HAW HAMBURG

Bachelorarbeit

Marlon Regenhardt

Analyse von Pose Estimation Sequenzen mit Deep Learning zur Sturzerkennung

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Engineering and Computer Science
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Marlon Regenhardt

Analyse von Pose Estimation Sequenzen mit Deep Learning zur Sturzerkennung

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Angewandte Informatik*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Thomas Lehmann
Zweitgutachter: Prof. Dr. Kai von Luck

Eingereicht am: 21.11.2024

Marlon Regenhardt

Thema der Arbeit

Analyse von Pose Estimation Sequenzen mit Deep Learning zur Sturzerkennung

Stichworte

Sturzerkennung, Neuronale Netze, RGB-Kamera, Pose Estimation, Ambient Assisted Living, LSTM, ConvLSTM

Kurzzusammenfassung

Der demografische Wandel führt zu einer alternden Gesellschaft, während der Pflegenotstand zunehmend kritischer wird. Um die Autonomie älterer Menschen zu fördern und die Pflege zu entlasten, untersucht diese Arbeit die Entwicklung eines effizienten und datenschutzfreundlichen Sturzerkennungssystems, das auf Skelettdaten aus RGB-Kamerabildern basiert. Der Einsatz von Pose Estimation ermöglicht die Extraktion von Bewegungsmustern, die durch neuronale Netze analysiert werden, um Stürze zuverlässig zu identifizieren. Verschiedene Modellarchitekturen, von kleineren ressourcenschonenden bis zu größeren, komplexeren Modellen, wurden evaluiert, um die optimale Balance zwischen Sensitivität und Spezifität zu bestimmen. Die Ergebnisse zeigen, dass selbst kleinere Modelle eine hohe Erkennungsrate erreichen und ohne Cloud-Anbindung eingesetzt werden können, was sowohl die Privatsphäre der Nutzer schützt als auch eine Echtzeiterkennung in häuslicher Umgebung ermöglicht.

Marlon Regenhardt

Title of Thesis

Analysis of Pose Estimation Sequences using Deep Learning for Fall Detection

Keywords

Fall Detection, Neural Networks, RGB Camera, Pose Estimation, Ambient Assisted Living, LSTM, ConvLSTM

Abstract

Demographic change is leading to an ageing society, while the care crisis is becoming increasingly critical. In order to promote the autonomy of elderly people and relieve the burden of care, this thesis investigates the development of an efficient and privacy-friendly fall detection system based on skeletal data from RGB camera images. The use of pose estimation enables the extraction of movement patterns, which are analyzed by neural networks to reliably identify falls. Different model architectures, from smaller resource-efficient models to larger, more complex models, were evaluated to determine the optimal balance between sensitivity and specificity. The results show that even smaller models can achieve a high detection rate and can be used without cloud connectivity, which both protects user privacy and enables real-time detection in a home environment.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Motivation

Im Zuge des demografischen Wandels wird das deutsche Pflegesystem in Zukunft mit neuen Herausforderungen umgehen müssen. Prognosen zeigen, dass bis zum Jahr 2035 eine deutliche Diskrepanz zwischen der Anzahl der Pflegebedürftigen und der verfügbaren Pflegekräfte bestehen wird[18, 26]. Eine isolierte Lebensweise kann bei älteren Individuen das Risiko von Sturzereignissen erhöhen, was sowohl zu physischen Verletzungen führen kann, als auch negative Auswirkungen auf das Selbstvertrauen und die Autonomie haben kann[14, 38].

Während Stürze eine der Hauptursachen für Verletzungen bei älteren Menschen sind[38], bleibt die rechtzeitige Erkennung eines Sturzes eine Herausforderung im häuslichen Umfeld[11]. Herkömmliche Ansätze zur Sturzerkennung, die auf körpergebundenen Sensoren basieren, sind oft unzuverlässig oder werden aus Bequemlichkeitsgründen nicht verwendet. Ein Beispiel hierfür ist der Hausnotruf des Deutschen Roten Kreuzes, der aus einem manuell zu betätigendem Knopf besteht, der an einem Armband oder einer Kette getragen wird[19]. Im Falle eines Sturzes kann das nicht immer gewährleistet sein, beispielsweise wenn die Person bewusstlos ist. Auch kann eingeschränkte Mobilität das Erreichen des Knopfes verhindern.

Aufgrund der durch den demografischen Wandel bedingten Alterung der Gesellschaft und der durch den bereits herrschenden Personalmangel[26] angespannten Bedingungen im Pflegebereich ist es denkbar, dass eine zuverlässige und automatisierte Sturzerkennung von erheblicher Bedeutung sein kann. Das Entwickeln von innovativen Lösungen wird erforderlich sein, um ältere Menschen vor den möglichen Folgen von Unfällen zu schützen bzw. bei Eintreten eines Unfalls eine schnelle Reaktion zur Mitigation von körperlichen und psychischen Schäden zu gewährleisten. Eine verlässliche, automatisierte Lösung kann außerdem dazu beitragen, die psychologische Belastung älterer Menschen zu reduzieren, die durch die Angst vor einem Sturz und den damit verbundenen Folgen entsteht.

1.2 Aufbau der Arbeit

Um diesen Herausforderungen zu begegnen, wird in dieser Arbeit ein automatisiertes System zur Erkennung von Stürzen älterer Menschen in häuslicher Umgebung entwickelt. Der Lösungsansatz basiert auf der Verwendung eines Systems, welches die Körperhaltung einer Person in einem Bild erkennen kann (Pose Estimation), sowie auf neuronalen Netzen, um Stürze zuverlässig und effizient zu erkennen. Bewegungsdaten werden in Skelettvektoren umgewandelt, die dann in ein neuronales Netz eingespeist werden, das auf die Erkennung von Bewegungsmustern von Stürzen trainiert wurde.

Um die Relevanz und Wirksamkeit dieses Ansatzes zu demonstrieren, werden unterschiedlich trainierte Modelle und Kombinationen von Trainingsdaten evaluiert und deren Leistung anhand ihrer Konfusionsmatrix bewertet.

Im Verlauf dieser Arbeit wird das Problem der Sturzerkennung analysiert, der Forschungsstand dazu erläutert und die technischen Details der Implementierung sowie die Ergebnisse und deren Implikationen diskutiert. Es wird diskutiert, wie diese Technologie dazu beitragen könnte, das Wohlbefinden älterer Menschen zu verbessern, und welche Herausforderungen noch zu bewältigen sind, um eine Anwendung in der Praxis zu ermöglichen.

2 Analyse

In diesem Kapitel wird die Sturzerkennung mittels Pose Estimation und Deep Learning detailliert analysiert. Zunächst werden die grundlegenden Konzepte und Herausforderungen der Sturzerkennung eingeführt. Anschließend werden verschiedene Sensorarten und ihre Vor- und Nachteile diskutiert. Der Abschnitt zur Pose Estimation beleuchtet die Erkennung von Körperhaltungen und deren Umsetzung in Koordinatenpunkte. Abschließend wird die Integration von neuronalen Netzen und deren Einsatz für die Sturzerkennung erklärt.

2.1 Sturzerkennung

Die Charakterisierung eines Sturzes kann nicht allein anhand des Vorliegens einer liegenden Position eines Individuums erfolgen. Eine Person kann beispielsweise im Bett liegen, ohne dass ein Sturz stattgefunden hat, oder sich hinknien, um etwas aufzuheben, beziehungsweise aus anderen Gründen hinsetzen. Diese Aktivitäten, die unter dem Begriff „Activities of Daily Living“ (ADLs) zusammengefasst werden, müssen von einer Software zur Sturzerkennung zuverlässig von einem Sturz unterschieden werden können. Darüber hinaus variieren Stürze in ihren Bewegungsmustern erheblich, was zusätzliche Anforderungen an die Automatisierung solcher Systeme stellt. Es ist daher erforderlich, ein System zu entwickeln, das unterschiedliche Sturzereignisse präzise erkennt und gleichzeitig ADLs korrekt als solche klassifiziert.

Hierfür hat Schiemers 2012 zwei grundsätzliche Kategorien von Sensoren vorgeschlagen: Körpergebundene und somit ortsunabhängige Sensoren, „die als ortsunabhängig betrachtet werden können“[28], und „berührungslose, ortsabhängige Methoden“[28], wie Bodensensoren, Kameras oder an Einrichtungsgegenständen wie WC oder Kühlschrank angebrachte Sensoren.

2011 wurden Bodensensoren zur Sturzerkennung vorgeschlagen und ausgiebig getestet[22]. Diese wurden dann 2015 mit einem an der Hüfte getragenen System, Vivid, kombiniert und online ausgewertet, um die Erkennungsrate insgesamt zu verbessern[23]. In der Studie konnte nachgewiesen werden, dass das getestete System in Verbindung mit den installierten Bodensensoren häufig in der Lage war, Stürze zu erkennen. Die Installation von Bodensensoren stellt jedoch eine beachtliche Hürde zur Installation

in vorhandenen Wohnungen oder Pflegeeinrichtungen dar. Außerdem muss das Gerät jederzeit an der Hüfte getragen werden, was sowohl für die pflegebedürftige Person als auch für eine etwaige Pflegekraft schwierig sein kann[28]. Eine pflegebedürftige Person ohne Pflegekraft kann vergessen das Gerät anzulegen oder zu laden. Eine Pflegekraft muss das Gerät warten und sicherstellen, dass das Gerät immer getragen wird. Personen mit Demenz können sich außerdem dagegen wehren, ein solches Gerät zu tragen. Diese Umstände können zu niedriger Akzeptanz und somit zu einer geringen Verbreitung oder Effektivität des Systems führen.

Gegenwärtig sind verschiedene tragbare Geräte zur Sturzerkennung verbreitet. Das sind entweder Geräte, die klein genug sind sie in der Tasche zu tragen[31], oder Smartwatches die entweder konkret für diesen Zweck entwickelt wurden[20] oder allgemeine Smartwatches die durch passende Software dazu befähigt werden, mit ihren vorhandenen Sensoren Stürze zu erkennen wie die Smartwatches von Apple[3], Samsung[27], Google[12] und Huawei[17]. Diese Geräte besitzen den Vorteil, dass sie bereits von einigen Menschen getragen werden und es deshalb denkbar ist, dass eine Nutzung von Smartwatches zur Sturzerkennung an weniger Hürden geknüpft ist, als der Einsatz von eigens konzipierten tragbaren Geräten zur Sturzerkennung.

Körpergebundene Sensoren haben mehrere Nachteile. Zum einen müssen sie ständig getragen werden, was bedeutet, dass man leicht vergessen kann, sie anzulegen. Zum anderen muss regelmäßig sichergestellt werden, dass sie aufgeladen sind. Dies ist besonders bei Smartwatches wie der Apple Watch oder Galaxy Watch problematisch, da sie für den Alltag entwickelt wurden und oft nur eine Akkulaufzeit von einem Tag haben. Berührungslose Sensoren haben den Vorteil, dass sie nicht getragen werden müssen und somit nicht vergessen werden können. Sie haben jedoch den Nachteil, dass sie in der Regel eine Installation erfordern, die nicht immer ohne Weiteres durchführbar ist, wie zum Beispiel bei den Bodensensoren. Ein Vorteil dieser festen Installation ist jedoch, dass sie üblicherweise direkt an den Strom angeschlossen sind und somit nicht aufgeladen werden müssen.

2012 haben Debard et al. mit Kameras durch Pose Estimation und Auswertung der Geschwindigkeit und Winkeländerung der Person Stürze erkannt[9]. Vadivelu et al. schlugen 2017 Sturzerkennung mit Wärmebildkameras vor[33]. Der Einsatz von Wärmebildkameras gestaltet sich vorteilhaft, da sie auch im Dunkeln funktionieren und aufgrund der grundlegenden Technik bereits anonymisierte Bilder liefern.

2.2 Pose Estimation

Die meisten kamerabasierten Ansätze zur Sturzerkennung beginnen mit einer Pose Estimation. Dabei handelt es sich um die Identifikation von Körperhaltungen in Bildern oder Videos sowie deren Kodierung in eine Menge von Koordinatenpunkten, sogenannten Landmarks, die bestimmte Körperteile wie Gelenke oder Augen repräsentieren. Diese

Landmarks beschreiben die Position des erkannten Körpers und können sowohl in zwei- als auch in dreidimensionaler Form errechnet werden. Frühe Lösungen der Pose Estimation, wie die von Debard et al., nutzten anstelle spezifischer Koordinatenpunkte des Skeletts noch vereinfachte Modelle wie Ellipsen oder Silhouetten, um Körperumrisse oder -formen näherungsweise darzustellen. Diese Methode erwies sich als weniger präzise im Vergleich zu den heutigen Verfahren [9].

Aktuelle Ansätze unterscheiden sich unter anderem in der Anzahl der erkannten Personen und der Landmarks pro Person und basieren in der Regel auf einem von zwei Ansätzen: dem Top-Down-Ansatz [24] oder dem Bottom-Up-Ansatz [8].

Im Top-Down-Ansatz wird zunächst ein auf Personen trainiertes Objekterkennungsmodell (Object Detector) eingesetzt, um die Position aller Personen im Bild zu bestimmen. Anschließend erfolgt innerhalb der jeweiligen Bounding Boxes eine detaillierte Erkennung, um die Landmarks, also die Körperteile der Personen, zu bestimmen. Der Bottom-Up-Ansatz hingegen erkennt im ersten Schritt alle Landmarks (z. B. Schultern, Füße, Köpfe, Hände) im gesamten Bild unabhängig von ihrer Gruppierung. Diese Landmarks werden anschließend anhand spezifischer Muster, wie der relativen Position der Gelenke zueinander, zu vollständigen Personen gruppiert. Dabei entstehen beispielsweise Körper mit einem Rumpf, zwei Beinen und zwei Armen, die räumlich nahe beieinander liegen.

Beide Ansätze bieten spezifische Vor- und Nachteile, die sich je nach Anwendungsfall unterscheiden. Beispiele für Projekte, die den Top-Down-Ansatz verfolgen, sind MediaPipe Pose, Pose Transformer und PoseNet. Bekannte Umsetzungen des Bottom-Up-Ansatzes umfassen OpenPose[7], SimpleBaseline[39] und AlphaPose[10].

Eine Pose Estimation liefert eine Beschreibung eines Skeletts durch Landmarks, die relativ zum Bild oder zueinander kodiert werden. Diese Landmarks ermöglichen viele weiterführende Anwendungen, wie beispielsweise Gestenerkennung, Sturzerkennung, oder die Analyse von Bewegungsmustern, ohne Identifizierung der betreffenden Person oder eine Analyse des gesamten Ursprungsbild. Durch diese Abstraktion wird die Komplexität solcher Analysen von einem Bild auf ein oder mehrere Skelette deutlich reduziert und aus darauf basierenden Anwendungen entfernt, wodurch sich diese Anwendungen auf die speziell dafür relevanten Informationen konzentrieren können. Es resultiert im Zuge eine Modularität, die es ermöglicht, die Pose Estimation Lösung durch eine andere zu ersetzen, ohne die darauf aufbauenden Anwendungen komplett anpassen zu müssen, da lediglich das Auslesen der Landmarks angepasst werden muss.

Außerdem gibt es hardwareunterstützte Methoden. Stereo- oder andere Tiefenkameras bieten aufgrund der inhärenten Entfernungserkennung eine effektive Möglichkeit zur präziseren Pose Estimation[5]. Diese Kameras können durch unterschiedliche Techniken die Entfernung von Objekten zum Kamerastandort erkennen und ermöglichen somit

eine genauere Bestimmung der Position von Landmarks im 3D-Raum. Im Vergleich zu monokularen Ansätzen kann die Verwendung von Stereokameras die Genauigkeit der Pose Estimation erheblich verbessern, da durch die verlässlich erkannte Entfernung jedes Teils einer Person das gesamte Skelett präziser beschrieben werden kann. Stereobasierte Methoden nutzen die Disparität zwischen den Bildern der linken und rechten Kamera, um die Entfernung jedes Pixels zur Kamera zu berechnen, was zu einer robusteren Schätzung führt, die weniger anfällig für Mehrdeutigkeiten bei der Objektgröße ist. Eine weitere Technologie sind LiDAR- oder Time-of-Flight-Kameras, die aktuell auf einer von zwei Techniken basieren. Entweder wird bei der Lichtlaufzeitmessung gemessen, wie lang gepulste Laserstrahlen zum Ziel und zurück unterwegs sind. Die andere Technik sendet „einen durchgehenden Wellenimpuls“, misst die Phasendifferenz zum empfangenen, reflektierten Lichtimpuls und berechnet daraus die Entfernung[30]. Diese Kameras sind in der Lage, die Entfernung zu Objekten mit hoher Genauigkeit zu bestimmen und können somit eine präzise 3D-Positionierung von Landmarks ermöglichen, sind jedoch mit erheblichen finanziellen Mehraufwand als andere Arten von Kameras verbunden.

Ein weiterer Ansatz sind KI-Modelle aus neuronalen Netzen, die Landmarks anhand von gelernten Mustern erkennen. Solche KI-Modelle nutzen Daten, welche auf annotierten Datensätzen von Bildern oder Videos beruhen. Für deren Training wurden die Positionen der Landmarks manuell oder auch teilweise automatisiert markiert. Beispiele dafür sind OpenPose, EfficientPose oder MediaPipe[21].

Je nach Anwendungsfall, und gerade beim Einsatz von neuronalen Netzen, kann es sinnvoll sein, keine vorhandene Pose Estimation Lösung zu verwenden, sondern das Bild selbst analysieren zu lassen. Sowohl Tiefenkameras als auch vorhandene Bibliotheken reduzieren die Komplexität der Aufgabe enorm, können aber eben damit auch die Anpassung an spezielle Anforderungen erschweren.

MediaPipe[21] ist eine Bibliothek von Google die ursprünglich 2019 vorgestellt wurde. Sie besteht unter anderen aus einigen Modulen um ein Convolutional Neural Network (CNN), das auf Pose Estimation trainiert ist. Es liegt zum Zeitpunkt des Schreibens dieser Arbeit in Version 10 vor. Es wird auf einem Bild ausgeführt und generiert eine Liste mit 33 Landmarks, die als Vektoren in einem 3D-Raum dargestellt werden können. Sie werden in einem Koordinatensystem relativ zum Standort der Kamera geliefert. Zusätzlich werden die Landmarks außerdem als „World Coordinates“ geliefert, die relativ zur Hüfte der erkannten Person und somit unabhängig vom Standort der Kamera sind. Dies kann für einige Anwendungen nützlich sein. Es ist jedoch für die Sturzerkennung wie sie in dieser Arbeit durchgeführt wurde potenziell weniger effektiv als die kamerabasierten Koordinaten, da bei einem Sturz möglicherweise auch die Bewegung der Person durch den Raum relevant ist, welche bei Nutzung der World Coordinates verloren gehen würde. Da der Nullpunkt des Koordinatensystems mit der Hüfte der Person identisch ist, enthält ein

durch World Coordinates dargestellter Bewegungsablauf keine Informationen darüber, wie sich die Person als Ganzes relativ zum Raum bewegt, in dem sie sich befindet. Deshalb werden in dieser Arbeit die kamerabasierten Koordinaten verwendet, um so viele originale Informationen wie möglich an das zu trainierende neuronale Netz weitergeben zu können. Abbildung 1 demonstriert die von MediaPipe gelieferten Ausgabedaten überlagert auf dem verarbeiteten Bild.

