Universität Bremen

Sturzerkennung in Infrarotvideos mit Hilfe von Rekurrenten Neuronalen Netzen

Bachelorarbeit am Cognitive Systems Lab
Prof. Dr.-Ing. Tanja Schultz
Fachbereich 3
Universität Bremen

von

Kilian Lüdemann

Betreuer:
Yale Hartmann, M. Sc.
Dipl.-Ing. Hui Liu, M. Sc.

Gutachter:
Prof. Dr.-Ing. Tanja Schultz
Dr.-Ing. Joachim Clemens

Tag der Anmeldung: 15. April 2021
Tag der Abgabe: 6. September 2021

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Bremen, den 6. September 2021

Zusammenfassung

Stürze stellen gerade für ältere Menschen ein großes Verletzungsrisiko dar. Oft schaffen es diese nur schwer, oder gar nicht wieder aufzustehen und Hilfe zu rufen. Aus diesem Grund möchte man eine automatische Erkennung von Stürzen haben, um Hilfskräfte zu alarmieren. Ich habe eine Sturzerkennung mit Hilfe von Rekurrenten Neuronalen Netzen in Kombination mit Trigger Word Recognition realisiert. Auf die verwendeten Daten wurde background subtraction angewandt und somit eine Personen Maske erzeugt. Umgesetzt wird die Lösung mit dem Machine-Learning-Framework Keras. Aufgrund meiner Herangehensweise fällt die Lösung des Problems in den Bereich der Bilderverarbeitung und des Machine-Learnings.

Die Forschungsfragen sind: "Lässt sich eine automatische Erkennung von Stürzen mit Hilfe von Rekurrenten Neuronalen Netzen in Tiefenkarten-Videos realisieren? Wenn ja, können alle Stürze erkannt werden?". Sie können mit ja, es können Stürze erkannt werden und nein, es wurden nicht alle Stürze erkannt, beantwortet werden. Auf ungesehenen Daten konnte ein Recall von 70% erreicht werden.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einleitung

Stürze stellen besonders für ältere Menschen ein großes Verletzungsrisiko dar. Laut dem Bundesministerium für Gesundheit können die Folgen eines Sturzes schwerwiegend sein [Bun20]. Beispielsweise können Knochenbrüche entstehen und im schlimmsten Fall zu einer Pflegebedürftigkeit oder längeren Klinikaufenthalten führen. Oft schaffen es ältere Menschen nur schwer, oder gar nicht wieder aufzustehen und Hilfe zu rufen. Aus diesem Grund möchte man eine automatische Erkennung von Stürzen haben, welche Hilfskräfte alarmiert.

Da Stürze aus einer Abfolge von Bewegungen einer Person entstehen, kann man diese mit Hilfe einer Kamera einfangen. Die daraus entstandenen Videos können weiter verarbeitet werden, um die Stürze anhand von ihren Merkmalen zu erkennen. Auf existierende Algorithmen zu dieser Problematik gehe ich im Kapitel 3 ein.
Diese bedienen sich meist traditionellen Herangehensweisen. Ich hingegen möchte versuchen eine Sturzerkennung mit Hilfe von Rekurrenten Neuronalen Netzen zu realisieren. Als Datengrundlage werden Tiefenkarten-Videos verwendet. Für die Umsetzung der Lösung verwende ich das Machine-Learning-Framework Keras.

Zuerst werde ich notwendige Grundlagen kurz erläutern. Dann auf bereits vorhandene Arbeit in diesem Bereich eingehen und darauf folgt eine Beschreibung der verwendeten Daten und erstellte Varianten dieser. Zu Letzt stelle ich meinen Algorithmus vor und diskutiere die Ergebnisse. Aufgrund meiner Herangehensweise fällt die Lösung des Problems in den Bereich der Bilderverarbeitung und des Machine-Learnings.

Die Forschungsfragen sind: Lässt sich eine automatische Erkennung von Stürzen mit Hilfe von Rekurrenten Neuronalen Netzen in Tiefenkarten-Videos realisieren? Wenn ja, können alle Stürze erkannt werden?

2. Grundlagen

In diesem Kapitel soll über die der Arbeit zu Grunde liegenden Techniken gesprochen werden. Als Erstes möchte ich darauf eingehen, was Tiefenkarten-Videos ausmacht, da sich diese von gewohnten RGB-Formaten unterscheiden. Danach gehe ich auf Neuronale Netze ein, welche eine fundamentale Methode darstellen. Diese grundlegende Erklärung wird durch die Klasse der Rekurrenten Neuronalen Netze erweitert. Dann spreche ich über Convolutional Neural Networks und Long Short-Term Memory (LSTM) Netze, welche im Prozess und der finalen Lösung eine entscheidende Rolle spielen. Zuletzt erkläre ich die Trigger Word Recognition (TWR), welche die Grundlage für den Lösungsansatz bildet.

2.1 Tiefenkarten-Videos

In dieser Arbeit werden Daten verwendet, welche Tiefeninformationen über die erfassten Videos aufweisen. Genauer wird hier der Abstand des Sichtfeldes zur Kamera im Pixelformat angegeben. Ein Bild mit Tiefeninformationen wird „Tiefenkarte“ (engl. Depth Map) genannt. Diese können mit Hilfe von mehreren Methoden erlangt werden. Aktuelle Beispiele dafür sind [RAR+11]:

- Stereo Kameras
- Time-of-Flight (TOF) Kameras
- Structured light

Da meine Daten mit Hilfe des Stereo Kamera Verfahrens erfasst wurden, möchte ich mich hier auf diese Variante beschränken. Für die Erfassung der Daten wurde die Kamera „The Intel® RealSense™ Depth Module D410“ verwendet [Cor19]. Dieses Modul verwendet zwei Kameras, welche im RGB-Format aufnehmen. Diese haben im Gegensatz zum Modul D420 beispielsweise keinen IR-Cut Filter. Das bedeutetet, dass das Infrarotspektrum des einfallenden Lichts mit erfasst wird. Viele Kameras haben standardmäßig einen solchen Filter eingebaut. Der Unterschied ist in Abbildung 2.1 zu sehen.

Abbildung 2.1: Unterschied zwischen der Verwendung eines IR-Cut Filters und der Bilderfassung ohne einen solchen. [Mar20]

Weiter wird ein Infrarot Projektor verwendet, welcher ein statisches Infrarot Muster auf das Sichtfeld projiziert und damit die Fähigkeit der Kamera verbessert, Tiefeninformationen zu errechnen. Besonders nützlich ist das bei Bereichen im Bild, welche eine geringe Textur aufweisen [Cor19].
Die Verwendung des Infrarotspektrums hat den Vorteil, dass die Kamera auch bei schlechten Lichtverhältnissen eine Tiefenkarte errechnen kann. Ein Vergleich zwischen guten und schlechten Lichtverhältnissen ist in Abbildung 2.2 zu sehen.

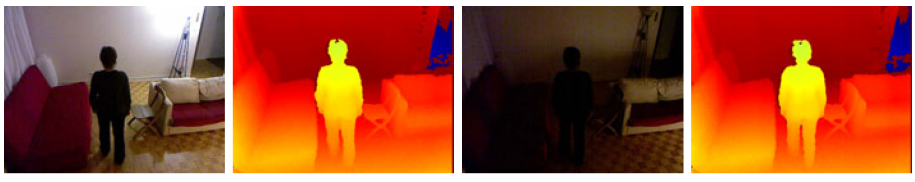

Abbildung 2.2: Unterschied zwischen einer Tiefenkarte, welche bei guten Lichtverhältnissen und einer welche bei schlechten Lichtverhältnissen erstellt wurde. [RAR+11]

Die Errechnung der Tiefenkarten findet mit Hilfe des erwähnten Stereoverfahrens statt. Dabei wird mit Hilfe der beiden Kameras eine Triangulation vorgenommen, um die Distanz zum Sichtfeld zu errechnen. Wie genau eine Tiefenkarte errechnet wird ist in Kapitel 4.3 - 4.9 des Datenblatts zur Produktfamilie der Kamera erläutert [Cor19].

2.2 Neuronale Netze

Traditionelle Machine Learning Ansätze sind bei komplexeren Daten mit vielen Features oft durch die Curse of dimensionality limitiert. Features sind Merkmale von Daten. So kann ein Apfel beispielsweise das Feature: Farbe - Rot haben. Die Curse of dimensionality beschreibt ein Problem, bei dem hochkomplexe Daten mit ihren vielen Features für Machine Learning Modelle zunehmend schwerer zu erlernen und zu unterscheiden sind. Zu diesen hochkomplexen Daten zählen durch ihre viele Pixel auch Bilder und Videos. Für die Lösung dieses Problems bieten sich Neuronale Netze an.

Die einfachste und älteste Form Neuronaler Netze ist das Perzeptron. Ein solches besteht aus Input-Variablen, ihren zugehörigen gewichteten Kanten, einem Bias, einer linearen Kombinationsfunktion und einer Activation Function. Eine Visualisierung des Perzeptrons ist in Abbildung 2.3 zu sehen.
Die lineare Kombinationsfunktion kann, wie unten gezeigt, beispielsweise eine Summe sein. Weiter wird die Activation Function auf den Output der vorherigen Funktion angewandt. Der Bias beeinflusst die Kombination der Input-Variablen und kann je nach Wert beispielsweise dazu führen, dass die Activation Function immer, oder gar nicht „feuert“. Wie in Abbildung 2.3 zu sehen, ist der Bias in diesem Beispiel 1.
Das Perzeptron summiert die Ergebnisse der Multiplikation der einzelnen Eingabewerte mit ihren zugehörigen Gewichten und dem Bias auf und gibt dieses Ergebnis in die Activation Function. In unserem Beispiel werden alle Ergebnisse ab Null und darunter zu Minus Eins und alle darüber liegenden Ergebnisse werden zu Eins. Gängige Activation Functions sind Sigmoid

$$sig(x) = \frac{1}{1 + e^{-x}} \tag{2.1}$$

tanh

$$tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.2}$$

und Rectified Linear Unit (ReLu)

$$relu(x) = max(0, x). \tag{2.3}$$

Die trainierbaren Parameter sind die Gewichte. Das Perzeptron kann so bereits mehrere Boolesche Operatoren wie UND, ODER und NICHT umsetzen.

