('app/ver'),
24     description=QSettings.value('app/desc'),
25     author="Stephan Seliner, Alex Koller",
26     options=options,
27     executables=executables,
28 )
```

Listing 4: Ausschnitt aus Setup Skript

7.7.2 Erstellung eines Installationsprogramms

Damit sich der Benutzer nicht mit der richtigen Verwendung des erstellten Executables auseinandersetzen muss, wird ein Installationsprogramm erstellt. Dazu wird Inno Setup [35], ein Installer für Windows Programme von Jordan Russell und Martijn Laan, verwendet. Dank diesem Compiler kann Yoga AI mit nur einer einzigen Datei verteilt werden und garantiert durch den geleiteten Installationsvorgang, dass die Applikation funktionsfähig installiert wird.

Für allfällige Änderungen an Yoga AI, ist mit Inno Setup bereits eine Lösung zum Update der Software möglich, auch zur Erweiterung der Kundenorientierung bietet das Tool, die Möglichkeit, in verschiedenen Sprachen ausgeführt zu werden.

7.8 Übersicht über die Module

Die komplette Implementierung von Yoga AI umfasst insgesamt 23 Python-Dateien, mit 1982 Zeilen Code und 1290 Zeilen Kommentaren. In den nachfolgenden Unterkapitel, werden die wichtigsten Funktionen der einzelnen Modulen aufgezeigt.

7.8.1 GUI Package

Dieses Package umfasst drei Dateien, welche alle für die Erzeugung des GUI benötigt werden.

- resources_rc.py: Enthält alle Ressourcen, welche mit Qt Designer eingefügt wurden. Dies beinhaltet alle Icons, Videos und Bilder
- ui_functions.py: Beinhaltet alle Funktionen des GUI sowie das Handling der Input-Bilder, welche in Listing 5 ersichtlich ist
- ui_main.py: Diese Datei wird mittels rcc, aus einer .ui Datei erzeugt und beinhaltet das eigentliche GUI

7.8.2 Pose Package

- angle_metric.py: Dieses Modul ist für die Berechnung der Winkel zuständig
- classifier.py: Diese Datei stellt die Funktionen (Listing 6) zur Klassifizierung der Yoga-Pose zur Verfügung
- data_collector.py: Beinhaltet alle Funktionen zur Speicherung der aufgezeichneten Daten
- detection.py: Dieses Modul enthält alle Funktionen zur Verarbeitung eines Bildes. Es ist die Verbindung zwischen der Benutzeroberfläche und dem Classifier, AngleMetric und Feedback
- feedback.py: Hier sind alle Funktionen zur Erzeugung des visuellen Feedbacks zu finden
- pose_constants.py: Enthält Konstanten welche zur Darstellung des Skeletts verwendet werden

7.8.3 Statistics Package

Dieses Paket enthält nur eine einzelne Datei, graphs.py. Diese Methoden erzeugen alle Graphen welche dem Benutzer zur Verfügung stehen und reagieren auf Einstellungen im Statistikbereich. Listing 7 zeigt ein Beispiel.

7.8.4 Trainer Package

- trainer_functions.py: Enthält alle Funktionen des Timers
- training.py: Beinhaltet die Klasse Training, welche Yoga-Trainings mit einzelnen aber auch mehreren Posen bietet z.B. Listing 8
- training_creator.py: Dieses Modul ist für das Laden, Speichern und Bearbeiten der Trainingssets zuständig

8 Evaluation

Bei der Entwicklung des Klassifizierers wurde bereits während der Realisierung mit bekannten Metriken wie Präzision, Recall und F1-Score, die Qualität des Algorithmus getestet. Die Auswertung verschiedener Klassifizierer und deren Performance ist in Kapitel 4.2, sowie 7.2 ersichtlich.

Zur Bestimmung der Qualität von Yoga AI wurde aber hauptsächlich auf Praxistests gesetzt. Die Anwendung wurde dafür jeweils nach Abschluss zusätzlicher Funktionen von den Autoren und teilweise auch von weiteren Personen getestet. Diese häufigen Praxistest waren hilfreich um einige Schwachstellen der Konzeption und Entwicklung schon früh zu erkennen und zu verbessern.

Eine dieser erkannten Schwachstellen ist die Klassifizierung der Yoga Posen. Diese kann in seltenen Fällen ungenau, sprich es kann eine Pose falsch oder auch gar nicht erkannt werden. Dies liegt hauptsächlich daran, dass dem ML-Modell nur Koordinaten als Input dienen und diese auch stark vom Abstand zu Kamera abhängen. Würden zusätzlich noch Winkel von ausgewählten Gelenken dem ML-Modell als Input mitgeliefert, würde dies räumliche Abhängigkeit zum Teil verbessert werden. Dadurch könnten auch noch weitere Asanas hinzugefügt werden und es würden möglicherweise bessere Ergebnisse bezüglich der Klassifikation erzielt.