Abbildung 1: Beispiel für Pose Estimation mit MediaPipe[13]. Eine Frau sitzt in meditativer Position auf dem Boden. Die von MediaPipe erkannten Landmarks sind als miteinander verbundene Punkte auf dem Bild dargestellt.

In dieser Arbeit wird eine von einem unabhängigen Programm ausgeführte Pose Estimation verwendet, damit die Analyse der Bewegungsmuster von Stürzen getrennt von der Analyse der Bilder selbst betrachtet werden kann.

2.3 Analyse der Bewegungsmuster

Die Analyse der Bewegungsmuster von Stürzen ist eine Herausforderung, da Stürze unterschiedlich ablaufen können und nicht immer sofort eindeutig sind. Ein Sturz kann aus dem Stehen oder aus dem Sitzen erfolgen und kann auch im Stehen aus unterschiedlichen Positionen erfolgen, indem sich die Person z. B. mit einer Gehhilfe bewegt, an einem Tisch lehnt oder dabei ist, sich hinzusetzen oder aufzustehen. Ein Beispiel für einen Sturz aus dem Sitzen ist in Abbildung 2 dargestellt.

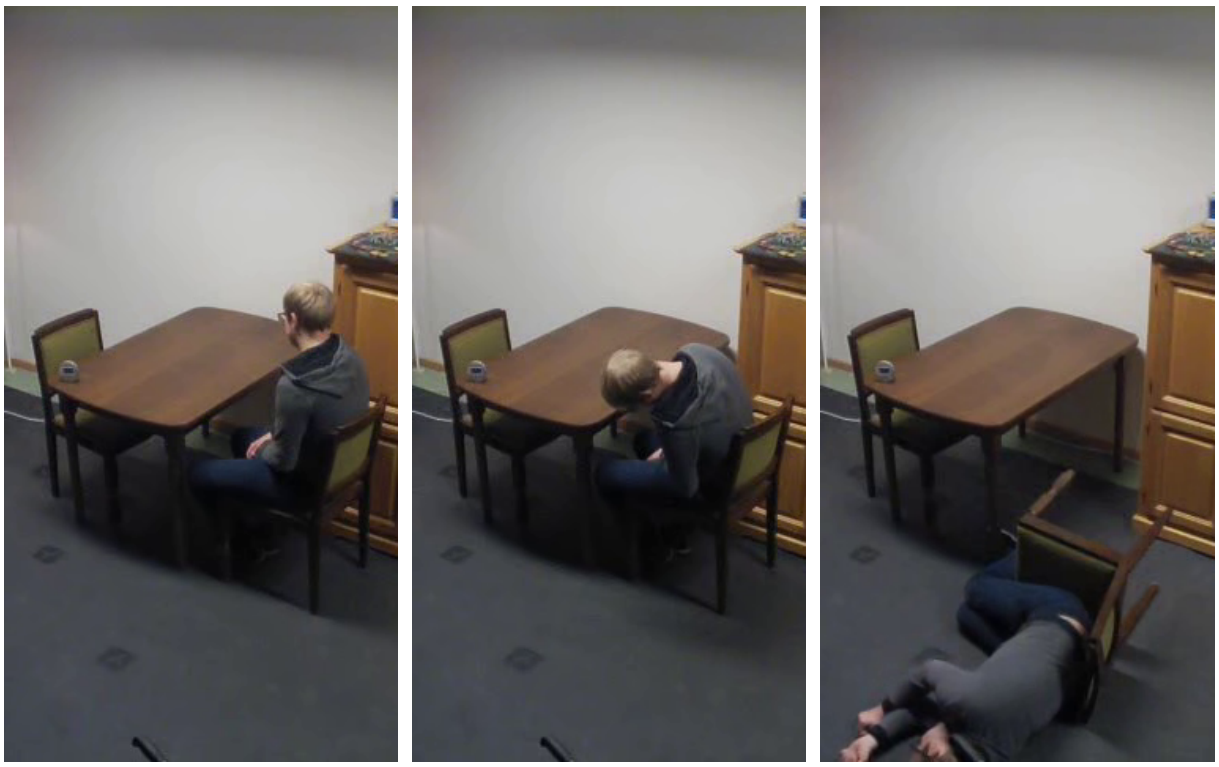


Abbildung 2: Beispiel für einen Sturz aus dem Sitzen. Die Person sitzt auf einem Stuhl und kippt zur Seite.

Stürze können in unterschiedlichen Körperhaltungen und Bewegungskontexten auftreten und beschränken sich nicht ausschließlich auf den aufrechten Stand. Auch im Sitzen, wie beispielsweise bei einer Person, die auf einem Stuhl einschläft und zur Seite kippt, kann es zu einem Sturzereignis kommen. Im Stehen existieren zudem vielfältige Sturzmechanismen, die je nach Situation und Bewegungsablauf variieren. Eine Person kann etwa beim Abstellen einer Gehhilfe stolpern, durch Abrutschen von einer gestützten Position am Tisch das Gleichgewicht verlieren oder beim Hinsetzen den Stuhl verfehlen, falls die Position des Sitzplatzes falsch eingeschätzt wird. Eine visuelle Darstellung eines derartigen Szenarios ist in Abbildung Abbildung 3 in Abschnitt 3.2.1 zu finden.

MediaPipe bzw. generell Pose Estimation liefert wie oben erwähnt die Daten für das gesamte Skelett für jeden Frame mit dem man es speist in Form von Koordinaten für die Landmarks[21]. Es gab auch Ansätze, bei denen Methoden des maschinellen Lernens verwendet wurden, die nicht auf neuronalen Netzen basieren: Vadivelu et al. haben 2017 anhand dieser Daten Bewegungen der Person mit einer Support Vector Machine (SVM) analysiert und als Sturz oder nicht-Sturz eingeordnet[33]. Pires et al. haben 2021 per SVM und K-Nearest-Neighbour (K-NN) jeweils aus einer Pose Estimation unterschiedliche Features extrahiert um so einen Sturz zu erkennen[25]. Schmidpeter hat 2022 durch Analyse der Änderung der Winkel zwischen den Gliedern der Person Bewegungsmuster als Stürze erkannt[29].

Hier mit einem eigenen Algorithmus bzw. mit symbolischer KI zu entscheiden, welche Informationen relevant sind, ist aufgrund der Masse an Informationen schwierig und aufwändig, da beliebig viele unterschiedliche Algorithmen verglichen werden können. Gerade aufgrund der Masse an Daten bietet sich die Anwendung neuronaler Netze an, die aufgrund großer Datenmengen lernen können, bestimmte Gegebenheiten zu erkennen und eine passende Funktion approximieren. Hier genügt es, die gesamten Daten als Input zu geben und diesen mit Labels zu versehen, wodurch das Netz selbstständig relevante Informationen extrahieren kann und lernt, welche Teile des Inputs für die Klassifizierung relevant sind und welche nicht. Ein neuronales Netz kann aus mehreren Schichten bestehen, die jeweils für untschiedliche Aspekte, Dimensionen oder auch Formate der Daten zuständig sind.

Einfache neuronale Netze (Dense) können viele Features verarbeiten, sind aber aufgrund der Verknüpfung aller Neuronen mit allen anderen sehr allgemein und müssten somit sehr groß ausgelegt werden, um temporale oder Räumliche Zusammenhänge zu lernen. Sie setzen jede Information gleichwertig mit allen anderen Informationen in Zusammenhang und extrahieren daraus Muster, haben also kein inhärentes Verständnis davon, dass ein Frame in einem Sturz direkt auf einen anderen folgt, sondern müsste diesen Zusammenhang auch erst lernen[32]. Dies würde zu einer sehr großen Anzahl an Neuronen und Schichten führen, die das Netz komplex und somit langsamer machen würden, als es andere Architekturen ermöglichen.

Ein Recurrent Neural Network (RNN) kann konkret Informationen aus dem vorherigen Verarbeitungsschritt miteinbeziehen und ist somit darauf ausgelegt, explizit temporale Abhängigkeiten erkennen[37]. Stürze geschehen über mehrere Sekunden und somit über eine Vielzahl an Bildern hinweg, was ein RNN besser analysieren kann als ein einfaches neuronales Netz. Allerdings müssen hier aufgrund der vielen Bilder viele Schichten genutzt und auf eine Weise verknüpft werden, auf die sie Informationen über mehrere Inputs hinweg verarbeiten können, was die Komplexität des Netzes enorm erhöht. Es müsste genauer untersucht werden, wie genau ein RNN grundsätzlich konstruiert werden muss, um Stürze zu erkennen.

Eine spezialisierte Variante eines RNN ist das Long Short-Term Memory-Netzwerk (LSTM) [16]. Dieses wurde explizit entwickelt, um temporale Abhängigkeiten über eine größere Anzahl von Zeitschritten hinweg zu erkennen. LSTMs verwenden sogenannte Zellen, die Informationen aus vorherigen Schritten speichern und an die folgenden Schritte weitergeben. Zentral ist dabei der Hidden State, der als Langzeitgedächtnis fungiert und Informationen aus sämtlichen vorherigen Zeitschritten zusammenfasst. Dies ermöglicht es dem Netzwerk, auch nach vielen Verarbeitungsschritten noch auf frühere Zusammenhänge zuzugreifen und diese zu berücksichtigen[36].

Aufgrund dieser Eigenschaften sind LSTMs besonders geeignet, wenn es darum geht, zeitliche Muster über mehrere Zeitschritte hinweg zu erkennen und zu analysieren. Ihr Einsatz bietet sich daher an, um die zeitlichen Abhängigkeiten in Bewegungsabläufen einer Person zu modellieren und darauf basierend Stürze präzise zu identifizieren.

Da jede einzelne Pose zunächst als eine Menge von Koordinaten repräsentiert wird, ist es sinnvoll, vor der Verarbeitung durch ein Long Short-Term Memory-Netzwerk (LSTM) eine oder mehrere Schichten eines Convolutional Neural Network (CNN) einzusetzen. CNNs sind speziell dafür ausgelegt, räumliche Abhängigkeiten in Bildern zu analysieren und zu erkennen[15]. Sie können daher genutzt werden, um die räumlichen Beziehungen innerhalb der Pose einer Person zu identifizieren und relevante Merkmale extrahiert an das LSTM weiterzuleiten.

Um die Fähigkeiten dieser unterschiedlichen Arten von Netzen zu kombinieren, wurden sie für diese Arbeit in einem Netz kombiniert, sodass eine Schicht die Informationen der jeweils vorherigen Schicht auswertet, außer der Eingabeschicht, die die Rohdaten direkt verarbeitet. Ein Beispiel für den Aufbau eines solchen Netzes ist in Abbildung 8 in Abschnitt 3.3 visualisiert. Dies ergibt eine Architektur ähnlich dem ConvLSTM, das bereits in anderen Anwendungsfällen erfolgreich getestet wurde[35].

2.4 Datenquellen

Die Videos, die in dieser Arbeit verwendet werden, müssen spezifische Anforderungen erfüllen, um für die Sturzerkennung geeignet zu sein. Zunächst müssen die Videos auf RGB-Kameras basieren, auf die diese Arbeit ausgerichtet ist. Die Videos sollten sowohl Stürze als auch Alltagsaktivitäten (ADLs) als Negativbeispiele enthalten, um die Modelle auf beide Szenarien zu trainieren und ihre Unterscheidungskraft zu verbessern. Insbesondere sind Videos von Stürzen älterer Menschen in häuslicher Umgebung von Interesse, da diese die Zielgruppe der Sturzerkennungssysteme darstellen. Die Aufnahmen sollten aus der Perspektive von schräg oben, ähnlich einer Deckenkamera, stammen und eine ausreichende Bildqualität für die Pose Estimation aufweisen, die bei zu schlechter Bildqualität ungenau werden könnte. Idealerweise sollten die Videos aus mehreren Kameraperspektiven aufgenommen werden, um die Robustheit der resultierenden Modelle zu erhöhen. Wenn möglich, sollten die Videos zudem nicht nur die Zustände wie Stehen, Sitzen oder Liegen, sondern auch explizit die Stürze selbst labeln, um dies nicht vorher durch eine Person manuell tun zu müssen.

Obwohl eine Vielzahl von Sturzaufnahmen online verfügbar ist, erfüllen die meisten dieser Videos nicht die spezifischen Anforderungen für wissenschaftliche Zwecke. Viele dieser Aufnahmen unterscheiden sich erheblich in ihrer Qualität, sind oft nicht annotiert und zeigen selten Stürze von älteren oder gebrechlichen Personen in einer häuslichen Umgebung. Diese Kriterien sind jedoch entscheidend, da die Modelle gezielt auf realitätsnahe

Szenarien trainiert werden sollen, um eine zuverlässige Erkennung in praktischen Anwendungen zu gewährleisten.

Da das Problem der Sturzerkennung schon länger als solches bekannt ist und daran gefoscht wird, gibt es jedoch auch professionelle Datasets, die explizit für die Sturzerkennung erstellt wurden. Das Multiple Camera Dataset[4] enthält 22 Stürze und 2 ADL-Videos, aufgenommen von je 8 Kameras. Adhikari et al. haben außerdem ein Dataset vorgestellt, das Videos aus 5 Räumen aus je 8 Perspektiven enthält[1]. Beide Datasets haben jedoch keine expliziten Labels für die Stürze, sondern nur für den jeweils aktuellen Zustand der Person wie „stehend", „sitzend" oder „liegend". Dies kann eine zusätzliche Herausforderung bei der Datenverarbeitung darstellen, da die Labels für Stürze manuell hinzugefügt werden müssen, was zeitaufwendig und fehleranfällig sein kann.

Das KUL Dataset[6] enthält 55 von professionellen Schauspielern in einem präparierten Raum nachgestellte Stürze aus je 5 Kameraperspektiven. Die Videos enthalten mit Zeitstempel annotierte Stürze aus stehenden und sitzenden Positionen und beinhalten Gehilfen und Rollstühle, weshalb es im Vergleich mit [4] und [1] besser in den Kontext dieser Arbeit passt. Es enthält außerdem 17 Videos zwischen 11 und 35 Minuten Länge, die nur ADLs enthalten, welche als Negativbeispiele genutzt werden können. Diese detaillierten Annotationen und die Vielfalt der Szenarien machen das KUL Dataset besonders wertvoll für die Entwicklung und das Training von Sturzerkennungssystemen.

2.5 Zielsetzung

Das primäre Ziel dieser Arbeit ist es, ein System zu ermöglichen, das die Zeit vom Eintritt eines Sturzes bis zu dessen Erkennung minimieren kann, um somit eine schnelle Benachrichtigung von Angehörigen, Pflegekräften oder Rettungsdiensten zu ermöglichen. Durch die Verwendung von RGB-Kameradaten soll das System Stürze älterer Menschen in häuslicher Umgebung zuverlässig erkennen können, um unterstützende Maßnahmen zeitnah einzuleiten. Damit soll nicht nur eine schnelle Hilfe gewährleistet, sondern auch das Sicherheitsgefühl und die Unabhängigkeit älterer Menschen in ihrem eigenen Zuhause gestärkt werden. Ein weiteres Ziel ist es, durch das System die psychologische Belastung, insbesondere die Angst vor dem Alleinsein aufgrund des Sturzrisikos, zu lindern und so ein sichereres und selbstbewussteres Leben zu ermöglichen.

Neben der technischen Leistungsfähigkeit wird auch dem Schutz der Privatsphäre der Nutzer besondere Aufmerksamkeit gewidmet. Es soll sichergestellt werden, dass keine unnötigen personenbezogenen Daten gespeichert oder weitergegeben werden. Als sekundäres Ziel wird die ökologische Nachhaltigkeit berücksichtigt, indem die Effizienz der Ressourcen maximiert und der Energieverbrauch reduziert wird. Dies soll insbesondere durch die Optimierung rechenintensiver neuronaler Netze erreicht werden, um den Energiebedarf zu senken, ohne die Leistungsfähigkeit des Systems zu beeinträchtigen.

Durch die Berücksichtigung dieser Anforderungen wird sichergestellt, dass das entwickelte System nicht nur eine hohe technische Leistungsfähigkeit besitzt, sondern auch ethischen und ökologischen Standards genügt, um eine breite Akzeptanz und praktische Anwendbarkeit zu fördern.

2.6 Lösungsansatz

Es wird untersucht, ob durch den Einsatz von Deep Learning ohne spezialisierte Sensoren oder Algorithmen Stürze zuverlässig erkannt werden können. Dabei sollen RGB-Kameras und neuronale Netze zum Einsatz kommen.

Dazu sollen Kamerabilder durch MediaPipe in Skelettvektoren umgewandelt und diese in ein neuronales Netz aus Convolutional und LSTM Schichten eingespeist werden, um sowohl räumliche als auch zeitliche Abhängigkeiten in den Bewegungen zu erkennen. Durch den Einsatz von MediaPipe Pose werden konkrete Bilddaten frühzeitig entfernt und nur die Skelettvektoren weiterverarbeitet, was die benötigte Komplexität des neuronalen Netzes reduziert. Dies entspricht dem Prinzip der Datenminimierung, da nur die notwendigen Daten weiter verarbeitet werden[34]. Neuronale Netze werden auf die Erkennung von Bewegungsmustern von Stürzen aus den öffentlich verfügbaren Videos des KUL-Datasets[6] trainiert. Die Modelle unterschiedlicher Größe werden anschließend evaluiert und ihre Leistung anhand der Confusion Matrix und des Loss Plots bewertet, um das am wenigsten rechenintensive Modell, das die in dieser Arbeit gesetzten Qualitätsanforderungen für die Sturzerkennung erfüllt, zu identifizieren oder eine qualifizierte Aussage darüber zu treffen, ob die Anforderungen überhaupt erfüllt werden können. Erreicht kein neuronales Netz die qualitative Anforderung, soll stichprobenartig untersucht werden, ob durch einfache Heuristiken die Erkennungsrate des Gesamtsystems verbessert werden kann. Mögliche Heuristiken sind, ob während eines Sturzes mehrere Frames hintereinander falsch erkannt werden, oder außerhalb eines Sturzes mehrere Frames hintereinander ein Sturz gemeldet wurde.

3 Design

In diesem Kapitel wird das Design des Systems zur Sturzerkennung detailliert beschrieben. Zunächst werden die genutzten Datenquellen und deren Verarbeitung erläutert. Anschließend wird die Modellarchitektur vorgestellt, die aus einer Kombination von CNN und LSTM Netzwerken besteht, um sowohl räumliche Abhängigkeiten in den Posen als auch zeitliche Abhängigkeiten in den Bewegungen zu erkennen. Der gesamte Prozess der Datenvorverarbeitung, Modellierung und Implementierung wird im Folgenden schrittweise erklärt. Der Aufbau des Codes ist in Abbildung 7 dargestellt.

3.1 Datenquelle

Als Datenquelle wurde das KUL Dataset[6] verwendet. Dafür wurden Stürze in einem Altersheim aufgenommen und hinterher von professionellen Schauspielern nachgestellt. Das Dataset ist öffentlich und frei verfügbar. Es enthält sowohl Videos mit Stürzen, als

auch Videos mit ADLs, die als Negativbeispiele genutzt wurden. Es enthält insgesamt 55 Sturzvideos und 17 ADL-Videos. Die Stürze sind aus jeweils 5 Kameraperspektiven aufgenommen und enthalten sowohl Stürze aus stehenden als auch aus sitzenden Positionen. Die Personen nutzen außerdem Mobilitätshilfen wie Rollatoren oder Rollstühle, um realistische Szenarien zu simulieren. Ein Beispiel ist in Abbildung 3 zu sehen. Für jeden Sturz gibt es 5 Videos, die jeweils mit dem Index der Kamera versehen sind, die das Video aufgenommen hat.