In Neuronalen Netzen werden Perzeptron-ähnliche Units zu so genanneten Fully Connected-Layern kombiniert. Ein Layer ist ein gebräuchlicher Begriff für eine Schicht in einem Neuronalen Netz. Ein Neuronales Netz besteht aus einem Input-Layer, Hidden-Layern und einem Output-Layer. Für gewöhnlich ist der Input-Layer so groß, dass er für alle gewünschten Features jeweils eine Unit bereitstellt.
Hidden-Layer befinden sich zwischen dem Input- und Output-Layer des Neuronalen Netzes. Es gibt meist mehrere Hidden-Layer, welche oft dafür da sind den Input

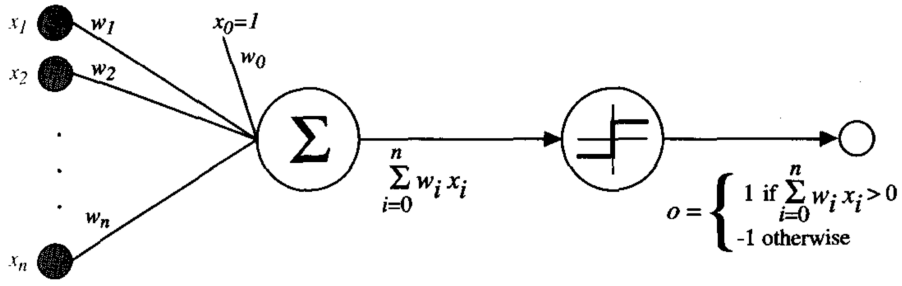

Abbildung 2.3: Das Perzeptron. Die Eingabevariablen sind mit x1 - xn bezeichnet. Die Gewichte der Kanten sind mit w1 - wn bezeichnet. Der Bias wird durch x0, sowie w0 dargestellt. Darauf folgt die Summenfunktion und dann die Activation Function. [Mit97]

in eine designierte Output-Form zu bringen. Die Anzahl der Hidden-Layer, sowie die Art der einzelnen zählen beispielsweise zu den Hyperparametern. Hyperparameter beeinflussen mit ihren Werten das Training des Neuronalen Netzes. Anders als trainierbare Parameter sind sie vom Entwickler bestimmt. Der Begriff Training beschreibt den Prozess in dem die Gewichte des Neuronalen Netzes iterativ an eine Mapping-Funktion von Input-Daten zu Output-Label angepasst werden. Ein Label beschreibt im Machine Learning die Zuordnung eines Datums zu einer Klasse. So kann Beispielsweise ein Apfel das Label Eins und eine Birne das Label Zwei haben. Der Output-Layer hat meist so viele Units, wie es Klassen im Klassifikator gibt. In anderen Architekturen wie zum Beispiel der Regression ist das anders. Auf diese werde ich allerdings nicht weiter eingehen, da sie uns hier nicht interessiert.

Um ein bestmögliches Fitting der Mapping-Funktion zu erreichen wird die Technik des Gradient Descent angewendet. Fitting wird beim Machine Learning als Begrifflichkeit dafür benutzt, wie gut ein Machine Learning Modell zwischen Klassen von Objekten unterscheiden kann. Dabei wird zwischen Under- und Overfitting unterschieden. Underfitting bedeutet, dass das Modell nicht gut zwischen den Klassen unterscheidet und Overfitting, dass das Modell so gut zwischen den gelernten Daten unterscheidet, dass es nicht mehr generalisieren kann. Overfitting wird oft auch als auswendig lernen bezeichnet Diese Unterscheidung wird mit Hilfe einer Hyperplane getroffen. Die Hyperplane beschreibt im Machine Learning eine n-dimensionale Grenze welche die durch ein Machine Learning Modell getrennten Klassen separiert.
Gradient Descent ist eine Optimierungsfunktion aus dem Machine Learning mit dessen Hilfe die Gewichte eines Machine Learning Modells so angepasst werden, dass die Loss-Funktion bestenfalls zu einem globalen Minimum findet. Während diese zu einem globalen Minimum konvergiert, wird versucht die Accuracy zu maximieren. Diese beschreibt den korrekten Anteil der Vorhersagen (engl. Predictions) des Netzes. Für die Umsetzung wird eine Learning Rate verwendet, welche die Schrittgröße bestimmt.
Als Loss bezeichnet man die Differenz zwischen disigniertem Output, also der per Klassenzuordnung erhaltenen Label und einer per Forwardpropagation der Trainingsdaten errechneten Vorhersage, mit Hilfe des zu trainierenden Machine Learning Modells. Die Kombination der einzelnen Losse der Trainingsdaten wird dann als Loss-Funktion bezeichnet. Die Forwardpropagation beschreibt einen Prozess beim Training eines Neuronalen Netzes, bei dem die Trainingsdaten vom Input-Layer,über

die Hidden-Layer zum Output-Layer verrechnet und anschließend mit dem designierten Label verglichen werden.
Mit Hilfe des Losses kann der Gradient Descent über die Backpropagation vorgenommen werden. Die Backpropagation beschreibt einen Prozess beim Training eines Neuronalen Netzes, bei dem die Gewichte der Kanten zwischen den Layern aufgrund ihres höheren oder niedrigeren Einflusses auf die Loss-Funktion angepasst werden.

Wie schon erwähnt, finden die oberen Schritte iterativ statt. Zum einen können mehrere Samples gleichzeitig am Gradient Descent teilnehmen, also Forward- und Backpropagation durchlaufen. Diese Parallelisierung führt dazu, dass die Rechenleistung des Computers besser ausgenutzt werden kann. Der Gradient Descent wird jedoch für jedes Sample einzeln vorgenommen. Eine gemeinsam durchlaufende Menge wird hier Batch genannt. Zum anderen kann das gesamte Trainings-Set mehrfach durchlaufen werden. Ein Durchlauf dieses Sets wird als Epoche bezeichnet. Nach dem Training wird mit Hilfe des Netzes und eines Test-Sets eine weitere Vorhersage errechnet. Dieses Set teilt keine Samples mit dem Trainings-Set und beinhaltet daher „ungesehene“ Daten. Diese Vorhersage wird mit einer Accuracy bewertet. Ähnlich des Test-Sets gibt es noch ein Validation-Set (auch Development-Set genannt), welches während dem Training, nach den Epochen eine dem Test-Set sehr ähnlichen Funktion dient. Mit Hilfe dieses Sets kann der Fortschritt des Trainings beobachtet werden. Eine detailliertere Erklärung ist hier zu finden [Mit97]. Eine Visualisierung eines Neuronalen Netzes ist in Abbildung 2.4 zu sehen.

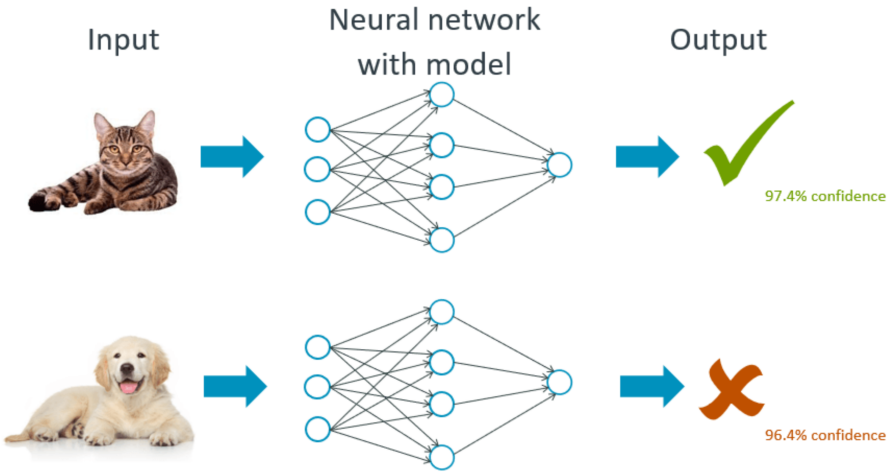

Abbildung 2.4: Ein Neuronales Netz für die Klassifikation von Katzenbildern. In der Mitte ist ein solches zu sehen, welches 3 Input Units (links), 4 Hitten Units in einem Hidden-Layer (mitte) und eine Output Unit (rechts) hat. Die Netzdarstellung ist für diese Aufgabe stark vereinfacht dargestellt. Die Abbildung zeigt, dass dieses Neuronale Netz zwischen Katzen- und Hundebildern unterscheiden kann [Öz20]

2.3 Rekurrente Neuronale Netze

Im Kapitel 2.2 wird erklärt, wie Neuronale Netze grundsätzlich funktionieren. Für die Anwendung auf temporale Daten wird allerdings ein Modell, bei dem ein vorheriger

Zeitschritt einen Einfluss auf den nächsten haben kann, benötigt. So kann ein temporaler Zusammenhang entstehen. Dafür werden Rekurrente (auch Rückgekoppelte) Neuronale Netze verwendet. Je nach Architektur des Modells wird der Output eines Zeitschritts dann beispielsweise weiter nach hinten zum Output des Netzes und an den nächsten Zeitschritt, oder nur an den nächsten Zeitschritt weitergeleitet. Eine Visualisierung dieser Art von Architektur ist in Abbildung 2.5 zu finden.
Rekurrente Neuronale Netze können einen temporalen Zusammenhang zwischen Zeitschritten erlernen. Jedoch gibt es Probleme, bei denen diese Zusammenhänge weit auseinander liegen. Denkt man beispielsweise an die Spracherkennung, so müssen manchmal Informationen, welche vor einigen Sätzen mitgeteilt wurden einen Einfluss auf einen späteren haben können. Rekurrente Neuronale Netze in ihrer ursprünglichen Form scheinen Probleme mit dem erlernen von weit auseinander liegenden temporalen Zusammenhängen zu haben. Begründungen dafür arbeitete der spätere Miterfinder der LSTM-Zellen Jürgen Hochreiter in seiner Diplomarbeit aus [Hoc91]. Die erwähnte Erweiterung der LSTM-Zellen kann temporale Zusammenhänge über weite zeitliche Entfernungen besser lernen und wird im nächsten Kapitel genauer erläutert. Weitere Informationen zu Rekurrenten Neuronalen Netzen sind hier zu finden [Ola15].

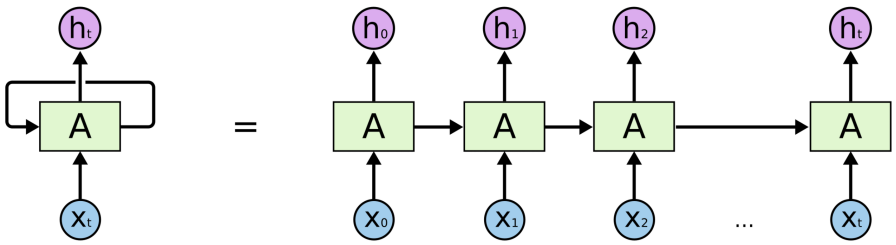

Abbildung 2.5: Ein Rekurrentes Neuronales Netz. Links zu sehen ist das Rekurrente Neuronale Netz, welches eine auf sich selbst zeigende Schleife hat. Die ausgerollte Version ist rechts zu sehen. Dabei ist A immer die gleiche Einheit, welche mit jedem Zeitschritt X0 - Xt anders interagiert und je nach dem einen anderen Output h0 - ht ausgibt. [Ola15]

2.4 Long Short-Term Memory Netze

Long Short-Term Memory-Netze sind eine State-of-the-Art Technik, welche eine Erweiterung zum Ansatz der Rekurrenten Neuronalen Netze darstellt. Wie schon erwähnt schaffen diese einen temporalen Zusammenhang zwischen selbst weit entfernten Zeitschritten. Erreicht wird dies durch die einzelnen LSTM-Zellen.
Eine Visualisierung ist in Abbildung 2.6 zu sehen.
In diesen LSTM-Zellen befindet sich der „cell state“. Dieser kann als ein Förderband zwischen den LSTM-Zellen bezeichnet werden, denn hier wird die Information der temporalen Zusammenhänge gespeichert und weiter geleitet. Über diesen cell state sind alle LSTM-Zellen verbunden. In Abbildung 2.7 ist dieser anschaulich dargestellt.
Des Weiteren befinden sich drei Gate-Units in jeder Zelle. Diese haben verschiedene Funktionen und beeinflussen den cell state. Es gibt wie in Abbildung 2.6 dargestellt das Forget-Gate, das Input-Gate und das Output-Gate.
Das Forget-Gate entscheidet mit Hilfe des Inputs des zugehörigen Zeitschritts und dem Ouput des letzten Zeitschritts, welche Informationen im cell state nicht länger