Um den aktuellen Klassifizierer zu verbessern, wurde eine vom Benutzer einstellbare Posenerkennungs-Schwelle hinzugefügt. Dieser Wert kann für eine individuelle Trainingsumgebung passend eingestellt werden, um anschliessend problemlos in dieser Umgebung zu trainieren.

Ebenfalls wurde auf diese Weise ein geeignetes visuelles Feedback selektiert. Bei den ersten Versionen wurde die mangelnde Benutzerfreundlichkeit und Unterstützung für den Benutzer als grosse Schwachstelle erkannt. So wurde das Konzept für dieses Feedback überarbeitet, bis schliesslich der passender Ansatz gefunden wurde.

8.1 Quantitative Auswertung

Da eine ausführliche Nutzerumfrage mit breitem Publikum sehr viel Zeitaufwand bedeutet, wurde eine reduzierte Testphase mit zehn Testpersonen durchgeführt. Hierfür wurde die Anwendung diesen Personen, mit und ohne Yoga Kenntnissen, nach einer kurzen Einführung zur Verfügung gestellt. Anschliessend wurde deren Feedback in einer Umfrage (siehe Anhang C: Fragebogen) zusammengefasst und gesamthaft ausgewertet.

Wie in Abbildung 29 zu sehen ist, kam es bei einzelnen Testpersonen, jeweils in den Bereichen von Installation, zurechtfinden in der Anwendung und unerwünschtes Verhalten während der Benutzung, zu nicht zufriedenstellenden Benutzererlebnissen. Diese damit in Verbindung stehenden Schwachstellen der Software wurden anschliessend bestmöglich behoben. Im Grossen und Ganzen erhält Yoga AI, in den verschiedenen

8 Evaluation

Punkten aber eine gute Bewertung. Auch waren die allgemeinen Rückmeldungen zu Anwendung sehr positiv. Yoga AI, sowie die Idee dahinter wurden als erfolgreich und geglückt empfunden. Die tabellarische Auswertung ist in Anhang D: Auswertung Fragebogen ersichtlich.

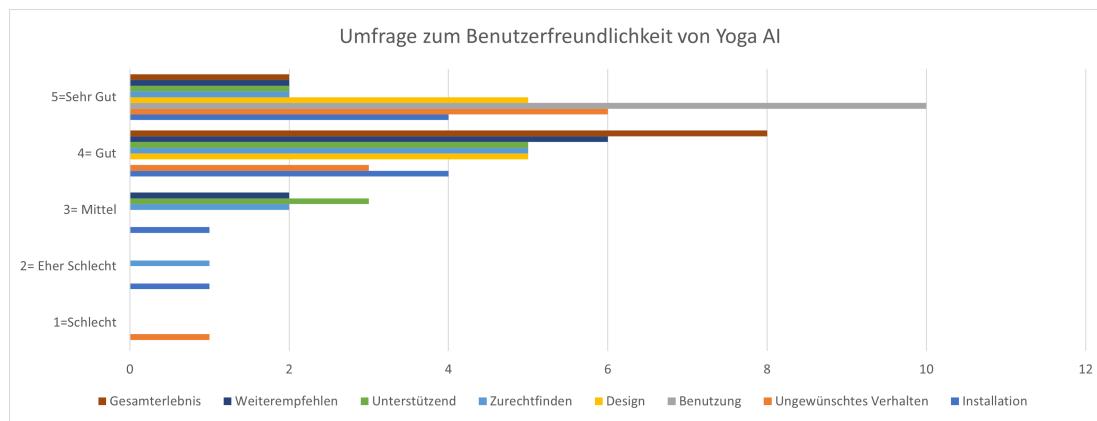


Abbildung 29: Auswertung Nutzerumfrage

Von den Testpersonen gingen auch einige Verbesserungsvorschläge ein. Dabei wird gewünscht einen Sprachcoach, der klare Anweisungen gibt, zu integrieren. Dies ist jedoch wie in Kapitel 6.4 bereits erwähnt schwierig zu realisieren, da offline keine geeigneten Sprachassistenten zur Verfügung stehen.

Weiter wird vorgeschlagen zusätzliche Tipps bei falscher Pose zu erhalten, den Timer bei fast korrekter Pose nicht zu stoppen oder die Anwendung mit Hintergrundmusik zu erweitern.

Manche Benutzer würden des Weiteren eine Web Applikation einer Desktop Applikation bevorzugen. Dies bringt den Vorteil, dass Yoga AI nicht nur auf Notebooks oder PCs einsatzfähig ist, sondern auch auf Smart TVs, Tablets oder Smartphones. Dies würde allerdings eine komplette Überarbeitung des Programms benötigen.