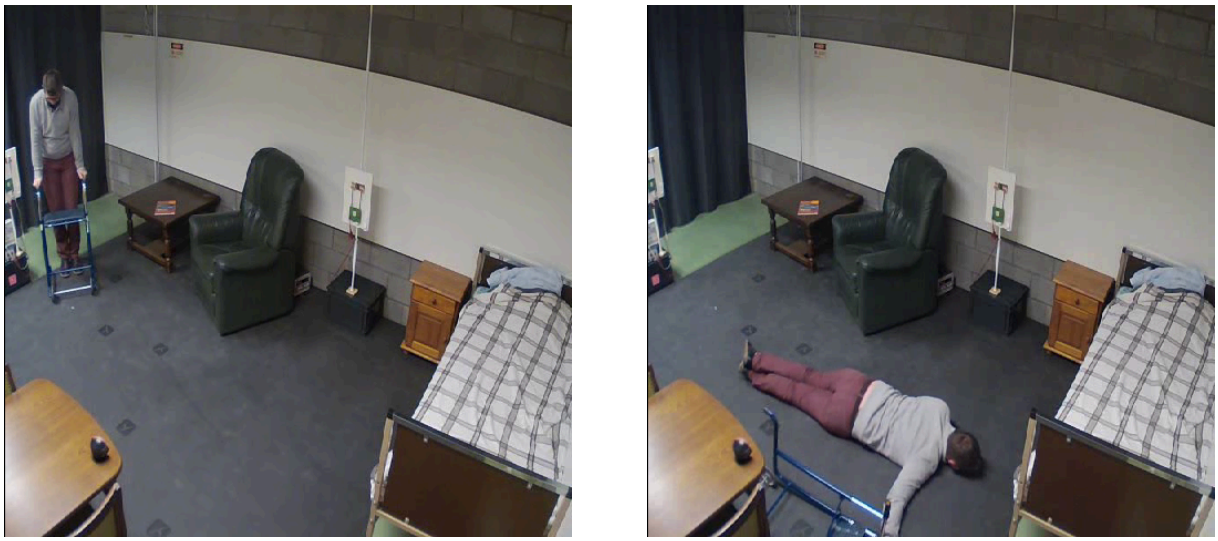


Abbildung 3: Ausschnitte aus dem KUL-Dataset[6]. Links gehend mit Rollator, rechts gestürzt.

Das Dataset enthält eine Excelliste mit Metadaten, die beschreibt, welche Videos es gibt und von wann bis wann der jeweilige Sturz im Video auftritt. ADL-Videos sind ebenfalls beschrieben, jedoch ohne Zeitstempel, da sie keine Stürze enthalten. Um die Videos zu verarbeiten, wurde ein Modul `KUL_data` erstellt, das die Metadaten aus der Excel-Datei in ein für das Programm nutzbares Format umwandelt. Die Klasse kombiniert die Videonamen mit den Labels und Kameraindizes, sodass die Videos bei der Verarbeitung sowohl eindeutig identifiziert als auch mit den entsprechenden Labels versehen werden können. Als Resultat folgt eine Liste von Objekten, die jeweils das Szenario und die Labels beinhalten und Methoden bereitstellen, um die Dateinamen aller Videos des Szenarios zusammenzusetzen und einfach iterierbar zu machen.

Um den Code vom Ablageort der Videos zu trennen, wird der Pfad zu den Videos über eine Umgebungsvariable gesetzt und von diesem Modul bei Ausführung des Programms gelesen. Um zumindest rudimentär sicherzustellen, dass ein Pfad übergeben wurde, wird beim Start des Programms überprüft, ob die Umgebungsvariable gesetzt ist, der Pfad existiert und im gegebenen Verzeichnis Dateien sind. Sollte dies nicht der Fall sein, wird die Ausführung sofort durch eine Exception unterbrochen.

Diese Daten sind lediglich für das Training und die Evaluation der Modelle genutzt worden. Für eine spätere Anwendung in einem realen Szenario müssen Bilder in Echtzeit von einer

Kameras übertragen werden, bei der konkreten Anwendung dient also nicht das KUL Dataset als Datenquelle, sondern die in Echtzeit übertragenen Bilder einer Kamera.

3.2 Vorverarbeitung

Da die Bilder nicht direkt in das neuronale Netzwerk eingespeist werden sollen, sondern nur die daraus extrahierten Skelettdaten, werden die Videos zunächst mit Hilfe eines Moduls `preprocessor` vorverarbeitet, wodurch die Pose Estimation nicht mehr Aufgabe des zu trainierenden neuronalen Netzes ist. Dazu werden die einzelnen Bilder zunächst durch das bereits trainierte und erprobte CNN von MediaPipe Pose analysiert und anschließend augmentiert.

3.2.1 Pose Estimation

Zunächst werden die Videos Bild für Bild durch MediaPipe analysiert. Dazu wird das Modul `MediaPipePose` genutzt, das die Pose Estimation durchführt und die Ergebnisse in Form von Koordinatenpunkten und zusätzlichen Metadaten wie der Information, welche Landmarks als Körper miteinander verbunden sind, liefert. Die Landmarks der jeweils 45 letzten Posen werden in einer Liste gespeichert, die für jedes Bild die Koordinaten der erkannten Landmarks enthält. Die Metadaten werden nicht weiter genutzt, da sie für die Sturzerkennung nicht relevant sind. Wird in einem Bild keine Person erkannt, wird die Liste geleert, um sicherzustellen, dass nur zusammenhängende Sequenzen von Posen weiter verarbeitet werden. Um kurze Aussetzer in der Pose Estimation zu überbrücken, wird jedoch nicht sofort die Liste geleert, sondern erst, wenn für zwei aufeinanderfolgende Bilder keine Person erkannt wurde. Dies soll etwaige Schwachstellen in MediaPipe Pose ausgleichen, die möglichst nicht Teil der Auswertung sein sollten.

Für das Training werden zu jedem Sturzvideo jeweils alle aus dem Sturz extrahierten Sequenzen in einer Datei und die Sequenzen ohne Sturz in einer separaten Datei abgelegt. Die Sequenzen aus den ADL-Videos werden ebenfalls in einer eigenen Datei pro ADL-Video gespeichert. Diese wurden nach dem jeweiligen Video benannt aus dem sie extrahiert wurden, um sie später zuordnen zu können. So konnte der Prozess bei Bedarf neu gestartet werden, ohne bereits verarbeitete Videos erneut durchgehen zu müssen, beispielsweise als aufgrund fehlender Optimierung der Speicher überlief. Durch die Trennung der Daten in Stürze, ADLs aus Sturzvideos und ADLs aus ADL-Videos konnte dasselbe Modell mit unterschiedlichen Kombinationen von Trainingsdaten trainiert und evaluiert werden, um die Auswirkungen der Daten auf die Leistung der Modelle zu untersuchen.

Da beim Echtzeiteinsatz die empfangenen Bilder direkt verarbeitet und anschließend nicht mehr benötigt werden, entfällt die Speicherung der Daten in Dateien. Beim Echtzeiteinsatz des Systems mit einer Kamera werden also aus den empfangenen Bildern direkt durch MediaPipe die Skelettdaten extrahiert und in Sequenzen gepuffert, um sie dann dem neuronalen Netzwerk zur Klassifizierung zu übergeben. Ein Beispiel für die Vorverarbeitung der Bilder ist in Abbildung 4 zu sehen, wobei beispielhaft 3 von 45 Frames gezeigt werden.

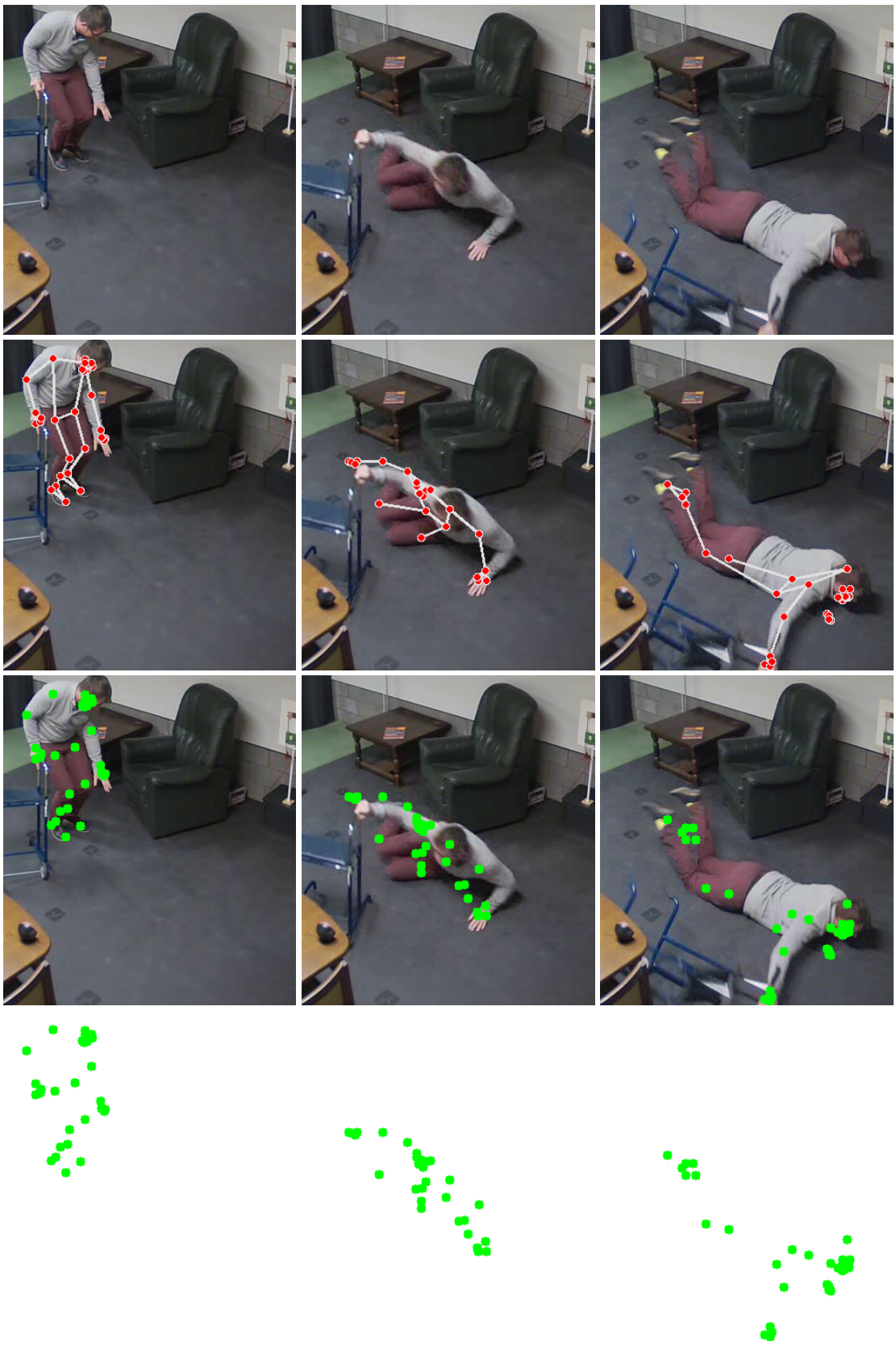


Abbildung 4: Vorverarbeitung der Bilder.

Oberste Reihe: Originalbilder aus den KUL Dataset[6].

Zweite Reihe: Bilder mit den erkannten Posen überlagert wie von MediaPipe geliefert.

Dritte Reihe: die Bilder mit den extrahierten Koordinaten überlagert.

Unterste Reihe: die extrahierten Landmarks ohne Bild.

Zum Training des neuronalen Netzes sollten die Daten einigermaßen zwischen positiven und negativen Beispielen ausbalanciert sein. Da es jedoch weit mehr ADLs als Stürze gibt, mussten die vorhandenen Sturzdaten möglichst effizient genutzt werden. In Abbildung 5 ist dargestellt, wie aus dem vorhandenen Videomaterial zunächst für jeden einzelnen Frame eine entsprechende Sequenz erstellt, um so viele nicht-synthetische Eingangsdaten wie möglich zu extrahieren. Für jeden Frame, den ein Sturz dauert, wurde also eine komplette Sequenz aufgezeichnet, welche jeweils 45 Posen aus den jeweils 45 letzten Frames enthält. Bei 30 FPS sind das also 30 Sequenzen pro Sekunde - für jeden Frame eine eigene Sequenz, extrahiert aus den 45 letzten Frames. Diese detaillierte Erfassung stellte sicher, dass auch kleinste Variationen in den Sturzbewegungen berücksichtigt werden, was die Robustheit der resultierenden Modelle erhöht.

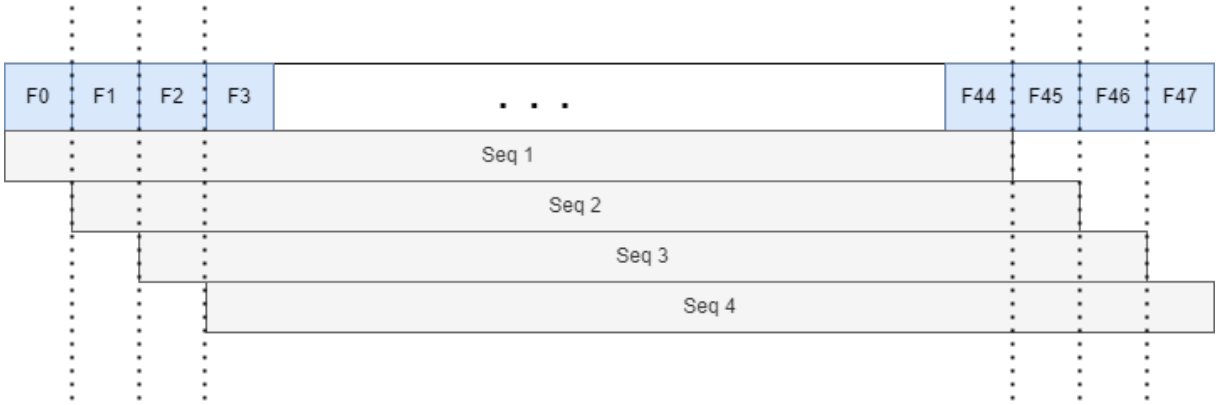


Abbildung 5: Aus Stürzen wird eine Sequenz pro Frame extrahiert.

Beim Extrahieren von Sequenzen für ADLs hingegen ist es nicht notwendig oder sinnvoll, für jeden Frame eine eigene Sequenz zu erstellen, da es eine Vielzahl an ADLs gibt und diese üblicherweise langsamer ablaufen als ein Sturz. Aus den ADLs wurde also die entsprechenden Sequenzen ohne Überlappung extrahiert, um eine klare und eindeutige Repräsentation jeder Aktivität zu gewährleisten und damit möglichst viele verschiedene Aktivitäten, die keine Stürze sind, in die Trainingsdaten einfließen. Der Extrahierugnsprozess der ADLs ist in Abbildung 6 demonstriert. Dieser Ansatz verhindert, dass Trainingsdaten redundante Informationen enthalten und sorgt dadurch für präzisere Ergebnisse bei der späteren Klassifizierung der Bewegungen.

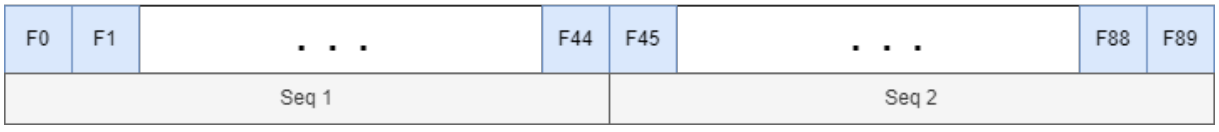


Abbildung 6: Sequenzen von ADLs werden ohne Überlappung extrahiert.

Als zentraler Bestandteil der Vorverarbeitung wurde eine eigene Klasse `PoseProcessor` erstellt. Die Hauptaufgabe dieser Klasse besteht darin, die Skelettdaten aus jedem Frame der Videoaufnahmen zu extrahieren und diese Daten für die weitere Verarbeitung im neuronalen Netzwerk vorzubereiten. Dazu wird in jedem Frame mithilfe von MediaPipe

ein Skelett extrahiert, das aus 33 Landmarks besteht. Diese Skelettdaten werden in einem Puffer gespeichert, der die jeweils letzten 45 Frames umfasst, um die dynamischen Bewegungsmuster über die Zeit zu erfassen. Der Puffer fungiert wie ein Schiebefenster, das sich mit jedem neuen Frame aus dem Video oder Kamerastream um ein Element nach vorne bewegt. Die Größe des Puffers kann variiert werden, da Videos und Kamerastreams in unterschiedlichen Bildraten vorliegen können. Die Daten im Puffer können zu jedem Zeitpunkt normalisiert ausgelesen werden, um sie als Eingabe für das neuronale Netzwerk zu verwenden. Diese Vorverarbeitung ermöglicht es, die Skelettdaten in Sequenzen zu organisieren und sie in einem einheitlichen Format für das Training und die Inferenz bereitzustellen.

3.2.2 Augmentierung der Trainingsdaten

Trotz der frameweise extrahierten Sturzdaten gibt es weit weniger Sturzsequenzen als ADL-Sequenzen. Zunächst wurde also festgelegt, dass sowohl die Trainingsdaten, als auch die Eingangsdaten während der Inferenz normalisiert werden, um Vielfalt in den Inferenzdaten zu reduzieren, wodurch die Modelle trotz weniger Trainingsdaten besser generalisieren können. Dazu wurden die Sturzsequenzen zunächst normalisiert, um sicherzustellen, dass diese möglichst zentral im Bild und in etwa gleich groß sind. Indem dies mit den Echtzeitdaten bei der Inferenz auch gemacht wird, kann dafür gesorgt werden, dass auch die Inferenzdaten möglichst ähnlich zu den Trainingsdaten sind und somit weniger Trainingsdaten benötigt werden. Dabei fiel auf, dass nach dieser Normalisierung die Sturzsequenzen relativ in der Mitte des Bildes zentriert waren, jedoch nur etwa 50% der Bildfläche einnahmen und nicht immer erwartungsgemäß akkurat zentriert waren. Dies würde dazu führen, dass einerseits die normalisierten Trainingsdaten unterschiedlicher ausfallen als erwartet, was dem Modell beim Generalisieren helfen könnte. Andererseits könnte es auch dazu führen, dass die Eingangsdaten während der Inferenz nicht so gut normalisiert werden wie zunächst angenommen, was wiederum die Generalisierungsfähigkeit des Modells einschränken könnte. Gleichzeitig ermöglicht es jedoch, die Trainingsdaten nach der Normalisierung zu augmentieren, indem jede Sequenz horizontal und vertikal leicht verschoben wird, um synthetische, aber plausible Trainingsdaten zu generieren. Diese Augmentierung sorgt dafür, dass viel mehr plausible Trainingsdaten zur Verfügung stehen, was die Robustheit und Genauigkeit der resultierenden Modelle verbessert.