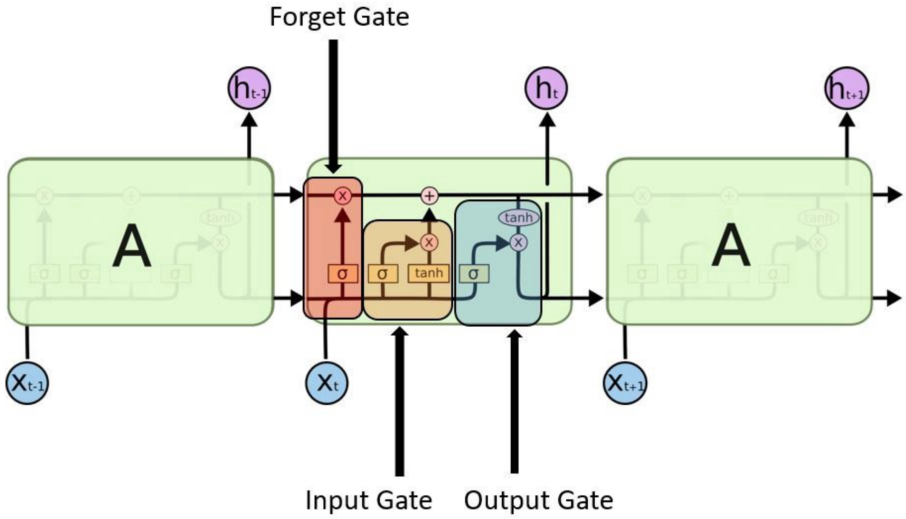

Abbildung 2.6: Zu sehen sind die Bestandteile einer LSTM-Zelle. A bezeichnet die gleiche Zelle, welche zu einem anderen mit Xt beschrifteten Zeitschritt zugeordnet wird. Die jeweiligen Outputs werden mit ht beschriftet. Die LSTM-Zelle besteht aus dem Forget Gate, dem Input Gate und dem Output Gate, sowie dem cell state, welcher in Abbildung 2.7 zu sehen ist. σ steht für die Sigmoid Activation Function (2.1), tanh steht für die tanh Activation Function (2.2) und x sowie + sind elementweise Multiplikation und Addition. Die Grafik mit eingezeichneten Gates stammt von [Öz20]. Das Original ist jedoch hier zu finden [Ola15].

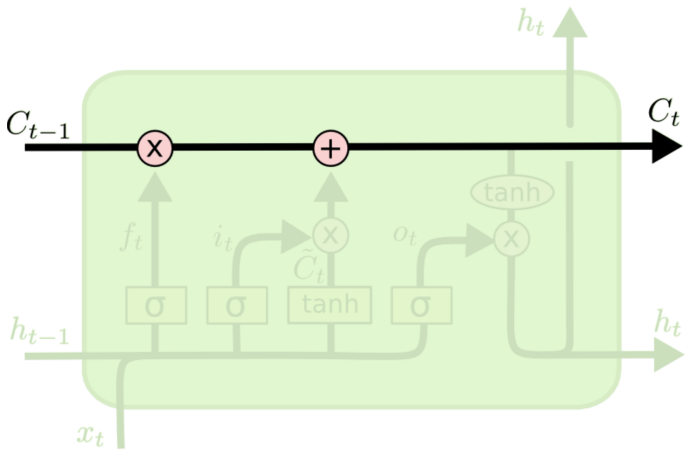

Abbildung 2.7: Der cell state einer Long Short-Term Memory-Zelle. Ct-1 markiert den Output des vorherigen Zeitschritts und Ct den Output des aktuellen Zeitschritts. X und + markieren die elementweise Multiplikation und Addition. [Ola15].

benötigt werden. Das Input-Gate fügt neue Informationen des zugehörigen Zeitschritts hinzu und das Output-Gate steuert, welche Informationen des cell states als Output der Zelle weitergereicht werden sollen. Für eine genauere Erläuterung der LSTM-Zellen empfehle ich das Original [HS97], oder diese etwas simplere Variante [Ola15]

2.5 Convolutional Neural Networks

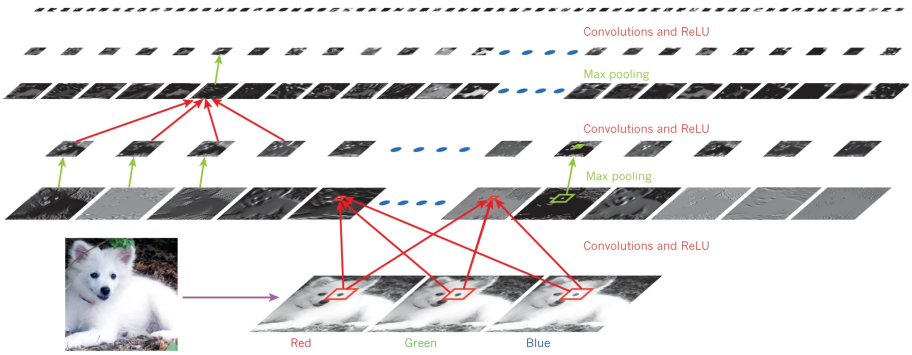

Abbildung 2.8: Ein Convolutional Neural Network. Zu sehen sind mehrere Layer eines solchen Netzes. Über den Input-Layer und die darauf folgenden Convolutions mit ReLu und den Max pooling Layern wird das Hundebild pixelweise kleiner, jedoch nehmen die Anzahl der Feature maps zu. Die roten Kasten mit Pfeilen zeigen welche Bereiche des Bildes mit Hilfe ihrer filter banks in den jeweiligen Feature maps resultieren. Grün zeigt die Anwendung von Max Pooling. [LBH15]

Convolutional Neural Networks sind eine nützliche Erweiterung Neuronaler Netze um hochkomplexe Daten wie Bilder einfacher zu verarbeiten. Bilder haben so viele Features wie sie Pixel haben, multipliziert mit der Anzahl ihrer Kanäle. Würde man alle diese mit Fully-connected-Layern erfassen wollen, wäre das sehr aufwändig. Convolutional Neural Networks vereinfachen das Problem erheblich.
Sie wenden so genannte „filter banks“ auf Ausschnitte des Bildes an. Filter banks haben einen Kernel fester Größe, welcher immer einen Ausschnitt des Bildes nimmt und diesen mit Hilfe eines Filters verrechnet. Das Ergebnis solcher filter bank-Anwendungen wird „feature map“ genannt. Alle Units in einer feature map teilen die selbe filter bank.
Filter können hier beispielsweise zum Erkennen von horizontalen oder vertikalen Kanten dienen. Die Anzahl der Kernel-Anwendungen, welche über den Stride (auch Verzug genannt) bestimmt werden, die Größe des Kernels und die Anzahl der Verwendeten Filter sorgen dann für eine Faltung der Informationen. Dies geschieht meist so, dass das Netz in den hinteren Layern mit schmaleren und längeren Volumen rechnet und somit die hohe Komplexität der Bilder stark herunter bricht. Das Ziel ist meist eine Form, die in einen Fully-conneted-Layer über gehen kann. Eine Visualisierung davon ist in Abbildung 2.8 zu finden.
Zwischen den Convolutional-Layern werden oft Pooling-Layer verwendet. Diese sorgen dafür, dass die Dimensionen weiter verringert werden. Da gibt es beispielsweise Max-Pooling und Average-Pooling. Wie der Name schon sagt, wird hier aus einem kleinen Ausschnitt des Outputs des vorherigen Layers nur der größte Wert, oder der Durchschnitt weiter getragen. Der Rest wird verworfen.
Ist für einen Layer keine Faltung erwünscht, kann Padding verwendet werden. Man unterscheidet zwischen Valid-Padding, Same-Padding und Full-Padding. Dabei werden die Ränder des zu faltenden Arrays mit Nullen aufgefüllt. Beim Valid-Padding wird kein Padding vorgenommen. Falls die Dimensionen des Filters den Rand überschreiten würden, fällt der nicht gefilterte Teil einfach weg.
Beim Same-Padding werden am Rand so viele Nullen hinzugefügt, dass nach der

Faltung die selben Dimensionen erhalten bleiben.
Full-Padding hingegen kann dafür sorgen, dass die Dimensionen nach der Faltung größer sind, als zuvor. Für eine genauere Erklärung empfehle ich [ON15] und [LBH15].

2.6 Trigger Word Recognition

Die Trigger Word Recognition ist eine Technik aus der Spracherkennung. Im dazugehörigen Paper wird erklärt, wie die Autoren das Wort „activate“ in Spektrogrammen von Audiosignalen erkennen [SDVS20].
Sie verwenden ein Rekurrentes Neuronales Netz mit Gated Recurrent Unit (GRU)-Layern. Das Netz kann ein Trigger Word erkennen, nachdem es gesagt wurde. Für die Umsetzung der Annotation wählten die Autoren einen Label-Vektor, welcher in einem Many-to-Many-Relationship zum Input steht. Das bedeutet hier, dass für jeweils 4 Input-Zeitschritte ein Eintrag im Label-Vektor vorhanden ist.
Da die Spektrogramme eine Länge von 5511 Zeitschritten haben, wird der Label-Vektor auf die Länge 1375 festgelegt. Initial sind alle Einträge auf Null gesetzt. Die Annotation des Trigger Words „activate“ wird jetzt so vorgenommen, dass 50 Einträge nachdem das Trigger Word gesprochen wurde den Wert Eins erhalten. Die Anzahl 50 dient hier als „landing pad“, um das Training zu vereinfachen, da eine einzige Eins deutlich schwieriger genau vorherzusagen ist.
Eine Visualisierung dieser Annotation ist in Abbildung 2.9 zu sehen. In den Trainingsdaten befinden sich weitere Worte und Nebengeräusche. Trotzdem kann eine Erkennung des Trigger Words erreicht werden.

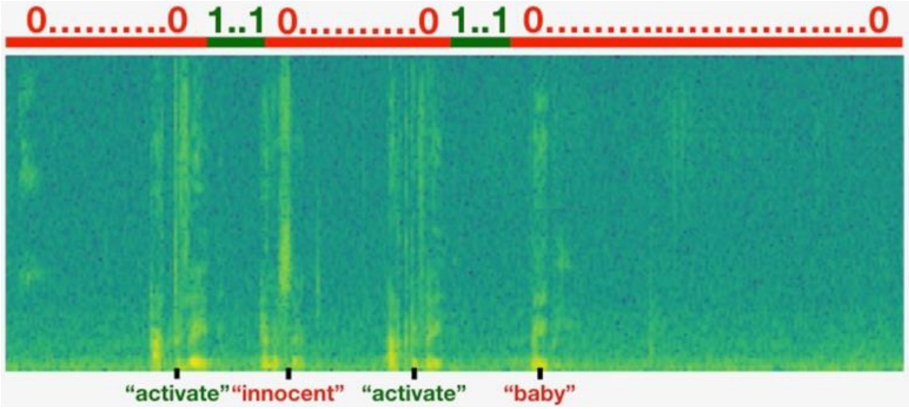

Abbildung 2.9: Die Annotationstechnik der Trigger Word Recognition. Zu sehen ist, dass wenn das Wort „activate“ im Spektrogramm vorkommt, eine Reihe von Einsen im dazugehörigen Label-Vektor folgen. [SDVS20]

3. Verwandte Arbeit

Eine automatische Sturzerkennung ist schon lange im Interesse der Gesellschaft. Es gibt schon viele Arbeiten mit unterschiedlichen Ansätzen. Im Folgenden soll die Diversität einiger interessanter Arbeiten in diesem Gebiet vorgestellt werden.