9 Fazit und Ausblick

Das Erkennen, Anwenden passender Metriken und insbesondere das Erstellen eines geeigneten Feedbacks für den Benutzer, stellte sich als besondere Herausforderung heraus.

Für das Erkennen einer Yoga-Asana, wird als erster Schritt MediaPipe BlazePose verwendet, welches sehr gute Ergebnisse im Erkennen der Keypoints einer Person, liefert. Anschliessend identifiziert der trainierte Random Forest Classifier eine der fünf Asanas oder eine sonstige Körperstellung. Bei diesem Schritt entstanden erste Probleme, verursacht durch individuelle Trainingsumgebungen oder den Ähnlichkeiten einiger Asanas. Beim nächsten Schritt, dem Festlegen geeigneter Metriken, wurde klar, dass einerseits die Asanas auf verschiedenen Arten ausgeführt werden können und andererseits die exakte Pose sehr individuell ist. Darum wurde eine Bandbreite von gut ausgeführten Asanas hinterlegt, um mit der Pose des Benutzers zu vergleichen.

Beim Erstellen eines Feedbacks bestand die Schwierigkeit darin, dieses intuitiv und benutzerfreundlich zu gestalten. So wurden mehrere Ideen und Ansätze gebraucht, bis eine passende Lösung gefunden wurde.

Der Statistikbereich war nach der ganzen Vorarbeit deutlich einfacher. So werden die erhaltenen Werte, aus Metrik und Feedback, für den Benutzer erfassbar präsentiert.

Einen professionellen Yoga-Lehrer lässt sich mit Yoga AI aber nicht ersetzen. Dieser kann auch den Ablauf von der Grundstellung in die Endpose beurteilen oder individuelle Fehlhaltungen erkennen. Zudem besteht Yoga auch nicht nur aus dem korrekten Ausführen einer Asana. Es besteht noch aus weiteren Aspekten, wie einer bewussten Atmung oder der inneren Reflexion. Auch wird Yoga meist in einer Gruppe praktiziert, diese soziale Interaktion mit den anderen Yoga-Schülern ist ein ebenfalls wichtiger, nicht zu vernachlässigender Aspekt.

Da Yoga eine praktische Lebensphilosophie ist, die auf eine Vereinigung von Geist und Körper abzielt, wird mit dieser Anwendung auch nur der Aspekt der Körperübungen behandelt.

Im Ermessen der Autoren wird die Konzeption und Entwicklung der Applikation dennoch als Erfolg betrachtet. Die Praxistest haben ergeben, dass ein Grossteil der Grundfunktionen und weiteren Anforderungen einer Trainingsapplikation nicht nur implementiert wurden, sondern auch auf Gefallen der Testpersonen stiess. Durch die Evaluationsphase von Yoga AI, konnten Schwachstellen der Konzeption und Entwicklung aufgedeckt und die vorliegenden Probleme behoben, oder in einer zukünftigen Erweiterung integriert werden.

Literaturverzeichnis

- [1] Valentin Bazarevsky und Ivan Grishchenko. *On-device, Real-time Body Pose Tracking with MediaPipe BlazePose*. English. 13. Aug. 2020. URL: <https://ai.googleblog.com/2020/08/on-device-real-time-body-pose-tracking.html>.
- [2] Elisha Odemakinde. „Human Pose Estimation with Deep Learning – Ultimate Overview in 2022“. In: *viso.ai* (8. Jan. 2021). URL: <https://viso.ai/deep-learning/pose-estimation-ultimate-overview/> (besucht am 06.01.2022).
- [3] *A Comprehensive Guide to Human Pose Estimation*. 2021. URL: <https://www.v7labs.com/blog/human-pose-estimation-guide> (besucht am 06.01.2022).
- [4] Lubomir D. Bourdev. „Poselets and Their Applications in High-Level Computer Vision“. In: *undefined* (2011). URL: <https://www.semanticscholar.org/paper/Poselets-and-Their-Applications-in-High-Level-Bourdev/a3f1db123ce1818971a57330d82901683d7c2b67>.
- [5] Rohit Josyula und Sarah Ostadabbas. *A Review on Human Pose Estimation*. 13. Okt. 2021. URL: <https://arxiv.org/pdf/2110.06877> (besucht am 06.01.2022).
- [6] Alexander Toshev und Christian Szegedy. „DeepPose: Human Pose Estimation via Deep Neural Networks“. In: *2014 IEEE Conference on Computer Vision and Pattern Recognition* (Juni 2014). DOI: 10.1109/cvpr.2014.214. URL: <http://dx.doi.org/10.1109/CVPR.2014.214>.