Es wurde darauf verzichtet, Körperelemente wie Unterarme oder Oberschenkel künstlich zu modifizieren (also zu verlängern oder verkürzen), um zusätzliche Datenvariationen zu erzeugen, da unterschiedlich große Körperelemente potenziell den Ablauf des Sturzes beeinflussen würden und die so erzeugten künstlichen Sequenzen nicht der Realität entsprechen würden.

Um das Training des Modells zeitlich und logisch von der Extraktion zu entkoppeln, wurde zusätzlich ein Modul `preprocessor` erstellt. Dies iteriert über die Metadaten des Datasets und führt sie dem `VideoProcessor` zu, der die Videos einliest und aus den Frames jeweils mit Hilfe des `PoseProcessor`s die Skelettdaten extrahiert und in Sequenzen organisiert. Die so generierten Sequenzen werden in Dateien gespeichert, die dann für das Training des neuronalen Netzwerks genutzt werden können. Der `preprocessor` und der `VideoProcessor` sind idempotent gestaltet, sodass bereits verarbeitete Videos nicht nochmal verarbeitet werden müssen. Dies ermöglicht es, den Vorverarbeitungsprozess bei Bedarf zu wiederholen, ohne dass bereits verarbeitete Daten erneut verarbeitet werden müssen, oder bei unerwartet hoher Verarbeitungszeit den aktuellen Prozess zu unterbrechen und zu einem späteren Zeitpunkt wiederaufzunehmen. Beim Aufrufen der Skelettdaten vom `preprocessor` für das Training, werden sie vor der Rückgabe an den Aufrufer mit einer `random_shift` Funktion augmentiert, die die Sequenzen horizontal und vertikal randomisiert verschiebt. Dies sorgt für zusätzliche Variationen in den Trainingsdaten, indem unterschiedliche Kameraspositionen simuliert werden.

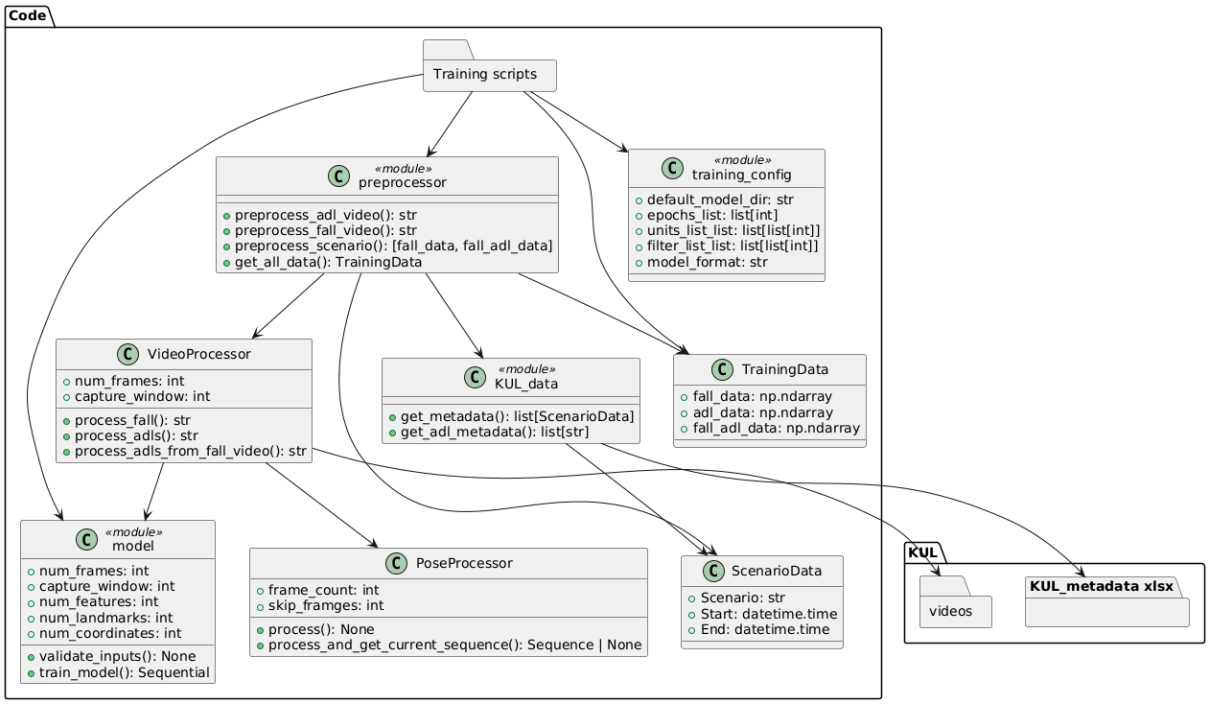


Abbildung 7: Organisation des Codes zur Verarbeitung der Daten aus dem KUL-Dataset und dem Training des neuronalen Netzes.

3.3 Aufbau des neuronalen Netzes

Um alle Parameter und Hyperparameter bezüglich des neuronalen Netzes zu speichern und zu verwalten, wurde ein Modul `model` erstellt. Es kapselt die Architektur für die Konstruktion des konkreten neuronalen Netzes und die Definition der Hyperparameter. Dies soll Fehler während Training und Inferenz vermeiden, indem die Dimensionen der Eingabedaten und die Validierung der Eingabedaten zentral festgelegt werden. Es soll außerdem sicherstellen, dass die Modelle korrekt erstellt, gespeichert und geladen

werden können. Das Modul enthält die Dimensionen der Eingabedaten und eine Funktion zur Validierung der Eingabedaten die als Teil der Trainingsfunktion aufgerufen wird. Die Trainingsfunktion reduziert bei Bedarf die Dimensionen der Eingabedaten, falls sie wider Erwarten dreidimensional übergeben werden. Um anfängliches Prototyping zu erleichtern, wurde das Erstellen des Modells in einer eigenen Funktion gekapselt, welche die Anzahl und Dimensionierung von Convolutional und LSTM Schichten, sowie die Anzahl der Neuronen im Dense Layer, als Parameter akzeptiert. Dies ermöglicht die Flexibilität, verschiedene Modelle mit unterschiedlichen Architekturen zu testen, ohne den Code für das Modell selbst ändern zu müssen, was Leichtsinnsfehler des Autors beim Schreiben der Traingsskripte vermeidet.

Für das neuronale Netz wurde das Framework `TensorFlow` eingesetzt. Zur Implementierung mehrerer aufeinanderfolgender Schichten wurde das Modell `Sequential` verwendet, das als Basis diente. Dadurch können die Convolutional- und LSTM-Schichten in einer Reihenfolge definiert werden, die es ermöglicht, die räumlichen und zeitlichen Abhängigkeiten der Skelettdaten zu analysieren. Diesem `Sequential`-Basismodell wurden als Erstes eine oder mehrere Convolutional-Schichten hinzugefügt, die jeweils in einer TimeDistributed-Schicht verschachtelt wurden. Deren Aufgabe ist es, die räumlichen Abhängigkeiten in jeder einzelnen Pose der Eingabedaten zu analysieren. Durch das Verschachteln in eine TimeDistributed-Schicht wird für jeden Frame, also Zeitschritt, der Eingabedaten die Convolutional-Schicht mit denselben Gewichten angewendet und trainiert. Dies ermöglicht der Convolutional-Schicht das Erlernen und Extrahieren der entscheidenden Features aus der aktuellen Körperhaltung der Person, unabhängig davon, zu welchem Zeitpunkt innerhalb eines Sturzes ein Frame ist. Anschließend wurden eine oder mehrere LSTM-Schichten eingesetzt, um die zeitlichen Abhängigkeiten zwischen den Frames zu analysieren. Die LSTM-Schichten sollen aus der Kombination der vorher extrahierten Features der jeweiligen Körperhaltung mehrerer Frames entsprechende Features des Bewegungsablaufs erkennen. Abschließend wurde eine Dense-Schicht mit einem einzelnen Ausgabe-Neuron verwendet, um die Ausgabe des neuronalen Netzes als Sturz oder Nicht-Sturz zu klassifizieren. Vor der ersten Convolutional-Schicht wurde eine `Reshape`-Schicht eingesetzt, die drei Koordinaten pro Landmark auf zwei Koordinaten pro Landmark reduziert, um die Dimensionen der Eingabedaten zu reduzieren, falls diese in 3D übergeben werden. Auf die letzte Convolutional-Schicht folgt eine `Reshape`-Schicht, die die Dimensionen der Ausgabe der Convolutional-Schicht dahingehend anpasst, dass sie von der Convolutional-Schicht in die LSTM-Schicht übergeben werden kann. Ein Beispielnetzwerk mit drei Convolutional-Schichten und zwei LSTM-Schichten ist in Abbildung 8 dargestellt.

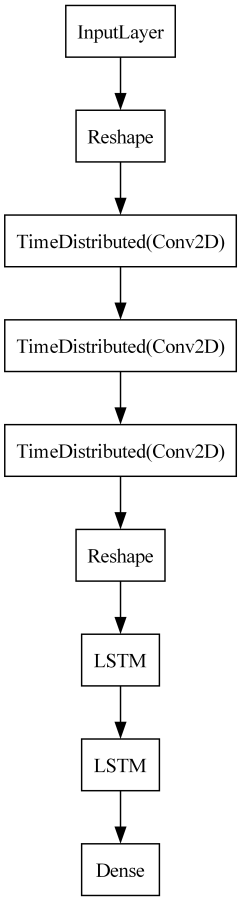


Abbildung 8: Modellarchitektur eines neuronalen Netzes mit 3 Convolutional-Schichten und 2 LSTM-Schichten wie es in dieser Arbeit trainiert und evaluiert wurde.

Um die Modelle voneinander zu unterscheiden wurde eine Nomenklatur eingeführt, die abhängig von der Art, Anzahl und Reihenfolge der Schichten ist. Die Basis sind Buchstaben, die jeweils für eine Schichtart stehen:

`c` für Convolutional, `l` für LSTM. Für `Dense` wurde kein Buchstabe gewählt, da es in jedem Model genau dieselbe, einzelne Dense-Schicht mit einzelnem Ausgabe-Neuron gibt. Eine Zahl vor dem Buchstaben gibt an, wie viele Schichten dieser Art an dieser Stelle sind.

Auf eine Deklaration einer Art von Schicht folgen Deklarationen der Größe der Schichten: Ein `f` für die Anzahl der Filter in einer Convolutional-Schicht, ein `u` für die Anzahl der Einheiten (Units) in einer LSTM- oder Dense-Schicht.

Die Anzahl der Schichten und die Größe der Schichten sind durch Unterstriche getrennt.

Der Modellname `2c_16f_8f_2l_100u_50u` steht beispielsweise für ein Modell mit 2 Convolutional-Schichten, die erste mit 16 und die zweite mit 8 Filtern, dann 2 LSTM-Schichten mit 100 und 50 Einheiten.

Diese Konvention wird von einem Modul `model` verwendet, um anhand der übergebenen Anzahl und Größe der Schichten das Modell zu benennen. Dies ermöglicht es, die Modelle eindeutig zu identifizieren und zu vergleichen, ohne dass die Architektur des Modells selbst ausgelesen werden muss.

3.4 Training

Um den zeitlichen Aufwand für potenziell ineffektive Trainingsläufe großer Modelle mit umfangreichen Datensätzen zu minimieren, wurde das Training in zwei Phasen unterteilt. Zunächst wurden kleinere Modelle mit den Daten eines einzelnen Sturzes trainiert, um festzustellen, ob die Daten und das Design des Modells grundsätzlich so nutzbar sind und um eine generelle Richtung für die Dimensionierung des Modells zu finden. Diese kleinen Modelle besitzen alle ca. 20 000 Parameter. Nachdem die kleinen Modelle bereits mit wenig Training Anzeichen einer Unterscheidung zwischen Sturz und Nicht-Sturz zeigten, wurden größere Modelle dieser Architektur mit ca. 180 000, 280 000 sowie 1 000 000 Parametern trainiert. Das größte der Modelle, mit 3 großen Convolutional-Schichten, 2 großen LSTM-Schichten und einer Dense-Schicht (3c_128f_64f_32f_2l_100u_50u), wurde außerdem mit unterschiedlichen Kombinationen der Trainingsdaten trainiert und liegt somit in 4 Varianten vor. Alle Modelle wurden mit dem Adam-Optimizer und Binary Crossentropy als Loss-Funktion trainiert. Die Modelle wurden für 50 Epochen trainiert, um sicherzustellen, dass sie hinreichend Zeit haben, die Bewegungsmuster zu lernen. Um sie später zu unterscheiden, wurde der Name der Varianten des größten Modells jeweils um ein Suffix ergänzt. Dieses Modell wurde mit den folgenden Varianten trainiert:

- Die erste Variante wurde mit allen verfügbaren Trainingsdaten trainiert, sowohl aus den Sturz- als auch aus den ADL-Videos (`_all_data`).
- Die zweite Variante wurde nur mit den Sturzdaten aus den Sturzvideos und den ADLs aus den ADL-Videos (explizite ADLs) trainiert, ohne die impliziten ADLs aus den Sturzvideos (`_without_implicit_adls`).
- Die dritte Variante wurde nur mit den Sturzdaten und den impliziten ADLs trainiert, also Bewegungsabläufen aus den Sturzvideos, die zu keinem Sturz führen. Hier wurden die expliziten ADLs aus den ADL-Videos nicht genutzt (`_without_explicit_adls`).
- Die vierte Variante wurde nur mit den impliziten und expliziten ADLs aber ohne Sturzdaten trainiert, um zu sehen, ob das Modell auch ohne Sturzdaten Stürze erkennen kann, was es ermöglichen würde, das Modell für reale Szenarien trainieren zu können, ohne Sturzdaten zu benötigen. Dies würde einer Anomaly Detection entsprechen, bei der das Modell Stürze erkennt, ohne jemals einen Sturz gesehen zu haben (`_only_adls`).

Eine schematische Darstellung des Trainingsablaufs ist in Abbildung 9 dargestellt. Die Auswahl der Modelle und die Kombinationen der Trainingsdaten wurden jeweils im Trainingsskript festgelegt.

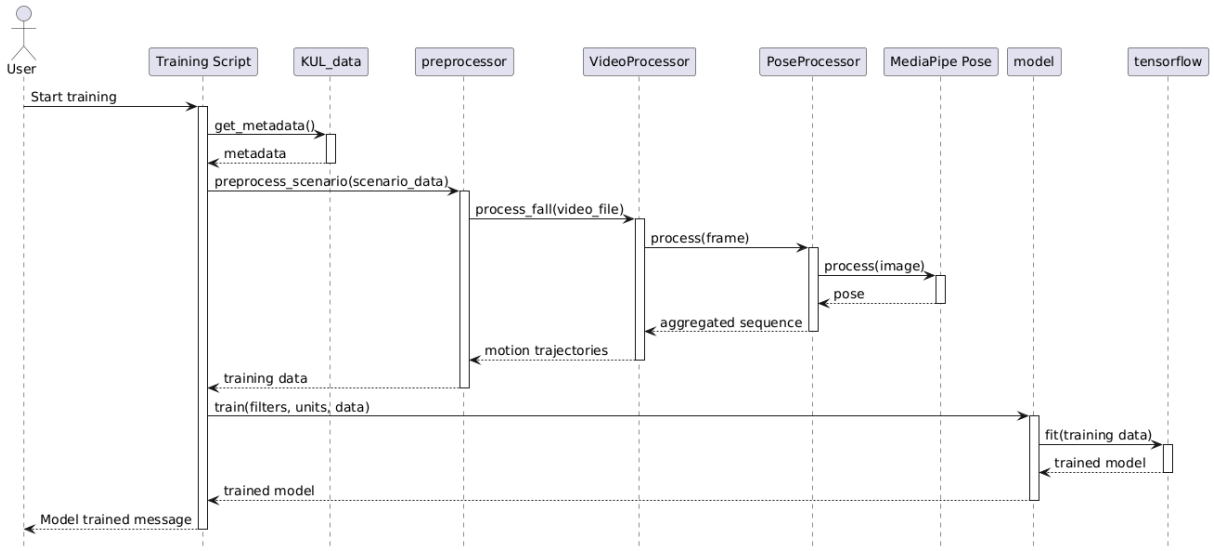


Abbildung 9: Ablauf des Trainings

3.5 Inferenz

Zur Benutzung eines Modells für diese Arbeit wurde eine Funktion geschrieben, die per cv2 nach und nach Bilder aus einem gegebenen Stream eines aus einer Datei gelesenen Videos liest. Somit müssen nicht, wie bei einem Produktiveinsatz, Livedaten von einer Kamera empfangen werden. Diese Bilder werden dann jeweils in einen eigens erstellten `PoseProcessor` gegeben, der die Skelettdaten mit Hilfe von MediaPipe Pose extrahiert und in eine Sequenz aggregiert. Der `PoseProcessor` puffert intern die letzten 45 Posen und gibt diese nach jedem Aufruf als Sequenz zurück, wenn alle 45 Posen extrahiert wurden. Wird in zwei aufeinander folgenden Frames keine Person erkannt, wird der Puffer geleert, um sicherzustellen, dass nur zusammenhängende Sequenzen weiterverarbeitet werden. In diesem Fall gibt der `PoseProcessor` erst wieder eine Sequenz zurück, wenn wieder für 45 Frames eine Person erkannt wurde. Die Sequenzen werden dann in das Modell eingespeist und die Ausgabe des Modells als Sturz oder Nicht-Sturz klassifiziert. Die Ausgabe der Inferenz der jeweils vorangegangenen 45 Posen wurde für diese Arbeit als Text auf dem nebenher gezeigten Video angezeigt, um die Inferenz zu visualisieren, wie in Abbildung 10 beispielhaft gezeigt.

Abbildung 10: Beispiel für die Visualisierung der Inferenz durch das Modell `3c_128f_64f_32f_2l_100u_50u_without_implicit_adls` anhand eines Videos aus dem KUL-Dataset[6]. Links ein Nicht-Sturz, rechts ein Sturz.

In Abbildung 11 ist der Generelle Ablauf zur Sturzerkennung dargestellt, deren Videoquelle in diesem Fall ein Video aus dem KUL-Dataset[6] ist und deren Konsument die Anzeige des Inferenzergebnisses auf dem Videobild ist.

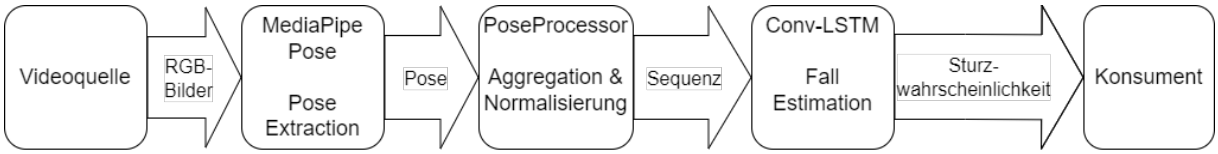


Abbildung 11: Pipeline zur Sturzerkennung

3.6 Evaluierung

Für die Auswertung wurden die bereits mithilfe von MediaPipe Pose extrahierten und durch den `PoseProcessor` aggregierten Daten aus dem `preprocessor` genutzt. Dadurch war es nicht notwendig, die Videos erneut mit MediaPipe Pose vorzuverarbeiten, da die entsprechenden Daten bereits in zuvor generierten `.npy`-Dateien vorlagen, die als `Numpy`-Arrays abgespeichert wurden. Die Evaluation basierte auf den vorhandenen Sequenzen und den während der Vorverarbeitung zum Training zugewiesenen Labels. Eine erneute Wiedergabe der Videos war daher nicht erforderlich.