Beginnen möchte ich hier mit der Arbeit von Georgios Mastorakis und Dimitrios Makris [MM12]. Sie haben eine Sturzerkennung mit Hilfe von Microsoft's Kinect Kamera realisiert und verwendeten daher Tiefenkarten-Videos. Eine weitere Gemeinsamkeit zwischen den Daten dieser Arbeit und meinen besteht darin, dass auch hier die Personen farblich hervorgehoben wurden. In den Infrarot-Videodaten der Kinect werden mit Hilfe der Open-Source-Software Open Natural Interaction (OpenNI) [Ope12] Personen gefunden und um diese eine 3D-Bounding Box berechnet. Eine Bounding Box ist rechteckig und umfasst ein Objekt in einem Bild. Um einen Sturz zu erkennen wird hier nur eine 3D-Bounding Box verwendet. Sie berechnen aus dieser die Geschwindigkeit mit der sie sich kontrahiert oder expandiert. Dies sowohl in Bezug auf Höhe und Breite, als auch auf die Tiefe dieser. Mit ihrer Methode detektieren die Autoren einen Sturz, sobald die Geschwindigkeit einen Schwellwert überschreitet und anschließend einen tiefen Schwellwert unterschreitet. Das garantiert, dass keine False Positives durch beispielsweise ein schnelles Hinsetzen entstehen. Ein False Positive beschreibt ein falsch vorhergesagtes positives Ereignis im Bezug auf eine Klasse. In unserem Fall ist ein False Positive eine Handlung, welche fälschlich als Sturz erkannt wurde.
Laut den Autoren wurden alle Stürze erkannt und es entstanden keine False Positives.

Als Nächstes möchte ich auf die Arbeit von Anahita Shojaei-Hashemi, Panos Nasiopoulos und James J. Little eingehen [SHNLP18]. Hier wurde eine Sturzerkennung mit Hilfe von Transfer Learning erreicht. Dies ist eine Technik des Machine Learnings, bei dem gelerntes aus einem Modell auf ein Anderes, meist ähnliches Problem übertragen wird.
Die Autoren verwenden, wie ich, Tiefenbilddaten und nutzen für ihre Erkennung auch ein Rekurrentes Neuronales Netz mit LSTM-Layern. Für das letztendliche Modell wird zuerst ein Rekurrentes Neuronales Netz mit mehreren Klassen von menschlichen

Handlungen trainiert. Es enthält noch keine Klasse für Stürze. Die Klassen bestehen aus Aktionen wie beispielsweise gehen, trinken und aufheben.
Anschließend werden die gelernten Gewichte in ein neues Modell transferiert, welches sich nur im letzten Layer unterscheidet und nur noch zwischen Sequenzen mit Sturz und ohne differenziert. Dies wird dann noch weiter trainiert, wobei sich nur die Gewichte im letzten Layer anpassen. Zum Training werden hier auch Sturzsequenzen mit hinzugezogen. Eine Sturzsequenz beschreibt in unserem Beispiel eine Folge von Frames, in der eine Person beim Sturz zu sehen ist. Dieser Sturz besteht aus: nach dem Stehen hinfallen und liegen bleiben.
Die Autoren konnten hier eine Precision von 93.23% und einen Recall von 96.12% erreichen. Die Precision beschreibt, wie hoch der Anteil der korrekt vorhergesagten positiven Ereignisse an den vorhergesagten positiven Ereignissen, im Bezug auf eine Klasse ist.
Der Recall beschreibt, wie hoch der Anteil der korrekt vorhergesagten positiven Ereignisse an den insgesamt positiven Ereignissen, im Bezug auf eine Klasse ist.

Weiter gehe ich auf die Arbeit von Subhash Chand Agrawal, Rajesh Kumar Tripathi und Anand Singh Jalal ein [ATJ17]. Diese Technik zur Sturzerkennung wurde auch mit Hilfe einer Bounding Box erreicht. Bei den verwendeten Daten handelt es sich um normale Kamera-Videos im RGB-Format. In diesen werden mit Hilfe von Computer Vision Methoden aus der OpenCV Library [Ope] 2D-Bounding Boxen um Personen berechnet. Genauer verwenden sie eine „Contour based template Matching“ Funktion, welche einen „Score“ beim vergleichen, sowie eine Bounding Box mit ihren Parametern berechnet. Aufgrund der Freistellung der Person und des daraus resultierenden Schwarz-Weiß-Frames ähnelt dieser Ansatz meiner Datenvariante 6 der Personen Maske (4.2.6).
Um daraus einen Sturz zu erkennen, wird eine plötzliche Änderung des Contour based template Matching Scores, sowie eine Veränderung im Höhe zu Breite Verhältnis dieser und dem Abstand zwischen dem Mittelpunkt der Bounding Box und dem Mittelpunkt der oberen Kante der Bounding Box wahrgenommen. Treten diese drei Veränderungen ein, überschreiten einen Schwellwert, und zeigt anschließend die Bounding Box für 100 konsekutive Frames keine Veränderung, wird das Ereignis eines Sturzes erkannt.
Diese Methode konnte 20 von 21 Stürzen erkennen und führte bei 30 sonstigen menschlichen Aktivitäten nur zu einem False Positive. Das ist ein erreichter Recall und eine Precision von $\sim 95.24\%$

Bei der Arbeit von Homa Foroughi, Alireza Rezvanian und Amirhossien Paziraee handelt es sich um eine Sturzerkennung mit Hilfe einer Multi-Class Support Vector Machine [FRP08]. Hier werden wieder normale Video-Daten verwendet, welche auch durch eine Freistellung der Person vorbereitet werden. Es resultieren wieder Schwarz-Weiss-Frames und diese ähneln meiner Datenvariante 6 der Personen Maske (4.2.6). Weiter ist eine Verwandheit aufgrund des Machine Learning Ansatzes vorhanden. Als erstes werden Personen vom Hintergrund getrennt und dann einer Feature Extraction unterzogen. Die extrahierten Features sind eine approximierte Ellipse um den menschlichen Körper, sowie eines horizontalen und vertikalen „Projection Histograms“ und die Kopf Position.
Als Nächstes werden diese Features dann von Multi-Class Support Vector Machine

genutzt, um die Aktionen Zehn unterschiedlichen Klassen zuzuordnen. Diese sind unter anderem nach vorne, nach hinten oder seitlich fallen, sowie normale Aktionen wie rennen, hinsetzen usw.
Die Ergebnisse der Autoren zeigen, dass ihr Ansatz 325 von 360 Stürzen erkannt hat und dabei nur 27 False Positives unter 558 Sequenzen auftreten. Diese Ergebnisse ergeben einen Recall von $\sim 90.28\%$ und eine Precision von $\sim 92.33\%$

Die nächste Arbeit ist von Caroline Rougier, Jean Meunier, Alain St-Arnaud, und Jacqueline Rousseau [RMSAR11]. Die Sturzerkennung dieser Autoren ist mit Hilfe eines Gaussian Mixture Models (GMM) umgesetzt. Bei den verwendeten Daten handelt es sich hier um normale Kamera-Videos, welche in einer Laborumgebung mit Hilfe von 4 in den Ecken eines Raumes angebrachten Kameras aufgenommen wurden. Verwandt ist dieser Arbeit aufgrund des Machine Learning Ansatzes.
Als erstes extrahieren sie die Eckpunkte der Silhouette einer Person mit hilfe des Canny Edge detectors und zwei aufeinander folgenden Frames. Hier kann erneut eine Ähnlichkeit zu meiner Datenvariante 6 der Personen Maske (4.2.6) gezogen werden, da die Informationen der Person vom Hintergrund freigestellt werden.
Diese Form wird dann in allen Frames des Videos gematched, um Deformierungen der Silhouette zu erkennen. Weiter nehmen die Autoren zwei Werte um die Deformierung zu erkennen. Die durchschnittlichen matching Kosten und die „full Proctustes distance“. Mit Hilfe dieser Werte und einer Inaktivitätsphase nach dem Sturz wird ein GMM trainiert, welche Stürze als abnormale Aktionen erkennt. Die Autoren kombinieren hier vier Kameras, welche parallel jeweils ihre GMMs auf das Material anwenden. Die Ergebnisse aller GMMs werden dann kombiniert und die Kombination führt zur Entscheidung, ob ein Sturz aufgetreten ist.
Das Ergebnis ist ein Recall von $\sim 99.6\%$.

Als letztes hat die Gruppe um Caroline Rougier einen weiteren Artikel veröffentlicht, bei dem sie eine Methode zur Sturzerkennung mit Hilfe von Folgen von 3D Kopfpositionen einen Sturz erkennen [RMSAR06]. Für die Erkennung dieser verwenden die Autoren eine normale Kamera und drei Software Partikelfilter. Aus den aufeinander folgenden Kopfpositionen errechnen sie eine vertikale und horizontale Geschwindigkeit. Übersteigen diese Geschwindigkeiten einen Schwellwert, wird ein Sturz erkannt. Die Verwandheit der Arbeit kann hier aufgrund der selben Problemstellung gezogen werden.
Dieser Algorithmus erreicht einen Recall von $\sim 66.67\%$ und eine Precision von $\sim 85.71\%$. Diese Ergebnisse wurden mit einem Datensatz aus 9 Sturzvideos und 10 normalen Aktivitäten erreicht.