- [7] Sudharshan Chandra Babu. „A 2019 guide to Human Pose Estimation with Deep Learning“. In: *AI & Machine Learning Blog* (12. Apr. 2019). URL: <https://nanonets.com/blog/human-pose-estimation-2d-guide/> (besucht am 06.01.2022).
- [8] *Overview of Human Pose Estimation - Deep Learning | Neuralet*. 11. Dez. 2021. URL: <https://neuralet.com/article/human-pose-estimation-with-deep-learning-part-i/> (besucht am 06.01.2022).
- [9] Jonathan Tompson u. a. *Efficient Object Localization Using Convolutional Networks*. 16. Nov. 2014. URL: <https://arxiv.org/pdf/1411.4280>.
- [10] Shih-En Wei u. a. *Convolutional Pose Machines*. 30. Jan. 2016. URL: <https://arxiv.org/pdf/1602.00134>.
- [11] Fan Zhang u. a. *MediaPipe Hands: On-device Real-time Hand Tracking*. 18. Juni 2020. URL: <https://arxiv.org/pdf/2006.10214>.
- [12] Arian Alavi. „A Review of Google’s New Mobile-Friendly AI Framework: Mediapipe“. In: *The Startup* (1. Okt. 2020). URL: <https://medium.com/swlh/a-review-of-googles-new-mobile-friendly-ai-framework-mediapipe-25d62cd482a1> (besucht am 06.01.2022).

- [13] GitHub. *google/mediapipe: Cross-platform, customizable ML solutions for live and streaming media*. 6. Jan. 2022. URL: <https://github.com/google/mediapipe> (besucht am 06.01.2022).
- [14] GitHub. *AriAlavi/SigNN: The goal of SigNN is to develop a software which is capable of real-time translation of American Sign language (ASL) into English text. It is capable of all letters, including J and Z (which are edge cases in the alphabet)*. 6. Jan. 2022. URL: <https://github.com/AriAlavi/SigNN> (besucht am 06.01.2022).
- [15] Ming Guang Yong. *Google Developers Blog: Object Detection and Tracking using MediaPipe*. Hrsg. von Google Developers. 10. Dez. 2019. URL: <https://developers.googleblog.com/2019/12/object-detection-and-tracking-using-mediapipe.html> (besucht am 05.01.2022).
- [16] mediapipe. *Pose*. 2020. URL: <https://google.github.io/mediapipe/solutions/pose> (besucht am 06.01.2022).
- [17] Martín Abadi u. a. *TensorFlow: A system for large-scale machine learning*. 27. Mai 2016. URL: <https://arxiv.org/pdf/1605.08695>.
- [18] Ronny Votel und Na Li. *Next-Generation Pose Detection with MoveNet and TensorFlow.js*. English. Mai 2021. URL: <https://blog.tensorflow.org/2021/05/next-generation-pose-detection-with-movenet-and-tensorflowjs.html> (besucht am 28.12.2021).
- [19] *Windows ist überall - noch!* 2. Okt. 2021. URL: https://www.itmagazine.ch/artikel/75608/Windows_ist_ueberall_-_noch.html.
- [20] Ayush Pant. *Workflow of a Machine Learning project*. 2019. URL: <https://towardsdatascience.com/workflow-of-a-machine-learning-project-ecldba419b94> (besucht am 20.08.2022).
- [21] Desting Gong. *Top 6 Machine Learning Algorithms for Classification*. 2022. URL: <https://towardsdatascience.com/top-machine-learning-algorithms-for-classification-2197870ff501> (besucht am 18.08.2022).
- [22] Satyam Kumar. *Use Voting Classifier to improve the performance of your ML model*. 2021. URL: <https://towardsdatascience.com/use-voting-classifier-to-improve-the-performance-of-your-ml-model-805345f9de0e> (besucht am 18.08.2022).
- [23] Niharika Pandit. *Yoga Poses Dataset*. English. 14. Okt. 2020. URL: <https://www.kaggle.com/niharika41298/yoga-poses-dataset>.
- [24] Manisha Verma u. a. „*Yoga-82: A New Dataset for Fine-grained Classification of Human Poses*“. In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 2020, S. 4472–4479.
- [25] Katharina Gossmann. *Das kleine Asana-Lexikon – die wichtigsten Asanas*. 2022. URL: <https://www.yogaeasy.de/artikel/asana-lexikon> (besucht am 10.08.2022).