Eine Sequenz – bestehend aus 45 Posen, die einen Bewegungsablauf einer Person darstellen – wurde nur dann dem Modell zur Inferenz übergeben, wenn diese zuvor vollständig durch MediaPipe Pose im Video erkannt worden war. Partielle Sequenzen wurden, wie bereits während des Trainings, ausgeschlossen. Die Modellausgabe wurde mit dem Label der Sequenz verglichen, wobei die korrekt und falsch klassifizierten Sequenzen gezählt wurden. Zur Visualisierung der Modellergebnisse wurde eine Konfusionsmatrix erstellt.

Die Messung der Inferenzzeit erfolgte, indem jeweils ein Modell geladen und auf die ersten 1000 von MediaPipe Pose erkannten und anschließend aggregierten Sequenzen eines Videos angewendet wurde. Für jedes Modell kamen dieselben 1000 Sequenzen aus demselben Video zum Einsatz. Die Zeit für jede einzelne Inferenz wurde gemessen

und anschließend der Durchschnittswert berechnet. Die Tests wurden auf einem Laptop mit einem integrierten Radeon 780M Grafikprozessor[2] durchgeführt, wobei möglichst wenige andere Programme ausgeführt wurden. Um die Ergebnisse nicht durch externe Faktoren zu beeinflussen, wurde der Laptop während der Messungen nicht anderweitig genutzt. Im selben Testlauf wurde zudem die durchschnittliche Inferenzzeit von MediaPipe Pose gemessen, um die Leistung mit anderen Geräten und Umgebungen vergleichen zu können.

4 Ergebnisse

In diesem Kapitel wird zunächst die Leistung der „kleinen“ Modelle (unter 100 000 Parameter) beschrieben, die jeweils mit allen Daten trainert wurde. Danach wird auf die Leistung der größeren Modelle (über 100 000 Parameter) eingegangen, die auch jeweils mit allen Daten trainiert wurden, und abschließend werden diese miteinander verglichen. Anhand unterschiedlich trainierter Varianten eines großen Modells (1 000 000 Parameter) wird außerdem die Leistung verschiedener Modellvarianten verglichen und bewertet.

4.1 Leistung der kleinen Modelle

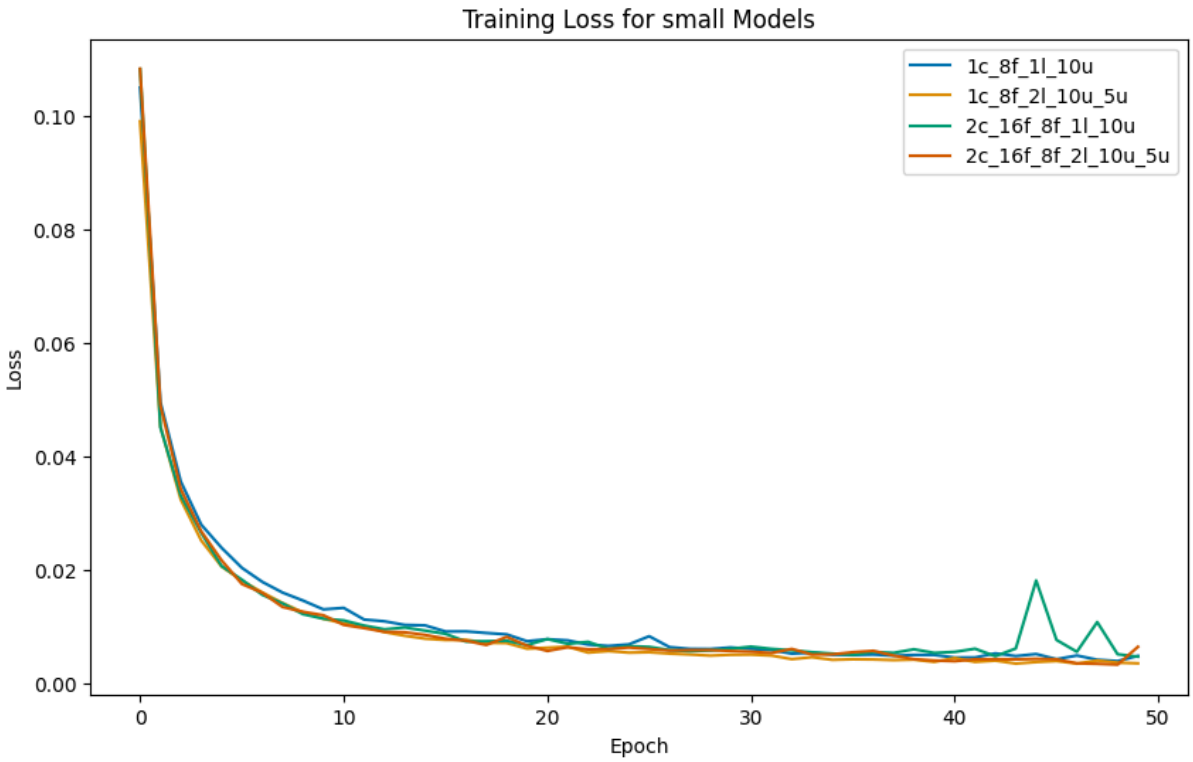


Abbildung 12: Trainingsverlust (loss) der kleinen Modelle

In Abbildung 12 ist zu sehen, dass bereits nach 6 Epochen alle kleinen Modelle einen Trainingsverlust von unter 0,02 erreicht haben. Nach 11-15 Epochen fiel der Verlust sogar auf unter 0,01, was darauf hinweist, dass die Modelle trotz ihrer geringen Komplexität die Bewegungsmuster in den Skelettdaten schnell erlernen konnten.

Das Modell `1c_8f_1l_10u`, welches nur eine Convolutional- und eine LSTM-Schicht hat, zeigt im Vergleich zu den anderen Modellen eine etwas langsamere Lernkurve. Die anderen

kleinen Modelle, die alle mindestens eine zusätzliche Schicht besitzen, lernen etwas schneller als das kleinste Modell, zeigen jedoch in den mittleren Epochen keine wesentlichen Unterschiede zueinander.

In späteren Epochen (ab Epoche 40) verbessern sich die meisten Modelle nur noch minimal. Eine bemerkenswerte Ausnahme stellt das Modell `2c_16f_8f_1l_10u` dar, das in Epoche 44 einen sprunghaften Anstieg des Traingsverlustes bis auf 0,018 in verzeichnet. Bis Epoche 50 fällt es jedoch auch wieder auf 0,0046.

Die Verlustwerte in Epoche 50 sind in Tabelle 1 dargestellt.

Modell	**Parameteranzahl**	**Verlust in Epoche 50**
1c_8f_1l_10u	21651	0,0049
1c_8f_2l_10u_5u	21966	0,0035
2c_16f_8f_1l_10u	22891	0,0046
2c_16f_8f_2l_10u_5u	23206	0,0064

Tabelle 1: Trainingsverlust der kleinen Modelle in Epoche 50

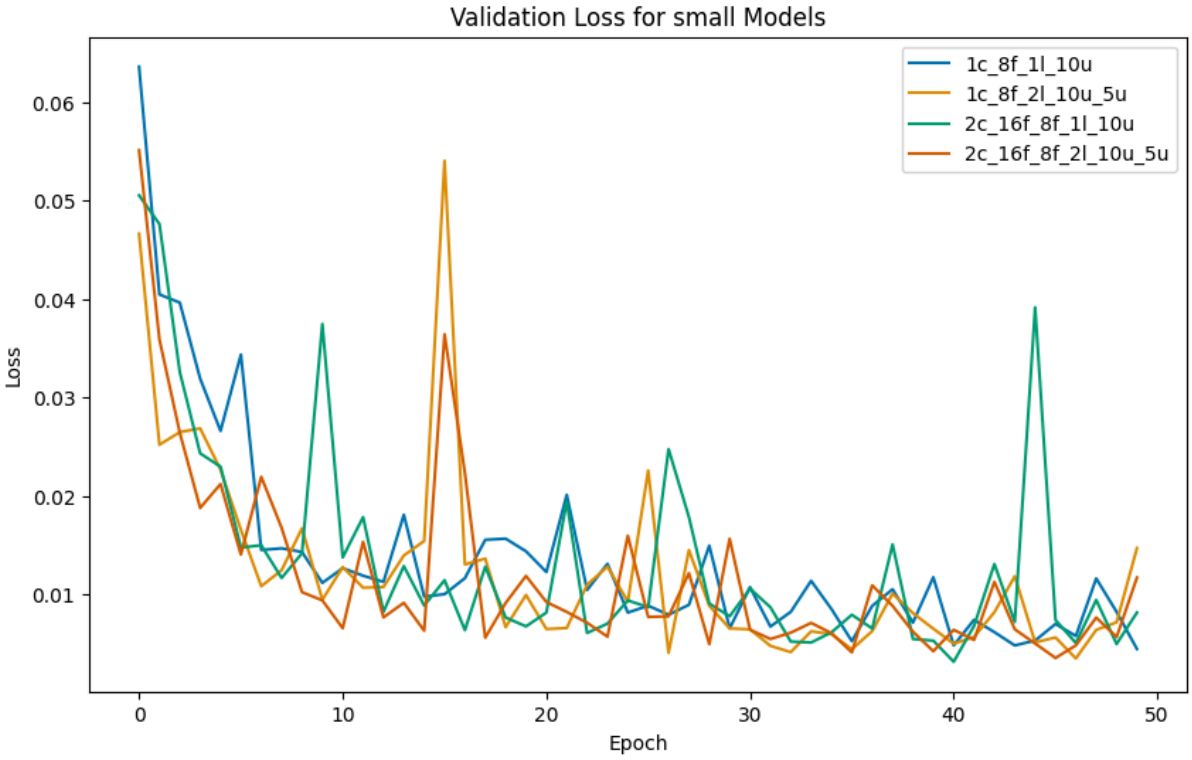


Abbildung 13: Validierungsverlust (val loss) der kleinen Modelle

In Abbildung 13 lässt sich erkennen, dass die Validierungsverluste zwar in den ersten Epochen stark sinken, jedoch weniger gleichmäßig als die Trainingsverluste. Alle Modelle haben bereits nach wenigen Epochen einen Validierungsverlust von unter 0,02 erreicht.

Bemerkenswert ist, dass sowohl das Modell `1c_8f_2l_10u_5u` als auch das Modell `2c_16f_8f_2l_10u_5u` bei Epoche 10 einen ausgesprochen niedrigen Validierungsverlust von 0,009 erreicht haben. Genau diese beiden Modelle zeigen jedoch in Epoche 15, wo

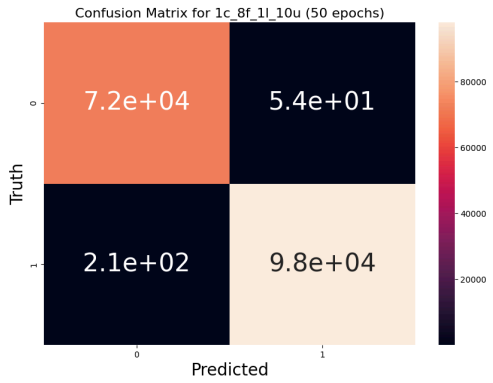
die Trainingsverluste bereits unter 0,01 sind, einen plötzlichen Validierungsverlust von je etwas über 0,03 und 0,05. Dies deutet auf ein mögliches Overfitting hin, was bedeutet, dass die Modelle möglicherweise die Daten „auswendig“ gelernt haben, ohne adäquat zu generalisieren.

Danach ist deren Verlauf wieder ähnlich der anderen Modelle und im Durchschnitt ähnlich zum Trainingsverlauf, bleibt jedoch stark fluktuierend. Das Modell `2c_16f_8f_1l_10u` zeigt außerdem bei Epoche 27 und Epoche 45 jeweils einen markanten Anstieg des Validierungsverlustes auf fast 0,02 bzw. 0,04, was ebenfalls auf potenzielles Overfitting in diesen Epochen hindeutet.

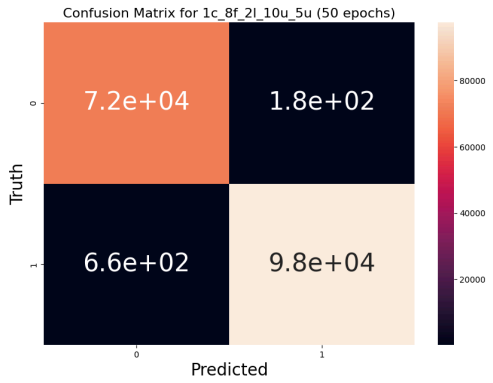
Bemerkenswerterweise weist in Epoche 50 das kleinte Modell, `1c_8f_1l_10u`, den niedrigsten Validierungsverlust von 0,0044 auf. Ein Vergleich ist in Tabelle 2 dargestellt.

Modell	**Parameteranzahl**	**Validierungsverlust in Epoche 50**
1c_8f_1l_10u	21651	0,0044
1c_8f_2l_10u_5u	21966	0,0147
2c_16f_8f_1l_10u	22891	0,0081
2c_16f_8f_2l_10u_5u	23206	0,0117

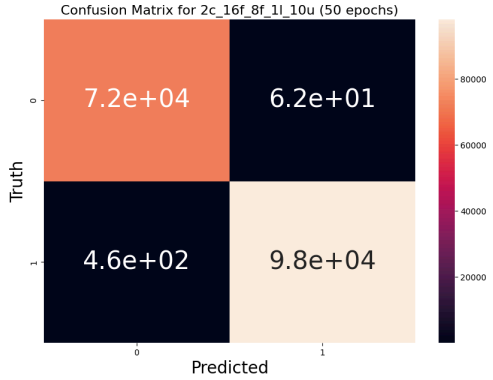
Tabelle 2: Validierungsverlust der kleinen Modelle in Epoche 50



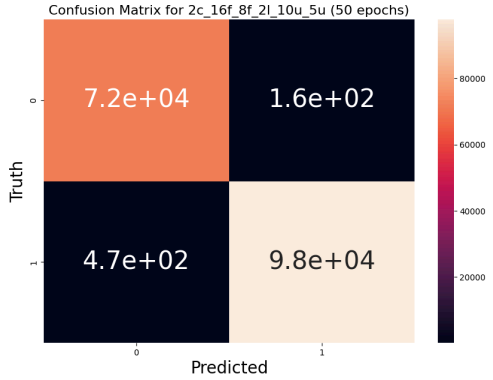
(a) Je eine Convolutional- und LSTM-Schicht



(b) Eine Convolutional- und zwei LSTM-Schichten



(c) Zwei Convolutional- und eine LSTM-Schicht



(d) Zwei Convolutional- und zwei LSTM-Schichten

Abbildung 14: Konfusionsmatrizen der kleinen Modelle nach Training über 50 Epochen.

In Abbildung 14 sind die Konfusionsmatrizen der kleinen Modelle nach 50 Epochen Training im direkten Vergleich zueinander dargestellt. Links sind jeweils die Konfusionsmatrizen der Modelle mit einer LSTM-Schicht, rechts die der Modelle mit zwei LSTM-Schichten. Die obere Zeile zeigt die Konfusionsmatrizen der Modelle mit einer Convolutional-Schicht, die untere Zeile die der Modelle mit zwei Convolutional-Schichten.

Besonders auffällig sind die Modelle mit nur einer LSTM-Schicht, die jeweils weniger als 100 False Positives (also Sturz-Einordnungen obwohl eine Sequenz nicht zu einem Sturz gehört) aufweisen. Das kleinste der Modelle, `1c_8f_1l_10u` (Abbildung 14a), hat die wenigsten False Positives mit nur 54 und ebenfalls die wenigsten False Negatives (also Sequenzen, die Teil eines Sturzes sind, die vom Modell jedoch nicht erkannt wurden) mit etwa 210. Im Gegensatz dazu zeigt das Modell `1c_8f_2l_10u_5u` (Abbildung 14b), welches eine zusätzliche LSTM-Schicht hat, die meisten False Negatives mit 660. Die Modelle mit zwei Convolution-Schichten, `2c_16f_8f_1l_10u` (Abbildung 14c) und `2c_16f_8f_2l_10u_5u` (Abbildung 14d), liegen dazwischen, mit etwa 460 bzw. 470 False Negatives.

Nach 50 Epochen Training liegen sowohl Präzision (Precision) als auch Sensitivität (Recall) bei allen kleinen Modellen über 99%. Bemerkenswert ist, dass das kleinste Modell,

`1c_8f_1l_10u`, mit 99,89% die höchste Sensitivität hat. Die Übersicht dazu ist in Tabelle 3 dargestellt.

Modell	Parameteranzahl	Sensitivität in %
1c_8f_1l_10u	21651	99,89
1c_8f_2l_10u_5u	21966	99,32
2c_16f_8f_1l_10u	22891	99,52
2c_16f_8f_2l_10u_5u	23206	99,52

Tabelle 3: Sensitivitäten der kleinen Modelle nach 50 Epochen

4.2 Leistung der großen Modelle

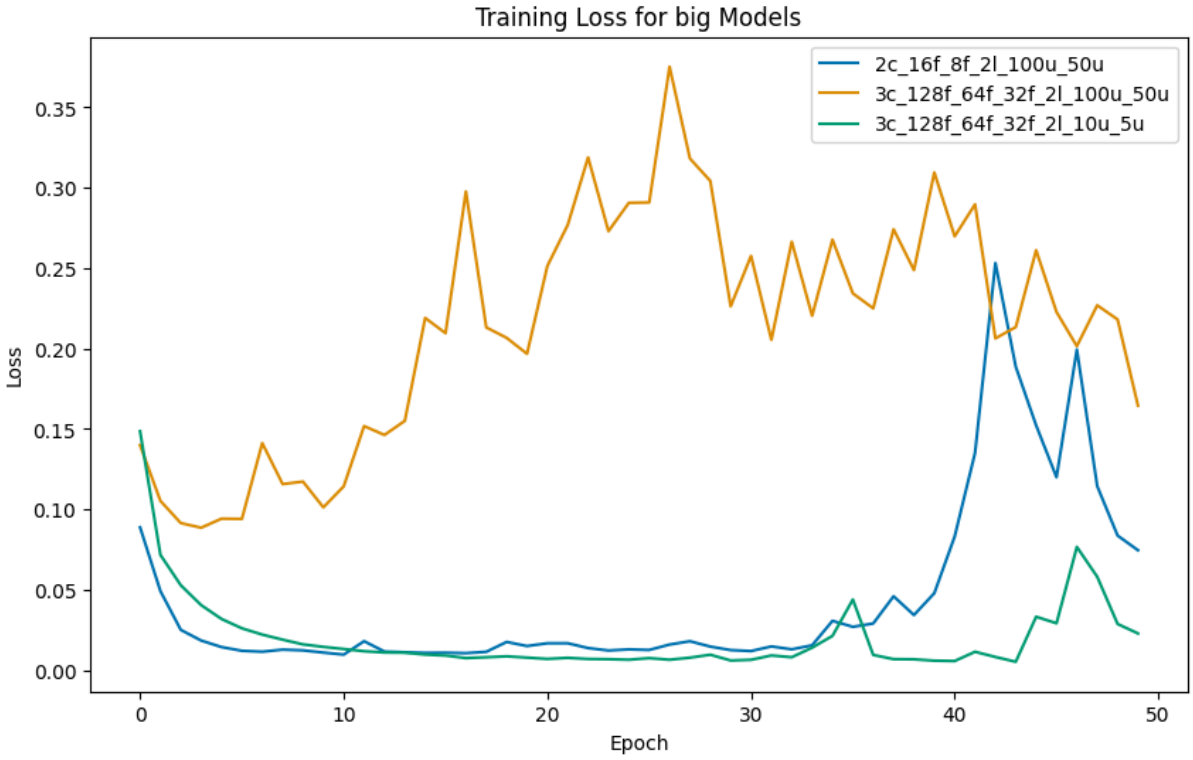


Abbildung 15: Trainingsverlust (loss) der großen Modelle.