4. Datengrundlage

Zur Lösung des Problems der Sturzerkennung habe ich von unserer Partnerfirma ein Datenset bestehend aus Tiefenkarten-Videos zur Verfügung gestellt bekommen. Die Videos wurden mit der Kamera „The Intel® RealSense™ Depth Module D410“ aufgenommen([Cor19]). Diese Kamera erreicht ein Tiefenbild mit Hilfe von zwei im Stereoverfahren verwendeten Kameras ohne Infrarotfilter. In diesem Datenset befinden sich 28 Tiefenkarten-Videos, welche nachgestellte Sturz-Situationen beinhalten. Die Verwendung dieser hat den Vorteil, dass sie nicht durch die Änderung der Beleuchtung zwischen hell und dunkel beeinflusst werden. Genauer ist dies im Kapitel 2.1 beschrieben. Die Kameras erfassen zusätzlich zu dem für uns sichtbaren Licht, auch das Infrarotspektrum. Es ergibt also Sinn solche zu verwenden, da sie Nachts genauso effektiv Stürze einfangen, wie Tagsüber.
Es sind in 13 der 28 Videos Stürze vorhanden und in 15 nicht. Es handelt sich bei den Daten um Sequenzen, welche Bilddaten der Breite 320 Pixel und der Höhe 240 Pixel aufweist. Des weiteren sind sie mit einer Frames-Per-Second-Rate (FPS-Rate) von Zehn aufgenommen worden und haben eine durchschnittliche Länge von 57 Sekunden. Unter den Videos befindet sich ein Exemplar, welches mit einer FPS-Rate von 30 aufgenommen worden ist. Dieses wurde aufgrund Zeitmangels nicht mit ins Training einbezogen.
Die Daten liegen im Rot-Grün-Blau (RGB) Format vor, haben also drei Kanäle für jeden Pixel. Der Grund dafür ist die Hervorhebung der Personen, welche weiter unten erläutert wird.
Am 24.Juni 2021 sind mir 18 weitere Sequenzen hinzu gegeben worden. In diesen neuen Videos ist Eins, welches einen Sturz enthält. In den restlichen 17 Videos befand sich keine Sturzsequenz. Die FPS-Rate beträgt ebenfalls 10. Da die Videos erst später hinzu kamen, wurden sie erst ab Experiment 8 mit einbezogen, welches im Kapitel 5.3.8 beschrieben wird. Zum relevanten Zeitpunkt wird erneut darauf hingewiesen.
Die Videos zeigen die Umgebung aufgrund der Tiefenkarten in Graustufen. Bei der Umgebung handelt es sich dabei um Innenräume einer Wohnungseinrichtung. Treten Personen ins Bild, sind diese farblich hervorgehoben. Dies wurde durch eine Vorverarbeitung der Daten erreicht, welche bereits beim Erhalt des Datensatzes vorgenommen worden ist. Dabei ist zu unterscheiden, dass je nach Entfernung der Person von der Kamera eine blaue (nahe Entfernung), grüne (mittlere Entfernung)

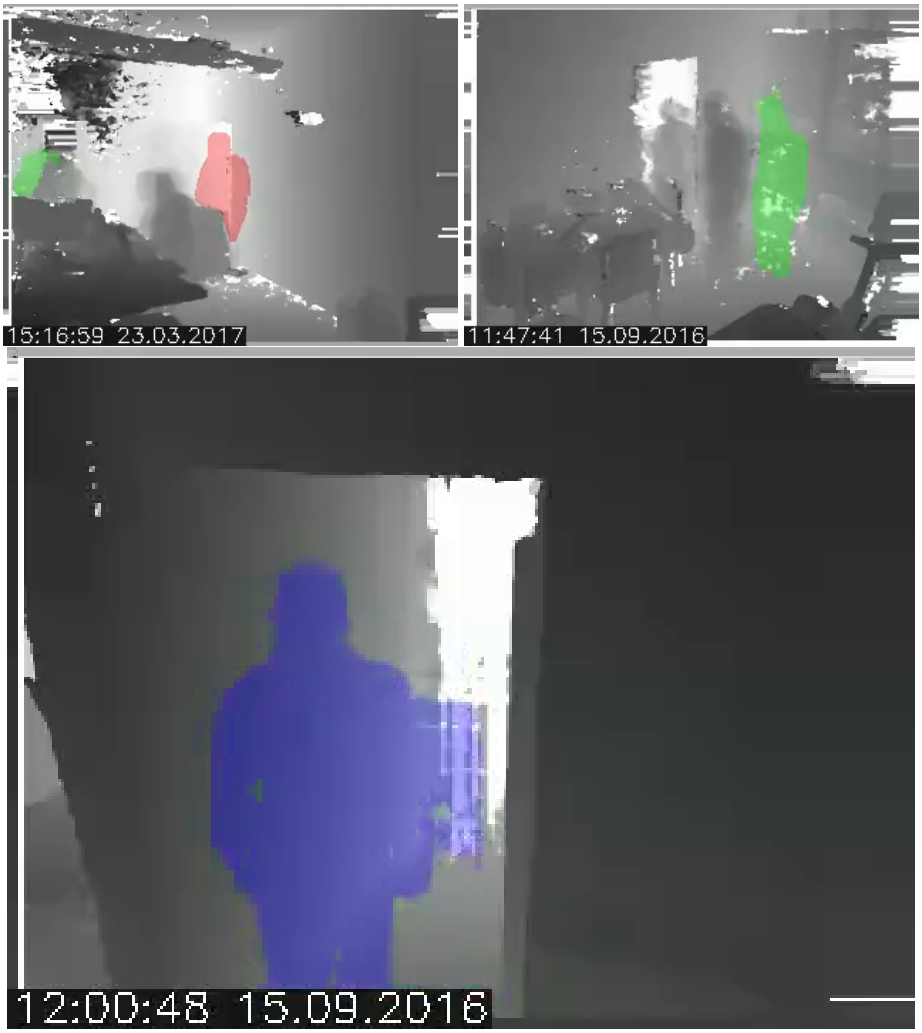

Abbildung 4.1: Es sind die drei unterschiedlichen Hervorhebungen der Personen zu sehen. Es wird zwischen naher (blau), mittlerer (grün) und weiter (rot) Entfernung zu Kamera unterschieden.

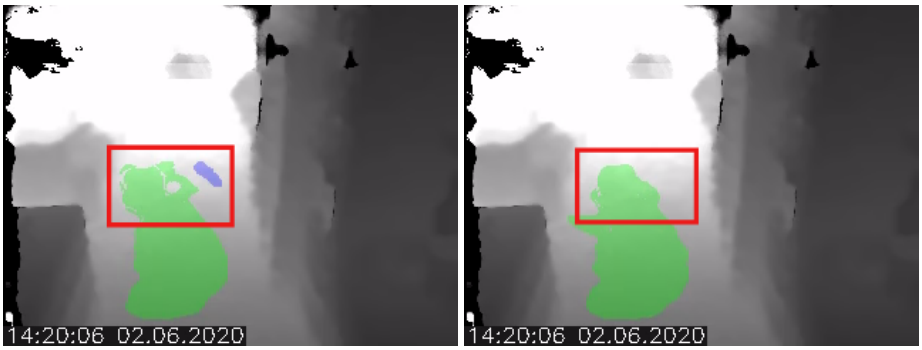

Abbildung 4.2: Ein Beispiel für die Noise in den Daten. Links zu sehen ist Frame 399 und rechts Frame 402 aus einer Datensequenz. Der blaue Fleck, sowie kleinere Bereiche um den Kopf herum sind Noise.

oder rote (weite Entfernung) Hervorhebung erfolgt ist. In Abbildung 4.1 werden die unterschiedlichen Hervorhebungen dargestellt.
Die Stürze in den Videos erfolgen in unterschiedliche Richtungen. Es gibt Stürze nach links, rechts, von der Kamera weg und zur Kamera hin. Des Weiteren ist zu erwähnen,

dass die Videos Noise beinhalten. Noise beschreibt ein Rauschen der Bildwerte in den Daten, welches zu falschen Ergebnissen führen kann. Manchmal flackert etwas in der Umgebung und öfter flackert die Hervorhebung der Person. Dies geschieht entweder hin und her, oder manche Bildteile fehlen von einem Frame zum nächsten. Ein Beispiel hierzu ist in Abbildung 4.2 zu sehen.

4.1 Annotation

Damit die Neuronalen Netze einen Zusammenhang zwischen Input und Output erkennen können, musste eine Annotation der Daten vorgenommen werden, welche trainierbare Informationen über die Stürze in den Sequenzen beinhaltet. Ich habe mich dabei für eine Annotation entschieden, welche an die TWR angelehnt ist. Diese ist im Kapitel 2.6 genauer erklärt. Es handelt sich dabei um einen Vektor, welcher einen Eintrag für jeden Frame in der Sequenz hat und zwischen Null und Eins unterscheidet.
Standardmäßig beinhaltet dieser Vektor nur Nullen. Ist jedoch eine Sturzsequenz annotiert, dann befindet sich eine Folge von Einsen im Vektor. Diese beginnt in dem Moment des Sturzes und endet nachdem die Person drei Sekunden lang auf dem Boden liegt. Diese Version der Annotation wird später gegen eine Längere getauscht. In der längeren Annotation reicht die Sturzsequenz bis zum erneuten aufstehen, oder dem Ende des Videos. Aus dieser Änderung resultierte ein größerer Anteil an zu Stürzen annotierten Frames. Der größere Anteil soll das Lernen vereinfachen und weiter ist das Ende der Sturzsequenz so weniger willkürlich gewählt.
Eine Übersicht über den Anteil der zu Stürzen annotierten Frames kann der folgenden Tabelle entnommen werden (4.1).
Des Weiteren habe ich mich dafür entschieden die Sequenzen auf eine Länge von 580 festzulegen. Somit besteht ein konstantes Many-to-Many-Relationship zwischen Eingabedaten und Label. Die kürzeren Sequenzen erfuhren ein Padding bis zu der Einheitslänge von 580 und die wenigen zu langen Sequenzen wurden abgeschnitten. Dabei gingen keine wichtigen, zu einer Sturzsequenz gehörenden Teile verloren.

	annotierte Frames	Alle Sequenzen	Nur Sturzvideos
Alte Annotation	730	$\sim 4.66\%$	$\sim 10.49\%$
Neue Annotation	2000	$\sim 12.77\%$	$\sim 28.74\%$
Mit neuen Daten	2180	$\sim 8.35\%$	$\sim 28.91\%$

Tabelle 4.1: Anteil der zu Stürzen annotierten Frames in den Sequenzen.

4.2 Datenvarianten

Im späteren Kapitel Experimente wird erklärt, dass beim Versuch das Problem zu vereinfachen der Datensatz angepasst wurde. Diese Anpassungen und die daraus resultierenden Datenvarianten sind in den folgenden Tabellen 4.2 und 4.3 in ihren unterschieden dargestellt. Dabei ist zu beachten, dass die aktualisierten Anzahlen durch die neuen Daten in Klammern aufgeführt werden.

Varianten	Nur Sturzvideos	Länge	Dimensionen
Var. 1: Alle Sequenzen, volle Länge	N	580	320x240x3
Var. 2: Nur Stürze, Volle Länge	J	580	320x240x3
Var. 3: 20 Sekunden Sequenzen	J / N	200	320x240x3
Var. 4: Reduzierte Kanäle	J / N	200	320x240x1
Var. 5: Reduzierte Kanäle und Pixel	J / N	200	?x ?x1
Var. 6: Personen Maske	J / N	200 / 580	320x240x1
Var. 7: Bounding Box	J / N	200 / 580	2
Var. 8: Quotient	J / N	200 / 580	1

Tabelle 4.2: Übersicht über die Datenvarianten. J & N stehen für Ja und Nein. Die Länge bezieht sich auf die Anzahl der zu einer Sequenz gehörenden Frames. Bei Variante 5 gibt es durch die Pixelreduktion keine feste Dimension.

	Anz. (Länge 580)		Anz. (Länge 200)	
Varianten	Alle Sequenzen	Nur Sturzvideos	Alle Sequenzen	Nur Sturzvideos
Var 1: ASVL	27 (45)			
Var 2: NSVL		12 (13)		
Var 3: 20SS			135 (225)	60 (65)
Var 4: RK			135 (225)	60 (65)
Var 5: RKUP			135 (225)	60 (65)
Var 6: PM	27 (45)	12 (13)	135 (225)	60 (65)
Var 7: BB	27 (45)	12 (13)	135 (225)	60 (65)
Var 8: Q	27 (45)	12 (13)	135 (225)	60 (65)

Tabelle 4.3: Übersicht über die Sequenzanzahlen in den Datenvarianten. Die Anzahl mit den neueren Daten ist in Klammern.