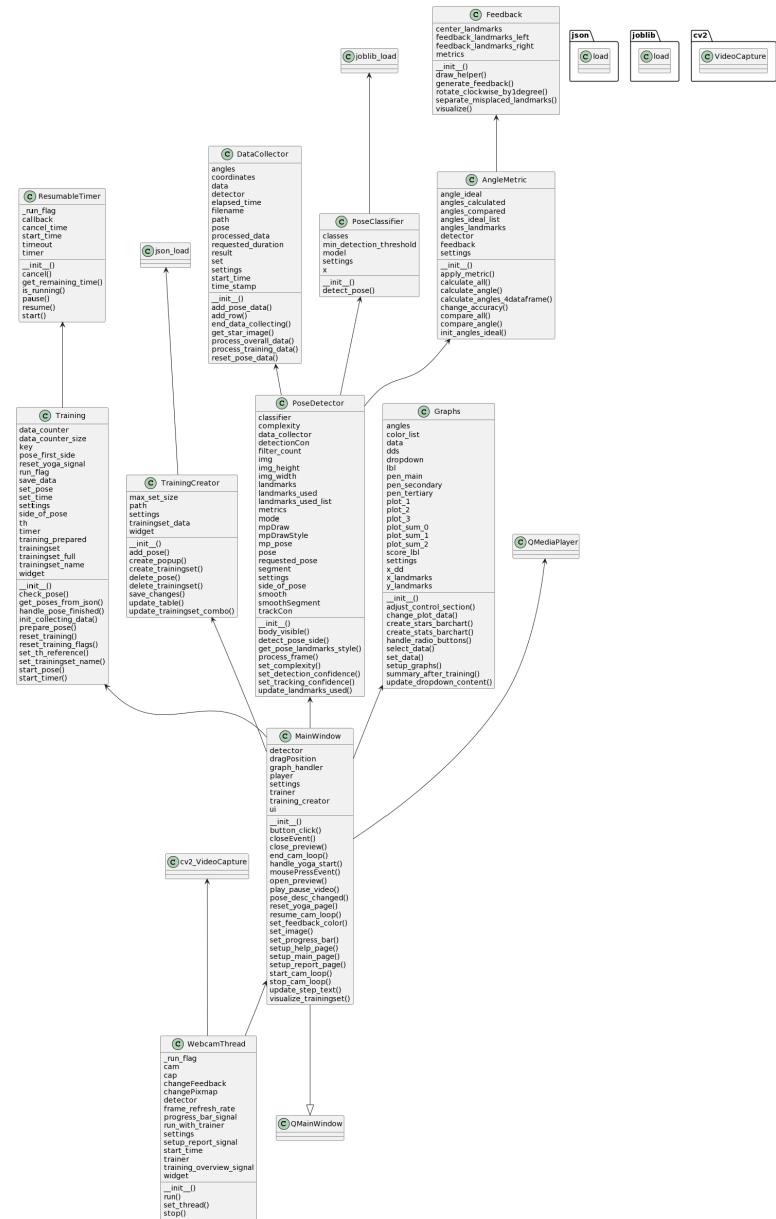
Literaturverzeichnis

- [26] Leonard Richardson. *Beautiful Soup Documentation*. 2020. URL: <https://www.crummy.com/software/BeautifulSoup/bs4/doc/> (besucht am 22.08.2022).
- [27] Muthukrishnan. *Understanding the Classification report through sklearn*. 2018. URL: <https://muthu.co/understanding-the-classification-report-in-sklearn/> (besucht am 15.08.2022).
- [28] Will Koehrsen. *Hyperparameter Tuning the Random Forest in Python*. 2018. URL: <https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74> (besucht am 15.08.2022).
- [29] scikit-learn developer. *Documentation of scikit-learn 0.21.3*. 2019. URL: <https://scikit-learn.org/0.21/documentation.html> (besucht am 22.08.2022).
- [30] Jason Brownlee. *A Gentle Introduction to k-fold Cross-Validation*. 2020. URL: <https://machinelearningmastery.com/k-fold-cross-validation/> (besucht am 22.08.2022).
- [31] PySide. *Python UI | Design GUI with Python | Python Bindings for Qt*. 2022. URL: <https://www.qt.io/qt-for-python> (besucht am 14.08.2022).
- [32] PyQt. *PyQt Reference Guide*. 2022. URL: <https://riverbankcomputing.com/software/pyqt/intro> (besucht am 14.08.2022).
- [33] PyQtGraph. *Welcome to the documentation for pyqtgraph — pyqtgraph 0.12.4.dev0 documentation*. 2022. URL: <https://pyqtgraph.readthedocs.io/en/latest/index.html> (besucht am 14.08.2022).
- [34] Marcelo Duarte. *Welcome to cx_Freeze's documentation! — cx_Freeze 6.11.1 documentation*. 2022. URL: <https://cx-freeze.readthedocs.io/en/latest/> (besucht am 22.08.2022).