Abbildung 15 zeigt den Verlauf des Trainingsverlustes der drei großen Modelle im Vergleich. Darin ist zu sehen, dass die beiden Modelle `2c_16f_8f_2l_100u_50u` (283 171 Parameter, zwei Convolution-Schichten und zwei etwas größere LSTM-Schichten) und `3c_128f_64f_32f_2l_10u_5u` (178 782 Parameter, drei Convolution-Schichten und twei etwas kleinere LSTM-Schichten) anfangs einen zügigen Abfall des Trainingsverlustes aufweisen. Das kleinste der großen Modelle, `3c_128f_64f_32f_2l_10u_5u`, erreicht bereits nach 7 Epochen einen Trainingsverlust von unter 0,02, während das mittlere Modell, `2c_16f_8f_2l_100u_50u`, diesen Wert bereits nach 3 Epochen unterschreitet.

In den mittleren Epochen zeigen beide Modelle eine gewisse Instabilität. Das mittlere Modell hat in Epoche 42 einen Anstieg des Trainingsverlustes auf 0,25, der sich bis Epoche 50 wieder auf 0,07 reduziert. Ähnlich weist das kleinste Modell in Epoche 35 einen leichten

Anstieg auf 0,04 sowie in Epoche 46 auf 0,08 auf, bevor der Trainingsverlust zum Abschluss auf 0,01 sinkt und damit den niedrigsten Wert unter den großen Modellen erreicht.

Das größte Modell, `3c_128f_64f_32f_2l_100u_50u` (1 008 987 Parameter, drei Convolution-Schichten und zwei größere LSTM-Schichten), zeigt zwar zu Beginn ebenfalls eine Reduktion des Trainingsverlustes und erreicht in Epoche 3 einen Wert von knapp unter 0,1, jedoch steigt der Verlust danach deutlich an. Das Modell erreicht sein Maximum in Epoche 26 mit 0,38, bevor es bis Epoche 50 wieder auf 0,16 abfällt. Trotz dieses Rückgangs erreicht der Trainingsverlust des größten Modells zu keiner Zeit die anderen Modelle, das Modell lernt also schlechter als die beiden kleineren Modelle.

Insgesamt zeigt sich, dass das kleinste Modell, `3c_128f_64f_32f_2l_10u_5u`, durchgängig den niedrigsten Trainingsverlust aufweist, während das größte Modell, `3c_128f_64f_32f_2l_100u_50u`, den höchsten Verlust verzeichnet. Ein Vergleich der Trainingsverluste in Epoche 50 ist in Tabelle 4 dargestellt.

Modell	**Parameteranzahl**	**Trainingsverlust in Epoche 50**
3c_128f_64f_32f_2l_10u_5u	178782	0,01
2c_16f_8f_2l_100u_50u	283171	0,07
3c_128f_64f_32f_2l_100u_50u	1008987	0,16

Tabelle 4: Trainingsverlust der großen Modelle in Epoche 50.

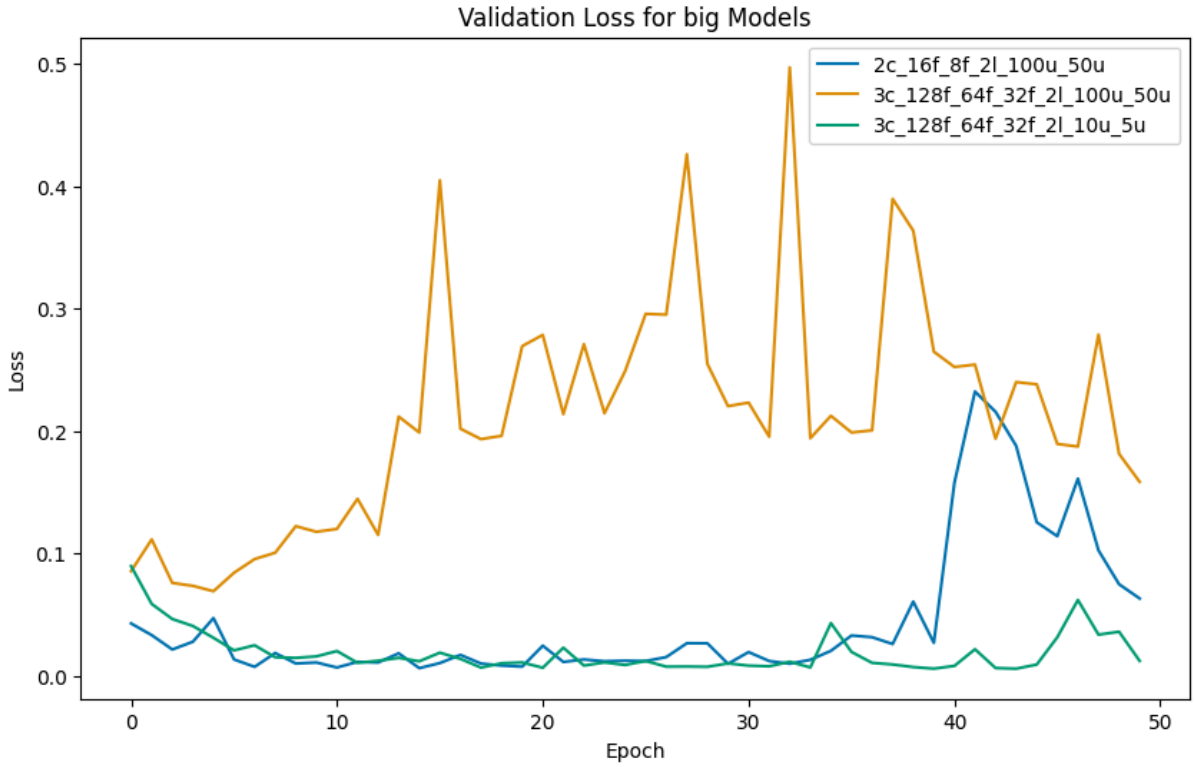


Abbildung 16: Validierungsverlust (val loss) der großen Modelle.

In Abbildung 16 ist der Verlauf des Validierungsverlustes der großen Modelle dargestellt, der sich ähnlich zum Trainingsverlust verhält. Das kleinste dieser Modelle,

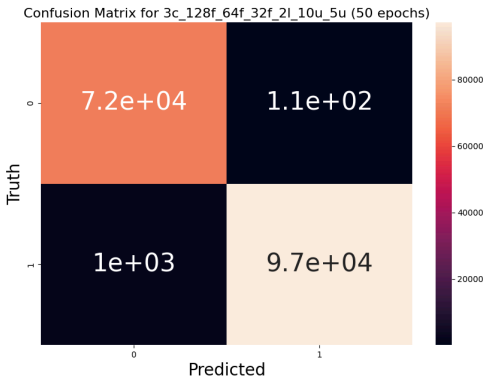
`3c_128f_64f_32f_2l_10u_5u`, erreicht bereits in Epoche 7 einen Validierungsverlust von unter 0,02. Das mittlere Modell, `2c_16f_8f_2l_100u_50u`, erreicht diesen Wert bereits in Epoche 5 und unterschreitet in Epoche 6 sogar die 0,01. Das kleinste Modell erreicht die 0,01 erst in Epoche 17. Das größte Modell, `3c_128f_64f_32f_2l_100u_50u`, erreicht diese Werte überhaupt nicht. Es erreicht nach wenigen Epochen seinen minimalen Validierungsverlust von 0,1 und steigt dann unregelmäßig immer weiter an, es scheint also überhaupt nicht oder etwas falsches zu lernen. Es erreicht ein erstes Maximum von 0,4 in Epoche 15.

In den mittleren Epochen verhalten sich die beiden kleineren Modelle ähnlich, wobei das kleinste Modell in Epoche 34 einen vorübergehenden Anstieg auf 0,04 aufweist, ähnlich wie sein Trainingsverlust eine Epoche vorher. Zu diesem Zeitpunkt fängt auch der Validierungsverlust des mittleren Modells an zu steigen, welches das Maximum von 0,23 jedoch erst in Epoche 41 erreicht, also wieder eine Epoche vor dem Maximum des Trainingsverlustes dieses Modells. Das große Modell bleibt weiterhin unregelmäßig und erreicht in Epoche 35 sein Maximum von 0,6.

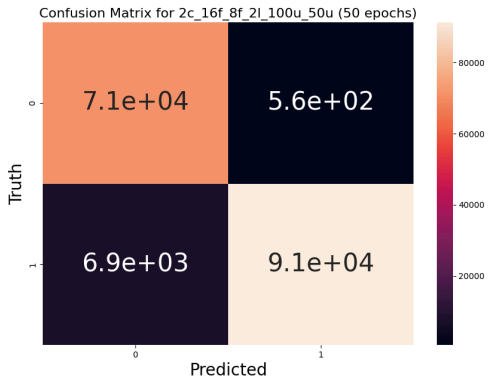
Insgesamt zeigt der Verlauf des Validierungsverlustes bei den großen Modellen eine ähnliche Dynamik wie der Trainingsverlust, wobei das kleinste Modell `3c_128f_64f_32f_2l_10u_5u` konstant die besten Ergebnisse liefert, während das größte Modell `3c_128f_64f_32f_2l_100u_50u` durchgängig schlechter abschneidet.

Modell	**Parameteranzahl**	**Validierungsverlust in Epoche 50**
3c_128f_64f_32f_2l_10u_5u	178 782	0,01239
2c_16f_8f_2l_100u_50u	283 171	0,06338
3c_128f_64f_32f_2l_100u_50u	1 008 987	0,15861

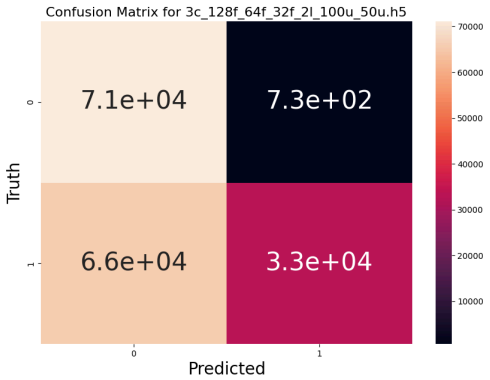
Tabelle 5: Validierungsverlust der großen Modelle bei 50 Epochen



(a) 3 Convolutional- und 2 LSTM-Schichten



(b) Je 2 Convolutional- und LSTM-Schichten



(c) 3 Convolutional- und 2 sehr große LSTM-Schichten

Abbildung 17: Konfusionsmatrizen der großen Modelle nach Training über 50 Epochen

In Abbildung 17 sind die Konfusionsmatrizen der großen Modelle im direkten Vergleich dargestellt. Sie zeigen, wie gut die Modelle nach 50 Epochen zwischen den beiden Klassen, Sturz (1) und nicht-Sturz (0) unterscheiden können.

Das Modell `3c_128f_64f_32f_2l_10u_5u` (Abbildung 17a) erreicht eine hohe Klassifikationsgenauigkeit für beide Klassen. Es klassifiziert die negative Klasse (0, nicht-Sturz) mit 72.000 korrekten Vorhersagen (True Negatives) fast fehlerfrei und hat nur 110 Fehlklassifikationen (False Positives). Für die positive Klasse (1, Sturz) werden 97.000 Sequenzen korrekt erkannt (True Positives), während 1.000 Sequenzen fälschlicherweise negativ klassifiziert werden (False Negatives).

Das Modell `2c_16f_8f_2l_100u_50u` (Abbildung 17b) zeigt ebenfalls eine solide, jedoch schwächere Performance als das kleinere Modell. Es klassifiziert die negativen Sequenzen mit 71.000 korrekten Vorhersagen, macht aber mit 560 Fehlklassifikationen etwas mehr Fehler. Bei der positiven Klasse werden 91.000 Sequenzen richtig erkannt, jedoch ist die Anzahl der False Negatives mit 6.900 deutlich höher als beim kleineren Modell. Die Ergebnisse deuten darauf hin, dass dieses Modell etwas mehr Schwierigkeiten hat, Stürze korrekt zu klassifizieren.

Das größte Modell, `3c_128f_64f_32f_2l_100u_50u` (Abbildung 17c), schneidet im Vergleich zu den anderen beiden Modellen signifikant schlechter ab, wie durch die Verlustgraphen zu erwarten. Es klassifiziert die negative Klasse zwar ähnlich gut, mit 71.000 korrekten Vorhersagen, aber die Anzahl der False Positives steigt auf 730. Besonders problematisch ist die Klassifikation der positiven Klasse, bei der nur 33.000 Sequenzen korrekt als Sturz erkannt werden, während 66.000 Sequenzen fälschlicherweise als negativ klassifiziert werden. Dieses Modell hat somit große Schwierigkeiten, Stürze korrekt als solche zu erkennen und zeigt die schlechteste Klassifikationsleistung unter den großen Modellen.

In der Übersicht in Tabelle 6 sind die Sensitivitäten der großen Modelle nach 50 Epochen im direkten Vergleich dargestellt. Auch hier zeigt sich, wie die Konfusionsmatrizen bereits andeuteten, dass das kleinste Modell `3c_128f_64f_32f_2l_10u_5u` die beste Sensitivität aufweist, während das größte Modell `3c_128f_64f_32f_2l_100u_50u` die schlechteste Sensitivität hat.

Modell	**Parameteranzahl**	**Sensitivität in %**
3c_128f_64f_32f_2l_10u_5u	178782	98,95
2c_16f_8f_2l_100u_50u	283171	92,96
3c_128f_64f_32f_2l_100u_50u	1008987	85,86

Tabelle 6: Sensitivität der großen Modelle nach 50 Epochen

4.3 Vergleich unterschiedlich großer Modelle

Nachdem oben bereits die Verläufe der Trainings- und Validierungsverluste der kleinen und großen Modelle einzeln betrachtet wurden, soll nun ein Gesamtbild der Unterschiede zwischen kleinen und großen Modellen gezeichnet werden. Dies bietet die Möglichkeit, den Effekt der Modellgröße auf die Generalisierungsfähigkeit und Klassifikationsgenauigkeit in der Sturzerkennung zu bewerten. Dies zeigt, ob sich der Mehraufwand für größere Modelle lohnt und inwiefern diese besser geeignet sind, um Stürze zu erkennen.

4.3.1 Trainings- und Validierungsverlust

Kleine Modelle | Große Modelle

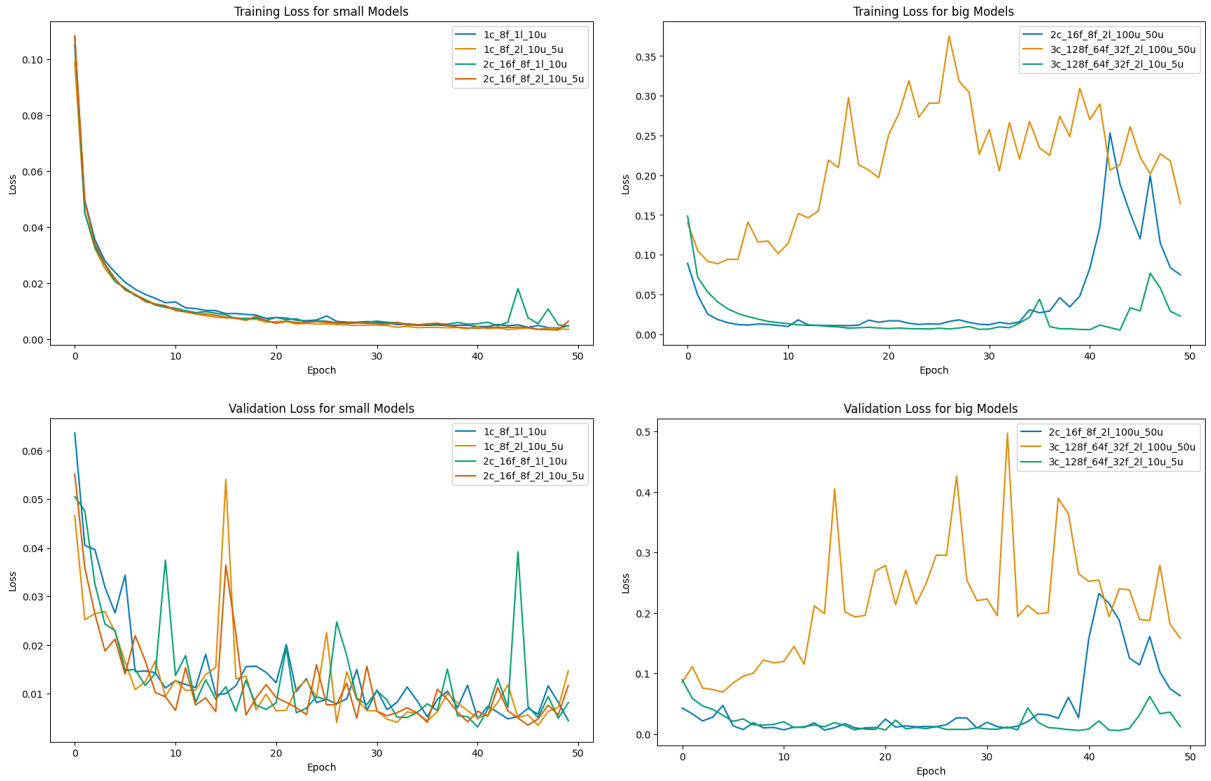


Abbildung 18: Gegenüberstellung der Verluste. Links die kleinen Modelle, rechts die großen Modelle. Oben die Trainingsverluste, unten die Validierungsverluste.

Abbildung 18 stellt die Trainings- und Validierungsverluste der kleinen (links) und großen (rechts) Modelle im direkten Vergleich gegenüber. Die kleinen Modelle erreichen insgesamt niedrigere Verluste als die großen Modelle. Die Trainingsverluste der kleinen Modelle sinken schneller und stabilisieren sich auf niedrigeren Werten als die der großen Modelle. Die Validierungsverluste der kleinen Modelle weisen hingegen deutliche Schwankungen auf, die von einem kontinuierlichen Verlauf abweichen, wie er bei idealisierter Stabilität zu erwarten wäre. Diese Abweichungen könnten auf Überanpassung an die Trainingsdaten oder eine suboptimale Lernrate hindeuten. Die großen Modelle weisen hingegen erst in späteren Epochen signifikante Abweichungen von einem glatten Verlauf auf. Das größte Modell `3c_128f_64f_32f_2l_100u_50u` zeigt dabei mit Abstand die schlechteste Performance, sowohl im Training als auch in der Validierung, was auf mehrere Probleme hindeuten kann. Dieses Modell ist auch im Vergleich zu den anderen großen Modellen um den Faktor 5-10 größer, so wie die anderen großen Modelle um den Faktor 5-10 größer sind als die kleinen Modelle. Da es im Vergleich sowohl in Größe als auch in Leistung deutlich von den anderen Modellen abweicht, lässt sich dieses Modell fast als eigene Kategorie einordnen.