4.2.1 Var. 1: Alle Sequenzen, volle Länge

Variante Eins enthält alle Sequenzen, welche in voller Länge vorhanden sind. Dabei ist zu berücksichtigen, dass die meisten Videos weniger als 580 Frames enthalten. Diese werden bei der Datenaufbereitung, wie im Kapitel 4.1 erklärt, durch Padding mit Nullen aufgefüllt, sodass alle Sequenzen die selbe Länge haben. Die Bilddaten bleiben unverändert.
Die Daten dieser Variante liegen in den Dimensionen 580x320x240x3 vor. Initial bestand diese Variante aus 27 Sequenzen. Nach dem Erhalt der neuen Daten beträgt die Anzahl 45.

4.2.2 Var. 2: Nur Stürze, Volle Länge

Wie der Tabelle 4.1 entnommen werden kann, liegt der Anteil der zu einem Sturz annotierten Frames, bei der alten Annotation, im Verhältnis zur Gesamtzahl bei $\sim 4.66\%$ (Dieser Wert bezieht die neuen Daten noch nicht mit ein). Diese Variante bezieht nur die Videos mit Stürzen ein und kommt somit auf einen Anteil von $\sim 10.49\%$. Der erhöhte Anteil soll eine Fokussierung des Trainings auf lediglich die Sequenzen, welche Stürze enthalten bieten. Dabei entfallen 15 Sequenzen ohne Stürze. Die Frames selbst, sowie die Länge der Sequenzen werden wie bei Variante 1 behandelt. Durch die Änderung in der Annotation liegen die neuen Anteile bei $\sim 12.77\%$ für alle Sequenzen und $\sim 28.74\%$ bei der Fokussierung auf nur Sturzvideos. Nach Erhalt der neuen Daten kam hier ein Video mit und 18 Videos ohne Stürze

hinzu. Die Anteile der zu Stürzen annotierten Frames verschob sich hier auf $\sim 8.35\%$ mit allen Sequenzen und $\sim 28.91\%$ mit nur Sturzvideos.

4.2.3 Var. 3: 20 Sekunden Sequenzen

In dieser Variante wird eine Fensterung über die Daten vorgenommen. Hierbei werden nur Ausschnitte aus den Daten betrachtet. Diese Fenster überlappen und werden über eine feste Länge „weiter geschoben“ .Für die Größe des Fensters verwende ich 200 Frames, da ein Durchschnittlicher Sturz mit der alten Annotation bei ~ 60.83 und mit der neuen Annotation bei ~ 166.67 liegt. Das Fenster wird dabei um 100 weiter geschoben. Diese Aufteilung sorgt für eine verfünffachung der Sequenzanzahl. Im letzten Schritt der Fensterung bleiben 180 Frames der gefensterten Sequenz übrig, welche erneut durch Padding auf 200 gestreckt werden.
Durch diese Fensterung schrumpft die Dimension der Länge auf 34,5% der ursprünglichen Größe. Veränderungen an den Bilddaten werden hier nicht vorgenommen. Diese Fensterung ist mit beiden Vorherigen Varianten kombinierbar.

4.2.4 Var. 4: Reduzierte Kanäle

Da Tiefenkarten-Videos in Graustufen vorliegen und diese in den drei verschiedenen Farbkanälen fast überall die gleichen Werte aufweisen, lässt sich die Komplexität der Daten für unseren Zweck weiter verringern. Der Unterschied in den Kanälen liegt dabei in der Hervorhebung der Personen. Wie in Kapitel 4 erwähnt gibt es drei unterschiedliche Varianten, welche die Entfernung zur Kamera darstellen. Eine Visualierung kann der Abbildung 4.1 entnommen werden. Jede entspricht dabei einer Erhöhung der Werte im Bereich der Person befindlichen Pixel im zugehörigen Kanal der Farbe.
In dieser Datenvariante wird nur der Kanal mit der Hervorhebung übernommen und die anderen Verworfen. Der Unterschied zwischen den Werten in Kanälen ohne Hervorhebung und dem Kanal mit Hervorhebung kann der Abbildung 4.3 entnommen werden. Die daraus resultierenden Dimensionen sind 320x240x1. Diese Variante ist aufgrund Zeitmangels nur mit der gefensterten Variante kombinierbar. Eine Veränderung der Pixelwerte wurde hier nicht vorgenommen.

4.2.5 Var. 5: Reduzierte Kanäle und Pixel

Diese Variante erweitert den vorherigen Ansatz um eine Pixelreduktion, wobei als Parameter angegeben werden kann, wie stark reduziert werden soll. Zuerst wird wie in der vorherigen Variante eine Kanalreduktion vorgenommen. Danach folgt die Pixelreduktion, wobei die Vorgehensweise ist, jeden n-ten Pixel zu behalten und die dazwischen zu verwerfen. Wird also jeder zweiten Pixel gewählt, resultiert eine Reduktion der Dimension der Breite von 320 auf 160 und der Höhe von 240 auf 120. Die daraus resultierende Dimension wäre hier 160x120x1.
Aus einer Reduktion auf jeden fünften Pixel würde die Dimension 64x48x1 resultieren, und so weiter... Diese Reduktion ist in Abbildung 4.3 unten rechts zu sehen.
Auch hier wurde aufgrund Zeitmangels nur eine Kombination mit der Fensterungs-Variante erreicht.

Abbildung 4.3: Es ist ein Vergleich vom Originalframe (oben links), zu einem Kanal ohne Hervorhebung (oben rechts), dem Kanal mit Hervorhebung (unten links) und einer Pixelreduktion des Kanals mit der Hervorhebung (unten rechts) zu sehen. Die Pixel wurden auf jeden 5ten reduziert.

4.2.6 Var. 6: Personen Maske

Bei der Variante der Personen Masken werden die farblich hervorgehobenen Personen in den Videos vom Hintergrund getrennt (background subtraction). Dies bereitet die Daten auf die nächste Vereinfachung vor, ist in sich aber auch eine eigene Variante der Daten.
Ich bediene mich hier der Beschaffenheit der Daten. Diese liegen aufgrund des Tiefenkarten Charakters Hauptsächlich in Graustufen vor und können daher einfach von der farblichen Hervorhebung der Personen getrennt werden. Da Graustufen normalerweise nur einen Kanal benötigen würden, um das Schwarz-Weis-Spektrum in 256 verschiedenen Stufen darzustellen, sind die Pixelwerte bei Daten in Graustufen mit drei Kanälen in jedem Kanal gleich. Es liegen also drei Kanäle mit fast den gleichen Werten vor. Ein großer Unterschied ist hier die Hervorhebung der Personen. Aufgrund der Beschaffenheit der Daten lässt sich herausfinden, welcher der Kanäle die Hervorhebung enthält, da er an der Stelle der Person höhere Werte als die anderen Kanäle aufweist. Wird nun eine Kanal ohne Hervorhebung von dem Kanal mit der Hervorhebung subtrahiert, bleiben lediglich an der Stelle der Hervorhebung der Person Werte übrig. Nach Anwendung zweier Schwellwerte (für sehr kleine und sehr große Werte) sind diese Daten dann auch von der Noise der Umgebung befreit. Diese wurden im Kapitel 4 als Flackern und Rauschen beschrieben. Das Ergebnis dessen kann dann als Personen-Maske verwendet werden. Eine Visualisierung der Datenvariante kann der Abbildung 4.4 entnommen werden. Die resultierenden Dimensionen belaufen sich auf 320x240x1.

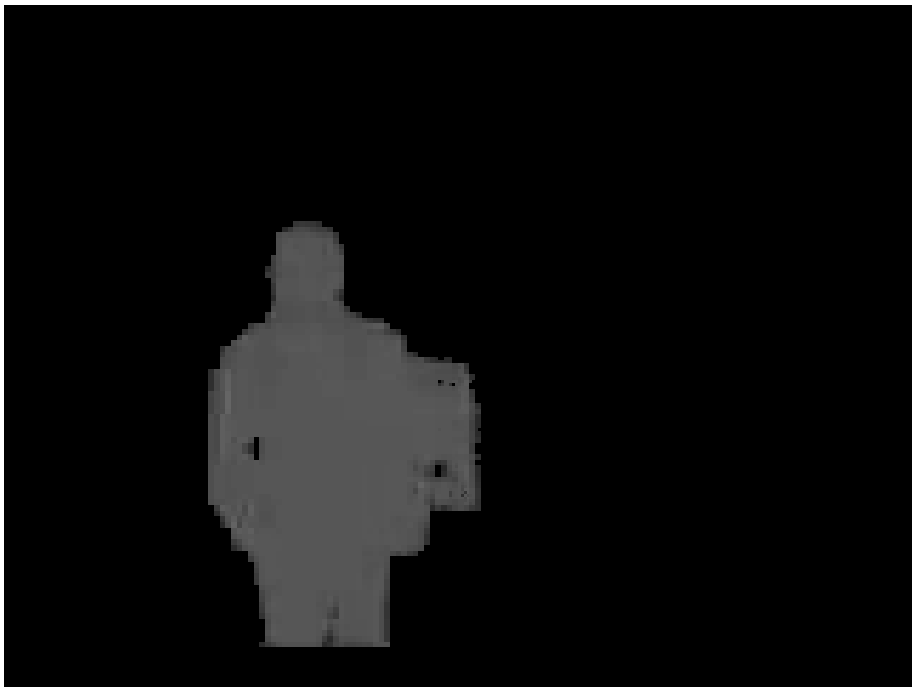

Abbildung 4.4: Die Anwendung der Personen Maske auf den Originalframe in Abbildung 4.3 (oben links).

4.2.7 Var. 7: Bounding Box

Mit Hilfe einer Bounding Box um die Person, lassen sich die hochkomplexen Daten auf zwei Werte pro Frame herunter brechen. Diese Werte beschreiben die Breite und Höhe der Bounding Box. Das ist deshalb nützlich, weil die Bounding Box um eine Person im Falle eines Sturzes in den meisten Fällen in der Höhe kleiner und in der Breite größer wird. Erreicht wird diese Bounding Box durch Funktionen, aus der OpenCV library [Ope].
Die Berechnungen finden auf der im vorherigen Kapitel 4.2.6 erklärten Personen-Maske statt. Das bedeutet, dass nur noch die Person freigestellt im Frame zu sehen ist. Ich wende eine Funktion an, welche alle Konturen im Frame finden soll und wähle davon die Größte.
Danach folgt eine Funktion, welche um die gewählte Kontur eine Bounding Box berechnet. Die Bounding Box besteht aus Ursprungs-Koordinaten und einer Höhe sowie Breite. Da die Koordinaten für unser Problem irrelevant ist, werden diese ignoriert. Lässt sich keine Bounding Box berechnen, weil keine Kontur gefunden werden konnte, werden Höhe und Breite auf 0 gesetzt.
Diese Variante reduziert die Dimension der Daten auf Zwei.

4.2.8 Var. 8: Quotient

Der Quotient bildet sich aus einer Division der Höhe und Breite der Bounding Box um eine Person. Diese wurde im vorherigen Kapitel 4.2.7 erklärt. So kann eine bestimmte Folge von Werten eines einzelnen Features als Sturz erkannt werden.
Zur Veranschaulichung: Ich nehme Startwerte von Höhe = 2 und Breite = 1. Der Quotient daraus wäre 2. Stürzt die Person nun, könnten die Werte Höhe = 1 und Breite = 1 auftreten. Der Quotient wäre hier 1. Kommt die Person zum liegen, könnten die Werte bei Höhe = 1 und Breite = 2 liegen. Der Quotient wäre also 0.5.