- [35] Jordan Russell. *Inno Setup: A free installer for Windows programs by Jordan Russell and Martijn Laan*. URL: <https://jrsoftware.org/isinfo.php>.

Abkürzungsverzeichnis

API	Application Programming Interface
CNN	Convolutional Neural Networks
CPM	Convolutional Pose Machine
CPU	Central Processing Unit
DNN	Deep Neural Network
FPS	Frames Per Second
GPU	Graphic Processing Unit
GUI	Graphical User Interface
HOG	Histogrammorientierte Gradienten
LGPL	Lesser General Public License
ML	Machine Learning
MSE	Mean Squared Error
RGB	Rot-Grün-Blau
ROI	Region of Interest
TPU	TensorFlow Processing Unit
UI	User Interface
WASM	WebAssembly
WYSIWYG	What You See Is What You Get
WebGL	Web Graphics Library

Anhang A: Klassendiagramm



Anhang B

Anhang B: Code Ausschnitte

```
1 def run(self):
2     """
3     Creates a thread for webcam application. cv2 starts a video capture on the
4     main camera and reads its input.
5     If the input is readable, the image will be transformed to an RGB image
6     and adjusted for qt format. The
7     camera will run in an endless loop until it is stopped externally.
8
9     Returns
10    -----
11    None
12    """
13
14    while self._run_flag:
15        # get possible frame size according to dimensions of the gui
16        x_max = self.cam.width()
17        y_max = self.cam.height()
18
19        # get camera input
20        ret, frame = self.cap.read()
21        if ret:
22            rgb_image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
23            # send to pose detection
24            rgb_image, pose_text = self.detector.process_frame(rgb_image)
25            # flip image to mirror user, 1 == y-axis
26            rgb_image = cv2.flip(rgb_image, 1)
27
28            h, w, ch = rgb_image.shape
29
30            bytes_per_line = ch * w
31            convert_to_qt_format = QImage(
32                rgb_image.data, w, h, bytes_per_line, QImage.Format_RGB888)
33            p = convert_to_qt_format.scaled(x_max, y_max, Qt.KeepAspectRatio)
34            self.changePixmap.emit(p)
35
36            # Trainer
37            pose_good = False
38            feedback = None
39            # start trainer
40            if self.run_with_trainer:
41                if self.trainer.run_flag:
42                    pose_good, feedback = self.trainer.check_pose(pose_text)
43                # init the training
44                elif not self.trainer.training_prepared:
45                    # set this flag False when trainingset is finished
46                    self.run_with_trainer = self.trainer.prepare_pose()
47                    # starts training/timer and set the run_flag to True when first
48                    # time in correct pose
49                    elif self.trainer.set_pose == pose_text:
50                        feedback = self.trainer.start_pose(pose_text)
```

Anhang B

```
47     if feedback:
48         pose_text += feedback
49
50     # set pose text
51     self.widget.lbl_visibility.setText(pose_text)
52
53     if pose_good:
54         color = self.settings.value('colors/pose_good')
55     elif self.run_with_trainer and 'fast' in pose_text:
56         color = self.settings.value('colors/pose_almost')
57     else:
58         color = self.settings.value('colors/pose_bad')
59     self.changeFeedback.emit(color)
60
61 else:
62     print("Error: Could not read frame")
```

Listing 5: Mainloop von Yoga AI

```
1 def detect_pose(self, landmarks, requested_pose=None,
2                 min_detection_threshold=None):
3     """
4     Detects the pose.
5
6     Parameters
7     -----
8     landmarks: google.protobuf.pyext._message.RepeatedCompositeContainer
9         Detected landmarks from MediaPipe
10    requested_pose: str
11        Name of the pose which is requested
12    min_detection_threshold: float
13        Threshold which defines at which probability a pose is detected
14
15    Returns
16    -----
17    str
18        Name of the detected pose
19    """
20
21    # extract x and y of landmarks y into the numpy array x
22    index = 0
23    for landmark in landmarks:
24        self.x[0, index] = landmarks[landmark][0]
25        self.x[0, index + 1] = landmarks[landmark][1]
26        index += 2
27    # predict matching probability of every pose
28    y_pred = self.model.predict_proba(self.x)
29    for pose, detection_prob in enumerate(y_pred[0]):
30        # an existing pose detected
31        if detection_prob > self.min_detection_threshold and self.classes[pose]
32        ] in PoseConstants.POSE_LIST:
33            # no requested pose given or matching requested pose
34            if not requested_pose or requested_pose == self.classes[pose]:
```

Anhang B

```
32         return self.classes[pose], True
33     # not matching requested pose
34     else:
35         return i18n.t("noti.wrong_pose"), False
36     # no pose detected
37     return i18n.t("noti.no_pose"), False
```

Listing 6: Funktion zur Klassifizierung einer Yoga-Pose

```
1 def create_stars_barchart(self) -> QChart:
2     """