Tabelle 7 zeigt die Verluste der Modelle nach 50 Epochen Training im direkten Vergleich. Die Fehlerraten der kleinen Modelle liegen bei rund 1%, während die Fehlerraten bei den großen Modellen mit der Größe des Modells auf jeweils 2,3%, 7,5% und 16,5% ansteigen.

Modell	Parameteranzahl	Trainingsfehler %	Validierungsfehler %
1c_8f_1l_10u	21651	0,478	0,442
1c_8f_2l_10u_5u	21966	0,347	1,468
2c_16f_8f_1l_10u	22891	0,464	0,813
2c_16f_8f_2l_10u_5u	23206	0,638	1,168
3c_128f_64f_32f_2l_10u_5u	178782	2,280	1,239
2c_16f_8f_2l_100u_50u	283171	7,456	6,338
3c_128f_64f_32f_2l_100u_50u	1008987	16,452	15,861

Tabelle 7: Trainings- und Validierungsfehlerraten der Modelle nach 50 Epochen Training. Die Trennlinie visualisiert die Unterscheidung zwischen kleinen und großen Modellen.

4.3.2 Sensitivität

Die Sensitivität der Modelle zeigt ebenfalls deutliche Unterschiede zwischen den kleinen und großen Modellen. Während die kleineren Modelle durchweg eine Sensitivität von über 99% erreichen, sinkt dieser Wert bei den großen Modellen, insbesondere bei dem größten Modell, auf unter 86%. Tabelle 6 zeigt die Sensitivitäten der großen Modelle nach 50 Epochen im direkten Vergleich.

Modell	Parameteranzahl	Sensitivität in %
1c_8f_1l_10u	21651	99,90
1c_8f_2l_10u_5u	21966	99,32
2c_16f_8f_1l_10u	22891	99,53
2c_16f_8f_2l_10u_5u	23206	99,52
3c_128f_64f_32f_2l_10u_5u	178782	98,95
2c_16f_8f_2l_100u_50u	283171	92,96
3c_128f_64f_32f_2l_100u_50u	1008987	85,86

Tabelle 8: Sensitivität der Modelle nach 50 Epochen

4.3.3 Ressourcen und Geschwindigkeit

Hier werden die Modelle anhand ihrer Inferenzzeiten und ihres Speicherbedarfs verglichen. Die Inferenzzeiten wurden über 1000 Inferenzen gemittelt, um eine aussagekräftige Vergleichsbasis zu schaffen. Der Speicherbedarf wurde als RAM-Arbeitsspeicher in Megabyte gemessen. Eine direkte Gegenüberstellung der Modelle ist in Tabelle 9 dargestellt.

Modell	Parameteranzahl	Speicher	Inferenzzeit
1c_8f_1l_10u	21651	0,08MB	51ms
1c_8f_2l_10u_5u	21966	0,08MB	51ms
2c_16f_8f_1l_10u	22891	0,09MB	51ms
2c_16f_8f_2l_10u_5u	23206	0,09MB	53ms
3c_128f_64f_32f_2l_10u_5u	178782	0,68MB	45ms
2c_16f_8f_2l_100u_50u	283171	1,08MB	52ms
3c_128f_64f_32f_2l_100u_50u	1008987	3,85MB	44ms

Tabelle 9: Inferenzzeiten und Speicherbedarf der Modelle

4.3.3.1 Speicherbedarf

Die kleinen Modelle sind durchweg ressourcenschonend und benötigen alle unter 0,1 MB RAM Arbeitsspeicher. Im Vergleich steigt der Speicherbedarf bei den großen Modellen erheblich an, mit einem Spitzenwert von 3,85 MB beim größten Modell, `3c_128f_64f_32f_2l_100u_50u`. Diese Zunahme reflektiert die größere Anzahl an Parametern und die gestiegene Modellkomplexität.

4.3.3.2 Inferenzzeiten

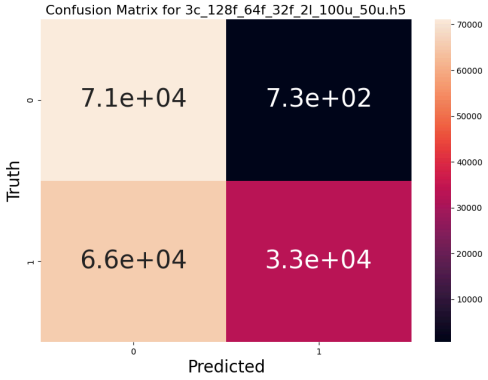
Die durchschnittlichen Inferenzzeiten der Modelle (gemittelt über 1000 Inferenzen) liegen in einem engen Bereich zwischen 44 und 53 ms. Überraschenderweise erzielen die größeren Modelle hier leicht kürzere Inferenzzeiten. Besonders das größte Modell, `3c_128f_64f_32f_2l_100u_50u`, benötigt nur 44 ms pro Inferenz, was von allen getesteten Modellen am schnellsten ist. Möglicherweise sind moderne Grafikkarten dazu in der Lage, so effizient zu parallelisieren, dass die größeren Modelle trotz ihrer Komplexität schneller Inferenzen durchführen können. Eine andere Erklärung könnte sein, dass das genutzte Framework abgesehen von der eigentlichen Inferenz am Modell einen gewissen Overhead hat und die eigentliche Inferent zeitlich keinen signifikanten Unterschied macht.

4.3.3.3 Gesamtbewertung

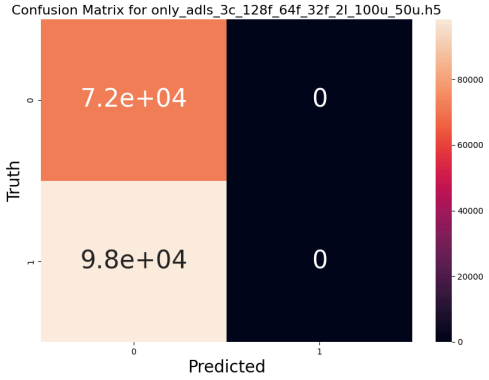
Obwohl die großen Modelle mehr Speicher verbrauchen, zeigen sie Überraschenderweise eine leicht verbesserte Effizienz bei der Inferenzzeit. Das kleine Modell `2c_16f_8f_2l_10u_5u` fällt mit der höchsten Inferenzzeit von 53 ms auf, benötigt jedoch erheblich weniger Speicher als die großen Modelle. Dies deutet darauf hin, dass die größere Modellkomplexität bei den großen Modellen die Rechenzeit pro Inferenz nicht zwingend negativ beeinflusst und diese teilweise sogar effizienter sein können.

Insgesamt brauchen alle Modelle im Verhältnis zu heutigen Rechenkapazitäten wenig Ressourcen und sind in der Lage, in Echtzeit Inferenzen durchzuführen.

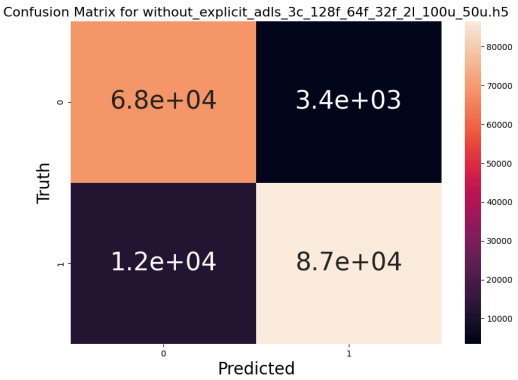
4.4 Vergleich der Modelle mit unterschiedlichen Trainingsdaten



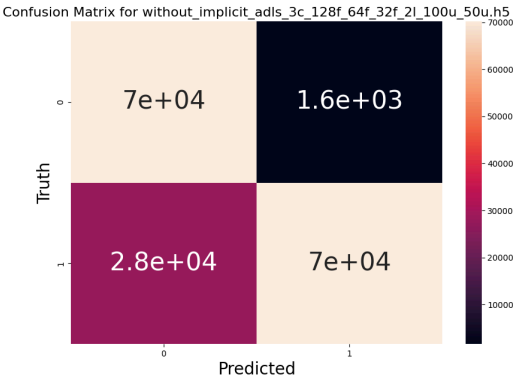
(a) Modell mit allen Trainingsdaten



(b) Modell, das nur ADLs verwendet



(c) Modell ohne explizite ADLs



(d) Modell ohne implizite ADLs

Abbildung 19: Vergleich der Konfusionsmatrizen für Modelle, die mit verschiedenen Trainingsdatenkombinationen trainiert wurden.

In Abbildung 19 sind die Konfusionsmatrizen der Modelle dargestellt, die mit verschiedenen Trainingsdatenkombinationen trainiert wurden.

Ein besonders auffälliges Ergebnis zeigt sich beim Modell, das nur ADLs als Trainingsdaten verwendet, dessen Evaluationsergebnis in Abbildung 19b dagestellt ist. Dieses Modell hat konsequent alle Eingabesequenzen als Nicht-Sturz klassifiziert, was eine Sensitivität von 0% bedeutet. Da keinerlei Stürze erkannt wurden, ist dieses Modell in realen Anwendungen für die Sturzerkennung nicht geeignet.

Die anderen Modelle zeigen eine differenzierte Leistung. Beim Modell mit allen Daten, dargestellt in Abbildung 19a, das Sturzsequenzen und sowohl ADL-Sequenzen aus expliziten ADL-Videos als auch implizite ADL-Sequenzen aus den Sturzvideos umfasst, wurden 71.000 Nicht-Sturz-Sequenzen korrekt erkannt (True Negatives), wobei 730 ADL-Sequenzen fälschlicherweise als Sturz klassifiziert wurden (False Positives). Im Vergleich dazu hat das Modell ohne implizite ADLs, dargestellt in Abbildung 19d, ebenfalls eine starke Erkennung von Nicht-Sturz-Sequenzen gezeigt, mit 70.000 True Negatives. Allerdings sind hier mit 1.600 False Positives etwas mehr ADLs fälschlicherweise als Sturz erkannt worden. Das Modell ohne explizite ADLs in Abbildung 19c, das nur Nicht-Sturz-Sequenzen aus Sturz-

videos als Trainingsdaten für ADLs verwendet, schneidet bei den Nicht-Sturz-Sequenzen nochmal etwas schlechter ab: Es erkennt lediglich 68.000 ADLs korrekt, während 3.400 ADLs als Stürze klassifiziert wurden.

Bei der Erkennung von Sturzsequenzen zeigt sich ebenfalls ein differenziertes Bild. Das Modell mit allen Trainingsdaten klassifiziert lediglich 33.000 Sturzsequenzen korrekt, während es 66.000 Sturzereignisse nicht erkennt. Im Gegensatz dazu zeigt das Modell ohne implizite ADLs eine deutlich bessere Erkennungsrate mit 70.000 korrekt identifizierten Stürzen und 28.000 nicht erkannten Sturzereignissen. Das Modell ohne explizite ADLs, das neben den Sturzdaten ausschließlich auf implizite ADLs aus Sturzvideos trainiert wurde, übertrifft beide anderen Modelle hinsichtlich der Sensitivität: Es identifiziert 87.000 Sturzsequenzen korrekt und verfehlt dabei nur 12.000. Diese Ergebnisse verdeutlichen, dass das Modell ohne explizite ADLs bei der Erkennung von Stürzen besonders leistungsfähig ist, jedoch auf Kosten einer erhöhten Rate an False Positives.

Zusammenfassend zeigt sich, dass das Modell ohne explizite Berücksichtigung von ADLs die höchste Sensitivität erreicht und somit die meisten Stürze korrekt erkennt. Allerdings geht diese Leistung mit einer erhöhten Rate an False Positives einher, wodurch ADLs häufiger fälschlicherweise als Stürze klassifiziert werden. Im Gegensatz dazu bietet das Modell, das alle verfügbaren Daten einbezieht, eine ausgewogenere Balance: Es erzielt weniger False Positives, jedoch auch eine geringere Anzahl an True Positives im Vergleich zum Modell ohne explizite ADLs.

Das Modell ohne implizite ADLs positioniert sich leistungsmäßig zwischen diesen beiden Ansätzen. Es erreicht mehr True Positives als das Modell, das alle Daten verwendet, jedoch weniger als das Modell ohne explizite ADLs. Gleichzeitig reduziert es die Anzahl der False Positives im Vergleich zum Modell ohne explizite ADLs, allerdings nicht so stark wie das Modell mit allen Daten.

In Tabelle 10 sind die Sensitivitäts- und Spezifitätswerte der verschiedenen Modellvarianten nach 50 Epochen Training zusammengefasst.

Variante	**Sensitivität**	**Spezifität**
Ohne explizite ADLs	87,88	90,96
Alle Daten	85,86	91,38
Ohne implizite ADLs	71,43	82,55
Nur ADLs	0,0	42,35

Tabelle 10: Sensitivität der Varianten nach 50 Epochen

Das Modell ohne explizite ADLs weist mit 87,88% die höchste Sensitivität auf, bei einer Spezifität von 90,96%. Es erkennt also von allen Varianten die meisten Stürze korrekt und klassifiziert gleichzeitig über 90% der ADLs korrekt als solche.

Das Modell, das alle Trainingsdaten verwendet, zeigt eine leicht geringere Sensitivität von 85,86%, jedoch eine etwas höhere Spezifität von 91,38%. Dies bedeutet, dass es zwar nicht ganz so viele Stürze wie das vorherige Modell erkennt, dafür jedoch etwas besser darin ist, ADLs korrekt als solche zu klassifizieren.

Das Modell ohne implizite ADLs verzeichnet mit 71,43% eine deutlich geringere Sensitivität, was darauf hinweist, dass es eine größere Anzahl von Stürzen nicht erkennt. Auch die Spezifität ist mit 82,55% deutlich niedriger, was bedeutet, dass dieses Modell auch weniger zuverlässig ist, wenn es darum geht, ADLs korrekt zu klassifizieren.

Das Modell, das nur ADLs verwendet, zeigt erwartungsgemäß die schlechtesten Ergebnisse: Eine Sensitivität von 0,0% bestätigt, dass es nicht in der Lage ist, Stürze als solche zu erkennen. Die Spezifität von 42,35% deutet darauf hin, dass es auch Schwierigkeiten hat, ADLs korrekt zu identifizieren. Dies zeigt, dass dieses Modell nahezu alle Ereignisse als ADLs interpretiert und somit völlig ungeeignet für die Sturzerkennung ist.

Das Modell, das nur ADLs verwendet, zeigt erwartungsgemäß die schlechtesten Ergebnisse: Die Sensitivität von 0,0% bestätigt eindeutig, dass es überhaupt nicht in der Lage ist, Stürze korrekt zu erkennen, da es jede Sequenz als ADL klassifiziert. Die Spezifität von 42,35% ergibt sich ausschließlich daraus, dass das Modell alle ADLs korrekt als solche erkennt, jedoch keinerlei Differenzierung zwischen ADLs und Stürzen vornimmt. Da die Spezifität nur von der Anzahl der korrekt erkannten ADLs abhängt, führt der Anteil der ADLs in den Testdaten zu einem scheinbar besseren Wert, der in diesem Fall lediglich ausdrückt, dass 42,35% der zur Evaluierung verwendeten Sequenzen ADLs sind.

4.5 Interpretation der Ergebnisse

In dieser Arbeit spielen Sensitivität (die Fähigkeit, Stürze korrekt zu erkennen) und Spezifität (die Fähigkeit, alltägliche Aktivitäten nicht fälschlicherweise als Stürze zu klassifizieren) eine zentrale Rolle. Ein zuverlässiges Sturzerkennungssystem sollte eine hohe Sensitivität aufweisen, um Stürze so schnell wie möglich zu erkennen und gleichzeitig eine hohe Spezifität, um Fehlalarme zu minimieren. Das richtige Gleichgewicht zwischen diesen Metriken ist entscheidend für die praktische Einsatzfähigkeit des Systems.

4.5.1 Kleine Modelle

Die Ergebnisse zeigen, dass kleinere Modelle eine hohe Sensitivität erreichen, was bedeutet, dass sie Stürze zuverlässig erkennen können. Dies ist eine positive Beobachtung, da die Zielsetzung dieser Arbeit darin besteht, Stürze möglichst frühzeitig zu identifizieren, um eine schnelle Benachrichtigung zu ermöglichen. Kleinere Modelle scheinen in der Lage zu sein, diese Anforderung zu erfüllen, und bieten zudem den Vorteil eines geringeren Rechenaufwands, was besonders bei der Echtzeitverarbeitung von Videodaten aus mehreren Kameras nützlich ist.

Ein mögliches Risiko bei kleineren Modellen besteht jedoch im Overfitting, insbesondere wenn die Lernrate für alle Modelle gleich gewählt wurde. Möglicherweise könnte eine hohe Lernrate für kleinere Modelle zu einer schnellen Anpassung an die Trainingsdaten führen, wodurch diese Modelle zwar im Training hohe Sensitivität erreichen, aber auf Testdaten schlechter generalisieren. In dieser Arbeit wurden jedoch keine eindeutigen Tests auf Overfitting durchgeführt, sodass weitere Analysen notwendig wären, um dies zu bestätigen.

Der geringe Ressourcenbedarf kleinerer Modelle eröffnet das Potenzial, sie auf ressourcenschwacher Hardware in häuslichen Umgebungen einzusetzen. Dies entspricht dem Ziel der Arbeit, ein System zu entwickeln, das auch in Echtzeitszenarien effektiv funktioniert und dabei wenig Rechenleistung benötigt.

4.5.2 Große Modelle

Größere Modelle zeigen eine etwas schlechtere Sensitivität, was darauf hindeutet, dass sie mehr Trainingsepochen oder eine höhere Anfangslernrate benötigen könnten, um ihr volles Potenzial zu entfalten. Ihre Stärke liegt jedoch in der höheren Spezifität, was bedeutet, dass sie seltener ADLs fälschlicherweise als Stürze klassifizieren. Dies macht sie zu einer möglichen Lösung, wenn es darum geht, Fehlalarme zu minimieren.

Die Ergebnisse legen nahe, dass größere Modelle möglicherweise besser darin sind, auf mehr Trainingsdaten zu generalisieren, was sie in Szenarien nützlich macht, in denen viele verschiedene Aktivitäten erfasst werden müssen. Trotz ihrer höheren Anzahl an Parametern zeigten die größeren Modelle keine signifikant längeren Inferenzzeiten, was auf die Effizienz moderner Hardware hindeutet. Es scheint, dass Grafikkarten in der Lage sind, die Berechnungen parallel zu verarbeiten, was die Inferenzzeiten niedrig hält. Alternativ könnte es sein, dass der Overhead im verwendeten Framework so hoch ist, dass die eigentliche Inferenzzeit nur einen kleinen Teil der gesamten Verarbeitungszeit ausmacht.