Die Folge: 2,1,0.5,0.5,... würde also auf einen Sturz hinweisen.
Diese Variante reduziert die zwei Werte der Bounding Box auf einen.

5. Algorithmus

5.1 Herangehensweise

Das Problem der Sturzerkennung in Videos lässt sich nicht auf die Klassifikation einzelner Frames vereinfachen, denn Stürze lassen sich nur anhand von sequenziellen Daten erkennen. Einzelne Frames würden hier aus dem Zusammenhang gerissen nur schwer Stürze abbilden können. Da diese in einem Zeitschritt nur eine Pose der Person zeigen und diese sich in alltäglichen Handlungen wie beispielsweise Sport, oder Hinsetzten auch finden ließe. Ein Sturz unterscheidet sich durch die Geschwindigkeit der Bewegung, wie auch dem Liegenbleiben nach dem Sturz von einer alltäglichen Handlung. Im Kapitel 3 bin ich bereits auf mehrere Ansätze eingegangen, welche dieses Problem mit Hilfe von Machine Learning und vor allem Rekurrenten Neuronalen Netzen umgesetzt haben. Es bieten sich deshalb an auch hier Sequenzmodelle zu verwenden.
Meine Idee ist es diese Sequenzen als ein Event in den Videos zu betrachten. Die Erkennung von Events in sequenziellen Daten existiert schon länger und wird beispielsweise bei Smart Speakern verwendet. Dort muss ein bestimmtes Wort oder eine bestimmte Phrase gesagt werden um den nachfolgenden Befehl auszuführen. Eine Problemlösung dafür ist die TWR, welche im Kapitel 2.6 näher beleuchtet wurde.
Um diese Idee umzusetzen, kombiniere ich bereits vorhandene Ansätze mit Sequenzmodellen und der TWR.
Diese werdem im laufe der Experimente anhand von wechselnden naiven Erkennungs-Schwellen verglichen. Da auch die jeweils verwendete Datenvariante wechselt und ein gewisses Maß an zufälligem mischen der Trainings- und Test-Sets vorgenommen wird, ändert sich auch diese. Sie ergibt sich aus dem Anteil der nicht zu einem Sturz annotierten Frames. Ein Sequenzmodell, welches immer vorhersagt, dass kein Sturz im jeweiligen Daten-Set vorhanden ist, würde genau diese naive Erkennungs-Schwelle treffen.

5.2 Umsetzung

Für die Umsetzung der Problemlösung wurde die Programmiersprache Python verwendet, da sie im Moment zum Standard für Machine Learning Projekte gehört

und eine große Vielfalt an Librarys für Machine Learning bereitstellt. Für die Rekurrenten Neuronalen Netze habe ich auf das Framework Keras zurückgegriffen [Ker]. Mit diesem Framework habe ich schon Erfahrung gesammelt und ein anderes Framework hätte den Zweck in ähnlichem Maße erfüllt. Alternativen zu Keras sind beispielsweise Pytorch [PyT] und scikit-learn [PVG+11]. Für das Handling der Daten selbst verwende ich die etablierte Library NumPy [HMvdW+20]. Diese Library ist durch ihre große Vielfalt an Array-Operationen zum etablierten Standard für Machine Learning Projekte mit Python geworden.
Wie im Kapitel 4.1 erklärt, habe ich die Annotation der Sturzsequenzen mit einer der TWR ähnlichen Herangehensweise vorgenommen.
In meinem Ansatz hingegen beginnt die Annotation nicht erst nachdem der Sturz geschehen ist, sondern schon beim Beginn der Sturzsequenz. Das ist deshalb so gewählt, weil das Trigger Word in einem Audioclip meist getrennt vom Redefluss und so als eigenständiges Ereignis erkannt wird. Bei einem Sturz kann es vorkommen, dass die Person mitten in einer anderen Aktion wie beispielsweise Gehen, Laufen und Putzen stürzt. Da eine Annotation nach dem Sturz hier dazu führen könnte, dass die eigentliche Aktion der Person mit zur Sturzsequenz hinzu gezählt wird, habe ich mich dazu entschieden die Annotation im Moment des Sturzes beginnen zu lassen. Abgesehen von der etwas abgewandelten Annotation ist meine Herangehensweise sehr ähnlich zu der der TWR.

5.3 Experimente

In diesem Kapitel wird der Prozess zur Entstehung des funktionsfähigen Algorithmus gezeigt. Es folgen aufeinander aufbauende Experimente, welche jeweils eine Hypothese überprüfen.

5.3.1 Experiment 1 - Initialer Test

Beim ersten Experiment wurde die erste Datenvariante verwendet. Ich testete, ob sich bereits beim lernen der originalen Datenvariante, gute Ergebnisse erzielen lassen. Wie im zugehörigen Kapitel 4.2.1 erklärt, handelt es sich hierbei um eine Variante, bei der alle Sequenzen mit der Länge von 580 Frames vorkommen. Jede Sequenz hat 580 Frames, welche eine Breite von 320 Pixeln und eine Höhe von 240 Pixeln aufweist. Jeder Pixel hat zudem drei Kanäle, da die Tiefenkarten typischen Graustufendaten im RGB-Format abgespeichert sind.
Ich überprüfte die Hypothese mit einem Rekurrenten Neuronalen Netz, welches die Input-Dimensionen (580,240,320,3) hat und auf eine Output-Dimension von (580,1) abbildet. Zu beachten ist hier, dass die 240 vor der 320 aufgezählt wird, da die Achsen im NumPy-Standard vertauscht sind.
Auf den Input folgt ein Convolutional-2D-Layer. Convolutional Neural Networks sind im Kapitel 2.5 genauer erklärt.
Dieser Layer hat den Zweck die Dimensionen der Frames herunter zu falten und auf kleine Bereiche, auch Kernel genannt, filter banks anzuwenden. Für die Größe des Kernels habe ich 80 Pixel gewählt, um so simpel wie möglich eine Veränderung der Pose der Person im Frame erkennen zu können und Personen im Frame nicht größer als 80 mal 80 Pixel vorkommen. Als Stride wählte ich 10 Pixel, da ein kleinerer Stride bei dieser Kernelgröße nicht nötig ist.
Damit tatsächlich eine Faltung vorgenommen wird, wurde kein Padding verwendet.

Ein Padding kann hier beispielsweise dazu verwendet werden, die Input-Dimensionen vor der Faltung auch danach zu erhalten. Ich verwende ein „valid-Padding“, bei dem die Dimensionen der Faltung entsprechend verändert werden. Auf die Kernel wurden dann jeweils vier Filter angewandt. Ein Beispiel ist hier der Sobel-Operator zur Kantenerkennung. Für eine anschauliche Erklärung des Sobel-Operators und anderer Kantenextraktionsverfahren empfehle ich [Wag06]. Jeder Kernel dieses Layers wendet dann alle Filter auf seinen Ausschnitt mit allen drei Kanälen an und die resultierenden feature maps werden an den nächsten Layer weitergegeben.
Die Anzahl Vier resultiert aus der Überlegung, dass bei Stürzen auf der grundlegendsten Art nur Vertikale, Horizontale und Diagonale Kanten erkannt werden müssen. Die Kanten können auch mit Posen der Person verglichen werden. Stürze von der Kamera weg und zur Kamera hin wurden hier vernachlässigt. Schließlich verwendete ich zur Sicherheit statt Drei, Vier Filter. Die aus dem Convolutional-Layer resultierende Dimension ist (580,17,25,4).
Nach dem Convolutional-2D-Layer folgten ein Batch-Normalization-/ und ein Dropout-Layer, um aufgrund der wenigen Daten ein Overfitting vorzubeugen.
Der Batch-Normalization-Layer wendet hier die Technik Normalisierung an. Normalisierung sorgt dafür, dass Datensätze mit zu großer Varianz zwischen den einzelnen Samples nicht dazu führen, dass die Gewichte des Netzes beim Training zu stark schwanken und durch die Normalisierung gezielter lernen können. Batch-Normalization wurde für die Normalisierung von Input-Features entwickelt, funktioniert aber auch zwischen den Hidden-Layern und sorgt dafür, dass der Durchschnitt und die Varianz der Layer-Inputs auf einen Durschnitt von Null und eine Varianz von Eins reguliert werden. Bei der Anwendung zwischen den Layern spricht man von Regularisierung statt Normalisierung, da nicht die Daten, sondern die Outputs der Layer angepasst werden. Weiter wird bei der Batch-Normalization von der Reduzierung des „Internal Covariate Shifts“ gesprochen. Für eine genauere Erklärung bitte im Original nachlesen [IS15]
Beim Dropout-Layer wird eine Regularisierung umgesetzt, indem ein „bestimmter Anteil“ der eintreffenden Werte zufällig auf Null gesetzt werden. Dieser „bestimmte Anteil“ wird über den Parameter Drop-Rate bestimmt. Ich habe hier eine Drop-Rate von 0.2 gewählt. Auf diese beiden Layer folgte dann ein Activation-Layer, welcher die Rectified Linear Unit (ReLu) Activation Function verwendet hat. Eine Activation Function ist eine nicht-lineare mathematische Funktion, welche auf die Outputs eines Layers angewandt wird. Beispiele für solche Funktionen sind „Tanh“ (2.2), „Sigmoid“ (2.1) und „ReLu“ (2.3). Ich verwende hier die ReLu Activation Function, weil sie generell sehr beliebt ist und außerdem auch in ähnlicher Art und Weise bei der TWR verwendet wurde. Die Verwendung von ReLu sorgt dafür, dass keine negativen Werte gehalten werden, welche im Output-Layer auch nicht vorkommen.
Nach diesen Layern folgt der rekurrente Teil, welcher die gefalteten und gefilterten Werte der einzelnen Frames in einem temporalen Zusammenhang setzt. Als Vorbereitung für den LSTM-Layer folgt zuerst ein Reshape-Layer, welcher die Dimensionen des vorherigen Layers Frame-weise konkateniert. Aus der vorherigen Dimension (580,17,25,4) wird nun (580,1700), da sie anders nicht in den LSTM-Layer passen. Die Funktionsweise der LSTM-Layer ist im Kapitel 2.4 beschrieben.
Der LSTM-Layer hat nur eine Unit für jeden temporalen Zeitschritt und daher werden aus den 1700 Werten pro Schritt nur einer. Es resultiert eine Dimension von (580,1). Ich verwende hier den LSTM-Layer, da er ein starkes Werkzeug ist, um einen temporalen Zusammenhang zwischen weit entfernten Zeitschritten zu erlernen. In

diesem Experiment habe ich den LSTM-Layer außerdem gegen den Gated Recurrent Unit (GRU)-Layer und einen Time-Distributed-Convolutional-Layer augetauscht. Der GRU-Layer ist eine Vereinfachung des LSTM-Layers und somit sehr verwandt. Der Time-Distributed-Convolutional-Layer hätte den temporalen Zusammenhang und die Faltung des Inputs in einem Schritt zusammenfassen können, führte allerdings zu deutlich mehr trainierbaren Parametern. Dabei kam ich zu dem Schluss, dass die verwandte Arbeit eher auf den LSTM-Layer zurückgreift [SHNLP18] und entschied mich für den genannten.
Darauf folgt ein normaler Dense-Layer und ein Activation-Layer, welcher auch die ReLu-Activation Function verwendet. Der Dense-Layer ist die Keras bezeichnung für einen Fully Connected-Layer. Eine Übersicht über die aus den jeweiligen Layern resultierenden Dimensionen ist in Tabelle 5.1 zu sehen. Die resultierenden Dimensio-