3     This function creates a barchart with stars.
4     It is used in the statistics page.
5
6     Returns
7     -----
8     QChart
9     The barchart with stars
10    """
11    set_1 = QBarSet('1 Stern')
12    set_2 = QBarSet('2 Sterne')
13    set_3 = QBarSet('3 Sterne')
14    set_4 = QBarSet('4 Sterne')
15    set_5 = QBarSet('5 Sterne')
16
17    # Get unique poses
18    unique_poses = self.data['pose'].unique()
19    # append requested and actual duration of each pose to the set
20    for pose in unique_poses:
21        df = self.data[self.data['pose'] == pose]
22        set_1.append(float(len(df[df['stars'] <= 1.0].index)))
23        set_2.append(float(len(df[(1.0 < df['stars']) & (df['stars'] <= 2.0)].index)))
24        set_3.append(float(len(df[(2.0 < df['stars']) & (df['stars'] <= 3.0)].index)))
25        set_4.append(float(len(df[(3.0 < df['stars']) & (df['stars'] <= 4.0)].index)))
26        set_5.append(float(len(df[(4.0 < df['stars']) & (df['stars'] <= 5.0)].index)))
27
28    series = QStackedBarSeries()
29    series.append(set_1)
30    series.append(set_2)
31    series.append(set_3)
32    series.append(set_4)
33    series.append(set_5)
34
35    chart = QChart()
36    chart.addSeries(series)
37    chart.setAnimationOptions(QChart.SeriesAnimations)
38
39    # Set Poses as x axis
```

Anhang B

```
40 axis = QBarCategoryAxis()
41 axis.append(unique_poses)
42 chart.createDefaultAxes()
43 chart.setAxisX(axis, series)
44 chart.setTheme(QChart.ChartThemeDark)
45 chart.setTitle(i18n.t('noti.graph_4_title'))
46 return chart
```

Listing 7: Erstellung des Säulendiagramms der Sternebewertung

```
1 def handle_pose_finished(self) -> None:
2     """
3         This method is called, as soon as the timer runs out of time (Callback).
4         Should handle the call of next pose or finish the training.
5
6     Returns
7     -----
8     None
9     """
10    # handle poses for both side
11    if self.pose_first_side:
12        print(i18n.t("noti.one_side_finish"))
13        self.run_flag = False
14        self.pose_first_side = False
15        # play sound
16        winsound.PlaySound(self.settings.value('Paths/sounds')
17                            + 'one_side_finished.wav',
18                            winsound.SND_ASYNC | winsound.SND_ALIAS)
19        # reset progress bar
20        self.th.progress_bar_signal.emit(self.set_time, 0)
21        return
22
23    self.trainingset.pop(self.key)
24
25    # play sound
26    winsound.PlaySound(self.settings.value('Paths/sounds')
27                        + 'pose_finished.wav',
28                        winsound.SND_ASYNC | winsound.SND_ALIAS)
29
30    # end collecting data
31    self.th.detector.data_collector.add_pose_data()
32
33    if len(self.trainingset) > 0:
34        self.reset_training_flags()
35    else:
36        try:
37            self.th.detector.data_collector.end_data_collecting(self.save_data)
38        except PermissionError:
39            print('Could not save to file or open file')
40            self.th.setup_report_signal.emit('setup_report_page')
```

Listing 8: Handling verschiedener Posen im Training

Anhang C

Anhang C: Fragebogen

Yoga AI Nutzerumfrage

Wie einfach war die Installation von Yoga AI?

1 2 3 4 5

Schwierig

Einfach

Wie oft bemerken Sie ungewünschtes Verhalten während der Benutzung? (Ruckler, Falschinformationen, etc.)

1 2 3 4 5

Selten

Häufig

War die Benutzung...?

- | | |
|------------------------|---|
| Aufwändig? | <input type="checkbox"/> Ja <input type="checkbox"/> Nein |
| Unterhaltsam? | <input type="checkbox"/> Ja <input type="checkbox"/> Nein |
| Schnell und effizient? | <input type="checkbox"/> Ja <input type="checkbox"/> Nein |

Wie sympathisch finden Sie die Anwendung? Ist sie ansprechend?

1 2 3 4 5

Nicht ansprechend

Sympathisch

Wie gut finden Sie sich in der Anwendung zurecht?

1 2 3 4 5

Schlecht

Sehr gut

Wie unterstützend ist Yoga AI, um Yoga zu lernen/üben?

1 2 3 4 5

Nicht hilfreich

Sehr hilfreich

Würden Sie Yoga AI Anderen weiterempfehlen?