4.5.3 Unterschiedliche Trainingsdaten

Ein besonders interessantes Ergebnis zeigt die Leistung des Modells, das ohne explizite ADLs trainiert wurde. Dieses Modell erzielte die höchste Sensitivität und wies somit eine starke Erkennungsleistung auf. Die Spezifität war jedoch etwas geringer als die des Modells, das mit allen Daten (einschließlich expliziter ADLs) trainiert wurde. Dies verdeutlicht, dass bereits wenige ADLs in den Trainingsdaten ausreichen können, um eine zuverlässige Sturzerkennung zu ermöglichen, ohne dass viele ADLs fälschlicherweise als Stürze erkannt werden. Dies deutet darauf hin, dass die Trainingsdaten effizienter gestaltet werden könnten, indem der Anteil an ADLs reduziert wird, was sowohl die Trainingszeit verkürzen als auch den Datenschutz verbessern könnte.

Das Modell, das mit allen verfügbaren Daten trainiert wurde, erreichte hingegen eine etwas höhere Spezifität. Dies legt nahe, dass das Einbeziehen einer größeren Anzahl von ADLs in die Trainingsdaten dazu beitragen kann, die Fehlalarme weiter zu reduzieren.

Zwar scheint es nicht zwingend erforderlich zu sein, eine große Menge expliziter ADLs einzubeziehen, jedoch bringt dies keine signifikanten Nachteile mit sich und kann in bestimmten Szenarien sogar vorteilhaft sein.

4.5.4 Bedeutung für das Gesamtsystem

Während die Modelle eine gute Grundlage für die Sturzerkennung bieten, könnte das Gesamtsystem durch Heuristiken die noch vorhandenen Schwächen der Modelle ausgleichen. Ein potenzieller Ansatz wäre, Stürze nur dann zu melden, wenn mehrere aufeinanderfolgende Frames (oder eine Mindestanzahl von Frames innerhalb einer Sekunde, z.B. 66%) als Sturz klassifiziert werden. Diese Heuristik könnte die Wahrscheinlichkeit von False Positives verringern, ohne die Sensitivität des Systems signifikant zu beeinträchtigen. So würde die Zuverlässigkeit des Systems gesteigert, was für den praktischen Einsatz besonders wichtig ist. Solange in einem Bild keine Person zu sehen ist, werden außerdem keine Daten an das Modell gesendet, was die Rechenleistung weiter reduziert.

4.5.5 Limitationen

Trotz der vielversprechenden Ergebnisse zeigen sich einige Limitationen, die für eine vollständige Bewertung eines Gesamtsystems berücksichtigt werden müssen. Eine der zentralen Herausforderungen betrifft das Risiko des Overfittings, insbesondere bei den kleineren Modellen. Da für alle Modelle eine gleiche Lernrate verwendet wurde, besteht die Möglichkeit, dass kleinere Modelle zu schnell auf die Trainingsdaten angepasst werden. Dies könnte zu einer schlechteren Generalisierung auf neue, unbekannte Daten führen. Overfitting wurde in dieser Arbeit jedoch nicht explizit getestet, sodass dies in zukünftigen Studien genauer untersucht werden sollte.

Ein weiteres potenzielles Limit liegt in den begrenzten Datensätzen, die für das Training und die Validierung der Modelle verwendet wurden. Die derzeitigen Ergebnisse basieren auf einem spezifischen Satz von Trainingsdaten, der möglicherweise nicht alle Variationen und Szenarien der realen Welt abdeckt. Insbesondere könnte die Generalisierungsfähigkeit der Modelle durch höhere oder niedrigere Kamerawinkel oder andere Arten von Kameras beeinträchtigt werden. Um sicherzustellen, dass die Modelle robust genug sind, müssen sie in Zukunft auf größeren und vielfältigeren Datensätzen getestet werden.

Obwohl im Verlauf der Arbeit Heuristiken als mögliche Lösung zur Reduzierung von False Positives genannt wurden, sind diese in der aktuellen Arbeit nicht explizit implementiert oder getestet worden. Heuristiken, wie beispielsweise die Anforderung, dass mehrere aufeinanderfolgende Frames als Sturz klassifiziert werden müssen, könnten in der praktischen Implementierung jedoch eine entscheidende Rolle spielen, um die Zuverlässigkeit des Gesamtsystems zu verbessern. Es bleibt offen, wie effektiv diese Methoden in Kombination mit den entwickelten Modellen arbeiten würden.

Zusätzlich gibt es ethische und datenschutztechnische Überlegungen, die bei der Entwicklung eines Sturzerkennungssystems berücksichtigt werden müssen. Insbesondere die

Frage, wie viele und welche Arten von ADLs in den Trainingsdaten verwendet werden sollten, um einerseits den Datenschutz zu wahren und andererseits eine hohe Spezifität zu gewährleisten, bleibt eine wichtige Frage. Ein Kompromiss zwischen einem möglichst geringen Datensammeln und der Optimierung der Modellerkennung könnte erforderlich sein.

4.5.6 Schlussbetrachtung

Obwohl die kleinen Modelle eine gute Sensitivität aufweisen und in der Lage sind, Stürze in Echtzeit zu erkennen, könnte sich bei zukünftigen Tests zeigen, dass größere Modelle mit besserer Spezifität eine wichtigere Rolle spielen, falls sich in realen Anwendungen trotz Heuristiken Fehlalarme als problematisch erweisen.

Die Effizienz der Modelle bietet gute Ansätze, um das System weiter zu optimieren, auch im Hinblick auf die Ressourcenschonung und die Einsetzbarkeit auf Standardhardware in privaten Haushalten, ohne größere Rechenkapazität der Cloud in Anspruch zu nehmen.

Künftige Arbeiten könnten sich auch darauf konzentrieren, Overfitting und variable Lernraten genauer zu untersuchen. Die Integration von Heuristiken und die Berücksichtigung ethischer und datenschutzrechtlicher Aspekte sind ebenfalls wichtige Schritte, um ein Sturzerkennungssystem zu entwickeln, das nicht nur effektiv, sondern auch verantwortungsbewusst und benutzerfreundlich ist.

5 Fazit

Diese Arbeit zeigt, dass ein effizientes und zuverlässiges Sturzerkennungssystem auf der Basis von Skelettdaten möglich ist, die aus RGB-Kamerabildern extrahiert wurden. Die durchgeführten Experimente belegen, dass sowohl kleine als auch große Modelle in der Lage sind, eine hohe Sensitivität zu erreichen und Stürze zuverlässig zu identifizieren. Während die kleineren Modelle mit einer Sensitivität von über 99% überzeugen und damit eine Echtzeiterkennung in ressourcenschwachen Umgebungen ermöglichen, bieten größere Modelle eine höhere Spezifität, jedoch keine signifikant verbesserte Sensitivität. Die Möglichkeit, kleinere Modelle lokal auszuführen, macht den Einsatz in privater Umgebung besonders attraktiv, da sie aufgrund ihrer geringen Rechenlast eine lokale Verarbeitung erlauben und somit den Schutz der Privatsphäre erhöhen, indem eine Cloudanbindung vermieden wird.

Diese Ergebnisse stellen eine wertvolle Grundlage für zukünftige Arbeiten dar. Insbesondere könnte die Generalisierungsfähigkeit der Modelle durch größere und vielfältigere Datensätze sowie durch eine gezielte Datenaugmentierung weiter verbessert werden. Eine interessante Methode wäre die synthetische Generierung von Sturzdaten, um das Modell robuster gegen seltene und ungewöhnliche Sturzmechanismen zu machen. Auch eine Untersuchung variabler Lernraten könnte zukünftig getestet werden, um die Trainingsprozesse optimal an die jeweilige Modellarchitektur anzupassen. Ergänzend

wäre die Erweiterung eines Modells durch zusätzliche Datenquellen, wie Tiefen- oder LiDAR-Kameras, ein vielversprechender Ansatz, um die räumliche Präzision, Effizienz und Zuverlässigkeit des Gesamtsystems weiter zu erhöhen.

Da die Verarbeitung der Sequenzen derzeit ausschließlich zweidimensional erfolgt, stellt eine Datenaugmentierung durch Spiegelung der Sturzdaten eine Möglichkeit dar, die Anzahl der Sturzdatensequenzen zu verdoppeln und somit das Training zu stärken. Diese Methode wurde in der vorliegenden Arbeit nicht umgesetzt, könnte jedoch einen sinnvollen nächsten Schritt zur weiteren Optimierung der Modelle darstellen.

Abschließend lässt sich festhalten, dass ein verlässliches Sturzerkennungssystem, basierend auf Pose Estimation und Deep Learning, nicht nur die Sicherheit und das Wohlbefinden älterer Menschen in ihrer häuslichen Umgebung signifikant erhöhen kann, sondern auch eine wichtige Entlastung für das Pflegepersonal darstellen kann. Damit trägt die hier entwickelte Lösung potenziell dazu bei, sowohl das Sicherheitsgefühl als auch die Unabhängigkeit der Anwender zu stärken und könnte so einen wesentlichen Beitrag zur Bewältigung der Herausforderungen im Pflegebereich leisten.

Bibliographie

[1] Kripesh Adhikari, Hamid Bouchachia, und Hammadi Nait-Charif. 2017. Activity recognition for indoor fall detection using convolutional neural network. In *2017 Fifteenth IAPR International Conference on Machine Vision Applications (MVA)*, 2017. 81–84. https://doi.org/10.23919/MVA.2017.7986795

[2] AMD. AMD Radeon 780M GPU. Abgerufen von https://www.amd.com/en/support/downloads/drivers.html/processors/ryzen-pro/ryzen-pro-7000-series/amd-ryzen-7-pro-7840u.html#product-specs-2

[3] Apple Inc. Apple Watch. Abgerufen 19. Juni 2024 von https://www.apple.com/watch/

[4] E. Auvinet, C. Rougier, J. Meunier, A. St-Arnaud, und J. Rousseau. 2010. *Multiple cameras fall dataset*. Abgerufen 19. Juni 2024 von http://www.iro.umontreal.ca/~labimage/Dataset/

[5] Pedram Azad, Tamim Asfour, und Rüdiger Dillmann. 2009. Stereo-Based vs. Monocular 6-DoF Pose Estimation Using Point Features: A Quantitative Comparison. In *Autonome Mobile Systeme 2009*, 2009. Springer Berlin Heidelberg, Berlin, Heidelberg, 41–48.

[6] Greet Baldewijns, Glen Debard, Gert Mertes, Bart Vanrumste, und Tom Croonenborghs. 2016. Bridging the gap between real-life data and simulated data by providing a highly realistic fall dataset for evaluating camera-based fall detection algorithms. *Healthcare Technology Letters* 3, 1 (2016), 6–11. https://doi.org/https://doi.org/10.1049/htl.2015.0047

[7] Z. Cao, G. Hidalgo Martinez, T. Simon, S. Wei, und Y. A. Sheikh. 2019. OpenPose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2019).

[8] Zhe Cao, Tomas Simon, Shih-En Wei, und Yaser Sheikh. 2017. Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields.

[9] Glen Debard, Peter Karsmakers, Mieke Deschodt, Ellen Vlaeyen, Eddy Dejaeger, Koen Milisen, Toon Goedemé, Bart Vanrumste, und Tinne Tuytelaars. 2012. Camera-Based Fall Detection on Real World Data. In *Outdoor and Large-Scale Real-World Scene Analysis*, 2012. Springer Berlin Heidelberg, Berlin, Heidelberg, 356–375. https://doi.org/10.1007/978-3-642-34091-8_16

[10] Hao-Shu Fang, Jiefeng Li, Hongyang Tang, Chao Xu, Haoyi Zhu, Yuliang Xiu, Yong-Lu Li, und Cewu Lu. 2022. AlphaPose: Whole-Body Regional Multi-Person Pose Estimation and Tracking in Real-Time. Abgerufen von https://arxiv.org/abs/2211.03375

[11] Jane Fleming und Carol Brayne. 2008. Inability to get up after falling, subsequent time on floor, and summoning help: prospective cohort study in people over 90. *BMJ* 337, (2008). https://doi.org/10.1136/bmj.a2227

[12] Google. Google Pixel Watch 2. Abgerufen 19. Juni 2024 von https://store.google.com/product/pixel_watch_2

[13] Google. 2023. MediaPipe Solutions: Vision - Pose Landmarker. Abgerufen 28. Mai 2024 von https://ai.google.dev/edge/mediapipe/solutions/vision/pose_landmarker

[14] R. Jan Gurley, Nancy Lum, Merle Sande, Bernard Lo, und Mitchell H. Katz. 1996. Persons Found in Their Homes Helpless or Dead. *New England Journal of Medicine* 334, 26 (1996), 1710–1716. https://doi.org/10.1056/NEJM199606273342606

[15] Rabia Hasib, Kaleem Nawaz Khan, Miao Yu, und Muhammad Salman Khan. 2021. Vision-based Human Posture Classification and Fall Detection using Convolutional Neural Network. In *2021 International Conference on Artificial Intelligence (ICAI)*, 2021. 74–79. https://doi.org/10.1109/ICAI52203.2021.9445263

[16] Sepp Hochreiter und Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation* 9, 8 (1997), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

[17] Huawei Technologies Co., Ltd. HUAWEI WATCH GT 3 - HUAWEI Global. Abgerufen 19. Juni 2024 von https://consumer.huawei.com/en/wearables/watch-gt3

[18] Susanna Kochskämper. 2018. Eine Entwicklung der Pflegefallzahlen in den Bundesländern - eine Simulation bis 2035. 10, 2018. Abgerufen 10. Oktober 2023 von https://www.iwkoeln.de/fileadmin/user_upload/Studien/Report/PDF/2018/IW-Report_33_2018_Pflegefallzahlen.pdf

[19] Deutsches Rotes Kreuz. Hausnotruf. Abgerufen 24. April 2024 von https://www.drk.de/hilfe-in-deutschland/senioren/altersgerechtes-wohnen/hausnotruf/

[20] Songsheng Li. 2023. Fall Detection With Wrist-Worn Watch by Observations in Statistics of Acceleration. *IEEE Access* 11, (2023), 19567–19578. https://doi.org/10.1109/ACCESS.2023.3249191

[21] Camillo Lugaresi, Jiuqiang Tang, Hadon Nash, Chris McClanahan, Esha Uboweja, Michael Hays, Fan Zhang, Chuo-Ling Chang, Ming Guang Yong, Juhyun Lee, und others. 2019. MediaPipe: A Framework for Building Perception Pipelines. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2019. 0. Abgerufen 24. April 2024 von https://developers.google.com/mediapipe

[22] Peter Mayer und Paul Panek. 2011. Ein AAL-Ansatz zur Status- und Aktivitätsbeurteilung durch Domainexpertenwissen mit wenigen und nichtintrusiven Sensoren. In *Demographischer Wandel - Assistenzsysteme aus der Forschung in den Markt*, 2011. VDE Verlag GmbH, Berlin. Abgerufen 18.

Juni 2024 von https://www.academia.edu/22045404/An_AAL_Approach_to_Status_and_Activity_Assessment_by_Use_of_Domain_Expert_Knowledge_based_on_Sparse_Nonintrusive_Sensors

[23] Paul Panek, Hannes Dangl, Daniela Krainer, Stefan Leipold, Peter Mayer, Johannes Oberzaucher, Marjo Rauhala, Daniel Thaler, Christoph Wagner, Franz Werner, Katharina Werner, und Wolfgang Zagler. 2015. Integration komplementärer AAL Systeme zu einem modularen und flexiblen Produkt zur Sturzerkennung und Alltagsunterstützung. 2015.

[24] George Papandreou, Tyler Zhu, Nori Kanazawa, Alexander Toshev, Jonathan Tompson, Chris Bregler, und Kevin Murphy. 2017. Towards Accurate Multi-person Pose Estimation in the Wild.

[25] Samara Pires, Sonia Rodrigues, Lourd Bharathy Arokiadass, und Sejal Chopra. 2021. A Real-Time Position Monitoring System For Fall Detection and Analysis Using Human Pose Estimation. In *2021 4th Biennial International Conference on Nascent Technologies in Engineering (ICNTE)*, 2021. 1–7. https://doi.org/10.1109/ICNTE51185.2021.9487724

[26] Rainer Radtke. 2022. Bedarf an Pflegekräften in Deutschland bis 2035. Abgerufen 10. Oktober 2023 von https://de.statista.com/statistik/daten/studie/172651/umfrage/bedarf-an-pflegekraeften-2025/

[27] Samsung Electronics. Samsung Galaxy Watch. Abgerufen 19. Juni 2024 von https://www.samsung.com/galaxy-watch/

[28] Gerhard Schiemer. 2012. Sturzerkennungsmethoden im Vergleich. Universität Wien, Universitätsbibliothek, 1010 Wien, Universitätsring 1. https://doi.org/10.25365/thesis.21611

[29] Sebastian Schmidpeter. 2022. Sturzerkennung mittels Pose Estimation im Kontext von Smart Home Umgebungen. Abgerufen von https://users.informatik.haw-hamburg.de/~ubicomp/arbeiten/bachelor/schmidpeter.pdf

[30] Doran Holger Scholl, Dominik Sandro Schwer, Richard Hempel, und Anton Petrov. *Funktionsweise und Vergleich von Methoden zur Generierung von Punktwolken in einer Versandstation*. Furtwangen, Deutschland. Abgerufen 20. Oktober 2024 von https://opus.hs-furtwangen.de/frontdoor/deliver/index/docId/10470/file/Funktionsweise.pdf

[31] Sturzmelder. Sturzmelder, Falldetektor und Notrufknopf mit automatischem Notruf. Abgerufen 19. Juni 2024 von http://www.sturzmelder.de/

[32] TensorFlow. 2024. Time series forecasting. Abgerufen 30. Juni 2024 von https://www.tensorflow.org/tutorials/structured_data/time_series

[33] Somasundaram Vadivelu, Sudakshin Ganesan, O. V. Ramana Murthy, und Abhinav Dhall. 2017. Thermal Imaging Based Elderly Fall Detection. In *Computer Vision – ACCV 2016 Workshops*, 2017. Springer International Publishing, Cham, 541–553.

[34] Paul Voigt und Axel Von dem Bussche. 2017. The EU General Data Protection Regulation (GDPR). *A Practical Guide, 1st Ed., Cham: Springer International Publishing* (2017).

[35] Roberta Vrskova, Robert Hudec, Patrik Kamencay, und Peter Sykora. 2022. A New Approach for Abnormal Human Activities Recognition Based on ConvLSTM Architecture. *Sensors* 22, 8 (2022). https://doi.org/10.3390/s22082946

[36] Yupeng Wang, Shibing Zhu, und Changqing Li. 2019. Research on Multistep Time Series Prediction Based on LSTM. In *2019 3rd International Conference on Electronic Information Technology and Computer Engineering (EITCE)*, 2019. 1155–1159. https://doi.org/10.1109/EITCE47263.2019.9095044

[37] Ronald J. Williams und David Zipser. 1989. A learning algorithm for continually running fully recurrent neural networks. *Neural Comput.* 1, 2 (Juni 1989), 270–280. https://doi.org/10.1162/neco.1989.1.2.270

[38] World Health Organization. 2008. WHO global report on falls prevention in older age. Abgerufen 10. Oktober 2023 von https://www.who.int/publications-detail-redirect/9789241563536

[39] Bin Xiao, Haiping Wu, und Yichen Wei. 2018. Simple Baselines for Human Pose Estimation and Tracking. *arXiv e-prints* (April 2018), arXiv:1804.06208. https://doi.org/10.48550/arXiv.1804.06208

Erklärung zur selbstständigen Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original