1 2 3 4 5

Unwahrscheinlich

Wahrscheinlich

Bewerten Sie Ihr Gesamterlebnis

1 2 3 4 5

Enttäuschend

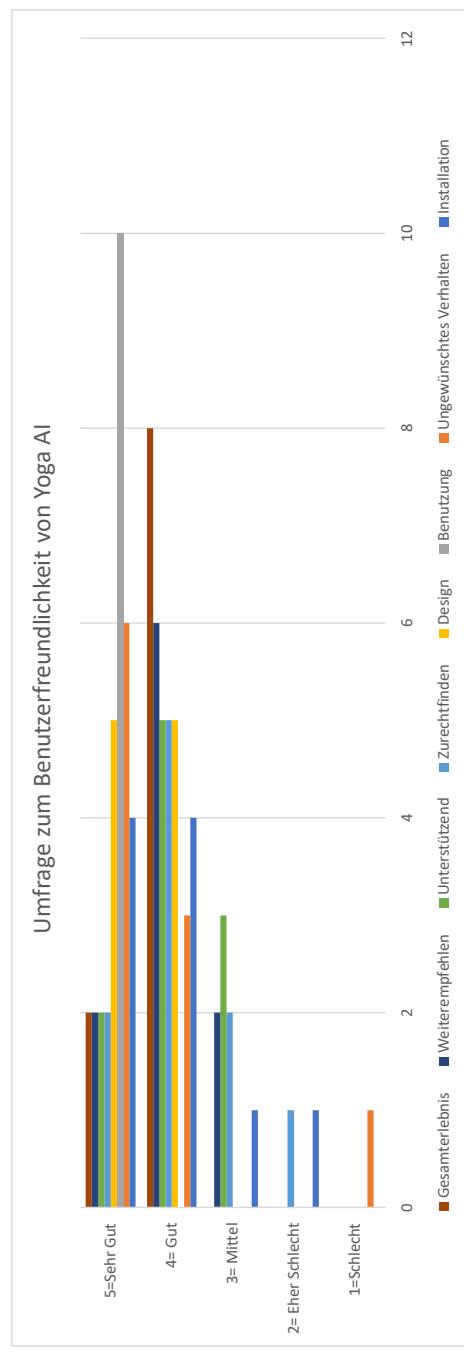
Außergewöhnlich

Wie können wir Yoga AI verbessern?

Anhang D: Auswertung Fragebogen

Fragebogen Auswertung

Frage 1: Nr. Installation	Frage 2: ungewünschtes Verhalten		Frage 3: Benutzung		Frage 4: Design		Frage 5: Zurechtfinden		Frage 6: Unterstützend		Frage 7: Empfehlung		Frage 8: Gesamterlebnis		Frage 9: Verbesserung	
	Frage 1: Installation	Frage 2: ungewünschtes Verhalten	Frage 3: Benutzung	Frage 4: Design	Frage 5: Zurechtfinden	Frage 6: Unterstützend	Frage 7: Empfehlung	Frage 8: Gesamterlebnis	Frage 9: Verbesserung	Frage 10:	Frage 11:	Frage 12:	Frage 13:	Frage 14:	Frage 15:	Frage 16:
1	5	5	5	4	4	3	4	4	4	4	4	4	4	4	4	4
2	4	5	5	5	4	5	4	5	4	4	4	4	4	4	4	4
3	5	4	5	5	3	4	4	5	4	4	4	4	4	4	4	4
4	4	5	5	4	4	4	4	5	4	5	5	5	5	5	5	5
5	3	5	5	5	4	4	4	4	4	4	4	4	4	4	4	4
6	4	1	5	5	4	5	4	5	5	5	5	5	5	5	5	5
7	4	5	5	4	2	3	3	3	3	3	3	3	3	3	3	3
8	5	5	5	4	5	3	3	4	4	4	4	4	4	4	4	4
9	2	4	5	5	5	4	4	3	3	3	3	3	3	3	3	3
10	5	4	5	4	3	4	4	4	4	4	4	4	4	4	4	4



Eidesstattliche Erklärung

Die Verfasser erklären an Eides statt, dass sie die vorliegende Arbeit selbstständig, ohne fremde Hilfe und ohne Benutzung anderer als die angegebenen Hilfsmittel angefertigt haben. Die aus fremden Quellen (einschliesslich elektronischer Quellen) direkt oder indirekt übernommenen Gedanken sind ausnahmslos als solche kenntlich gemacht. Die Arbeit ist in gleicher oder ähnlicher Form oder auszugsweise im Rahmen einer anderen Prüfung noch nicht vorgelegt worden.

St. Gallen, 26.08.2022


.....
Stephan Seliner


.....
Alex Koller