Layer	Dimension
Input	(580,240,320,3)
Convolutional	(580,17,25,4)
Batch-Norm.	(580,17,25,4)
Dropout	(580,17,25,4)
Reshape	(580,1700)
LSTM	(580,1)
Dense	(580,1)

Tabelle 5.1: Aus den Layern resultierende Dimensionen (Exp. 1)

nen aus den Activation-Layern wurde hier nicht beachtet, da Activation Functions die Dimensionen nicht beeinflussen. Dieser Netzaufbau hat 83,622 trainierbare Parameter. trainierbare Parameter sind die Gewichte zwischen den Layern. Anders als die Hyperparameter werden sie bis auf bei der Initialisierung nicht vom Entwickler beeinflusst. Die Gewichte werden zu Beginn auf so genannte Initialisierungsvektoren gesetzt. Je nach Wahl des Gradient Descent-Verfahrens, werden diese anders gewählt. Trainiert wurde das Netz mit dem Adam-Optimizer [KB15]. Der Adam-Optimizer ist eine erweiterte Form für Gradient Descent und momentan sehr beliebt. Als Parameter wurden eine Learning Rate von 0.0001, ein $\beta 1$ von 0.9, $\beta 2$ von 0.999 und eine decay-rate von 0.01 verwendet. Bei diesen Werten habe ich mich durch die Arbeit der TWR inspirieren lassen [SDVS20]. Sie wurden in allen weiteren Experimenten, wie auch dem finalen Algorithmus so beibehalten.
Als Loss-Funktion wurde binary-crossentropy verwendet. Auch bei dieser Wahl habe ich mich bei der Arbeit der TWR inspirieren lassen. Das Training dauerte 10 Epochen.
Es resultierte eine Trainings-Set Accuracy von $\sim 95.16\%$ und eine Test-Set Accuracy von $\sim 95.81\%$. Die Accuracy beschreibt wie präzise ein Machine Learning Modell die designierten Label der zugehörigen Daten vorhersagen konnte. In unserem Beispiel habe ich die binary Accuracy verwendet, welche bei einem Schwellwert von 0.5 zwischen richtigen und falschen Vorhersagen unterscheidet. Müsste ein Frame beispielsweise das Label 1 haben und es wurde 0.3 Vorhergesagt, zählt dieser Frame als falsch vorhergesagt. Analog wäre eine Vorhersage von 0.6 hier richtig vorhergesagt. Diese Werte gleichen sich mit den Werten einer naiven Erkennung, welche immer sagt, dass kein Sturz vorkommt. Es liegt ein Anteil von $\sim 95.81\%$ nicht zu einem Sturz annotierten Frames im Test-Set vor. Dies deckt sich mit den Vorhersagen des

trainierten Modells. Des Weiteren hat sich die Accuracy auch im laufe der Epochen beim Training nicht verändert.

5.3.2 Experiment 2 - Nur Stürze

Da die Ergebnisse des vorherigen Experiments die naiven Erkennungs-Schwellen nicht übertreffen konnten, prüfte ich, ob eine Vereinfachung des Problems in Form von einem höheren Anteil an zu einem Sturz annotierten Frames bessere Ergebnisse erzielt. Wie im Kapitel 4.2.2 der Datenvariante 2 erklärt, wurden die Sequenzen in denen keine Stürze vorkommen weggelassen.
Hier wurde nur eine Änderung an der Datenvariante vorgenommen. Deshalb wurde das selbe Netz mit den selben Parametern beim Training verwendet.
Das Netz ist also immer noch so aufgebaut, dass es mit einem Convolutional-Layer beginnt. Dieser hat die Kernelgröße 80, einen Stride von 10 und 4 Filter. Es wird valid-padding verwendet.
Darauf folgt der Batch-Normalization-/ und der Dropout-Layer mit der Drop-Rate 0.2.
Als nächstes ein Activation-Layer mit der Activation Function ReLu, gefolgt vom für den LSTM-Layer benötigten Reshape-Layer.
Zum Schluss dann der LSTM-Layer mit einer Unit, welcher von einem Dense-Layer, gefolgt von einem Acvitvation-Layer mit der Activation Function ReLu auf den Output abbildet. Eine Übersicht über die aus den Layern resultierenden Dimensionen ist in der Tabelle 5.1 dargestellt. Auch hier hat das Netz 83,622 trainierbare Parameter.
Der naive Erkenner würde jetzt im Trainings-Set eine Accuracy von $\sim 87.33\%$ erreichen und mein Modell kann diesen Wert decken. Beim Test-Set liegt die Schwelle bei $\sim 86.95\%$, welche mein Modell mit $\sim 88.77\%$ um $\sim 1.82\%$ übersteigen kann. In den Vorhersagen ist sichtbar, dass vereinzelt Werte die 0.5 Hürde überschreiten und somit laut der binary Accuracy als richtig vorhergesagt gelten.

5.3.3 Experiment 3 - Kürzere Sequenzen

Das letzte Experiment zeigte, dass sich etwas aus den Daten lernen lässt. Weitergehend prüfte ich nun, ob das Problem der Sturzerkennung mit meinem Ansatz in kürzeren Sequenzen einfacher zu lösen ist. Wie im Kapitel 4.2.3 zur Datenvariante 3 erklärt, wurden die Daten gefenstert und auf eine Länge von 200 Frames, also 20 Sekunden mit einer Frames-Per-Second-Rate von 10 beschränkt. Aufgrund dessen verkürzt sich im ansonsten gleich bleibenden Neuronalen Netz die zweite Dimension, welche hier vereinfacht als „zeitliche Dimension“ betrachtet werden kann, auf 200. Diese Variante ist mit den Varianten 1 und 2 kombinierbar.
Zur Veranschaulichung sind die aufeinander folgenden Dimensionen der Layer in Tabelle 5.2 dargestellt. Die trainierbaren Parameter belaufen sich hier trotz der Änderungen auf 83,622.

Beim Test mit allen Sequenzen konnte dabei die naive Erkennungs-Schwelle im Trainings-Set von $\sim 84.57\%$ nicht überschritten werden. Im Durchlauf mit dem Test-Set hingegen lag die zu übertreffende Schwelle mit $\sim 94.83\%$ etwas höher. Diese konnte mit $\sim 95.71\%$ um $\sim 0.88\%$ überschritten werden.
Bei der Variante mit nur Stürzen konnte dabei auch im Trainings-Set die naive Erkennungs-Schwelle von $\sim 72.22\%$ mit $\sim 73.63\%$ um $\sim 1.41\%$ überboten werden.

Layer	Dimension
Input	(200,240,320,3)
Convolutional	(200,17,25,4)
Batch-Norm.	(200,17,25,4)
Dropout	(200,17,25,4)
Reshape	(200,1700)
LSTM	(200,1)
Dense	(200,1)

Tabelle 5.2: Aus den Layern resultierende Dimensionen (Exp. 3)

Außerdem kam es auch im Test-Set zu einer Überschreitung dieser von $\sim 68.39\%$ um $\sim 3.61\%$.

5.3.4 Experiment 4 - Reduzierte Kanäle

Nachdem das letzte Experiment die naiven Erkennungs-Schwellen sowohl im Trainings- als auch im Test-Set überschreiten konnte, prüfte ich, ob sich diese Ergebnisse mit einer weiteren Vereinfachung der Daten noch steigern ließen. Außerdem testete ich, ob das Netz aufgrund der wenigen Daten besser lernt, wenn es weniger trainierbare Parameter hat.
Dafür verwendete ich die Datenvariante 4. Hier nutzte ich die Eigenschaft, dass es sich bei den Bilddaten um Graustufen handelt, welche dann jeweils einen Kanal haben, der sich von den anderen beiden unterscheidet. Das ist so, weil er die Hervorhebung der Person beinhaltet. Die genaue Erklärung befindet sich in Kapitel 4.2.4. Weiter wurde hier wieder eine Fensterung vorgenommen.
Da bei den Experimenten mit Datenvarianten, in denen nur Sequenzen mit Stürzen zum Training benutzt wurden bessere Ergebnisse erzielt werden konnten, beschränke ich mich hier auf das Experiment mit nur eben diesen.
Aus dieser Kanal-Reduktion der Daten resultierte eine Schmälerung des Input-Layers. Die daraus resultierenden Dimensionen sind (200,240,320,1). Alle weiteren Dimensionen des Netzes blieben dabei wie in Tabelle 5.2. Das hat zur Folge, dass der Convolutional-Layer und damit auch das Netz insgesamt weniger trainierbare Parameter hat. Statt der 83,622 trainierbaren Parameter hat das Netz jetzt 32,422.
Die restlichen Bestandteile des Netzes wurden nicht verändert und die besten Ergebnisse konnten mit 20 Epochen Trainingsdauer erzielt werden.
Im Trainings-Set wurde die naive Erkennungs-Schwelle von $\sim 71.65\%$ um $\sim 2.42\%$ überschritten. Im Test-Set wurde die Schwelle von $\sim 70.11\%$ in ähnlichem Maße um $\sim 2.39\%$ überschritten.

5.3.5 Experiment 5 - Reduzierte Kanäle und Pixel

Die Vereinfachungen des letzten Experiments zeigten eine höhere Überschreitung der naiven Erkennungs-Schwellen. In diesem Experiment prüfte ich daher, ob eine weitere Vereinfachung der Daten zu noch leichter trainierbaren Daten führt. Dafür verwendete ich die Datenvariante 5, welche in Kapitel 4.2.5 näher erläutert ist. Diese nimmt zusätzlich zur Reduktion der Kanäle außerdem eine Reduktion der Pixelwerte vor. So werden beispielsweise nur jeder zweite, vierte, oder fünfte Pixel übernommen und der Rest nicht.

Durch die Reduktion der Pixelwerte verkleinerte sich der Input-Layer weiter. Ein Experiment, bei dem ich die Pixel auf jeden fünften reduzierte, führte dazu, dass der Input-Layer auf (200,48,64,1) schrumpfte. Ich verwendete hier erneut die Fensterung, damit kürzere Sequenzen das Training so einfach wie möglich gestalten. Außerdem bezog ich hier auch die Sequenzen mit ein, in denen keine Stürze vorkommen. Diese Reduktion erlaubte es mir, den Convolutional-Layer auch um das selbe Verhältnis (5 zu 1) herunter zu skalieren. statt der Kernelgröße 80 und dem Stride 10, verwendete ich nun eine Kernelgröße von 16 und einen Stride von 2. Die Veränderungen in den Dimensionen sind in Tabelle 5.3 dargestellt. Diese Anpassungen der Daten führten

Layer	Dimension
Input	(200,48,64,1)
Convolutional	(200,17,25,4)
Batch-Norm.	(200,17,25,4)
Dropout	(200,17,25,4)
Reshape	(200,1700)
LSTM	(200,1)
Dense	(200,1)

Tabelle 5.3: Aus den Layern resultierende Dimensionen (Exp. 5)

zu einer weiteren Reduzierung der trainierbaren Parameter von 32,422 auf 7,846.

Nach einem Training mit 50 Epochen Dauer, konnte die Trainings-Set Erkennungs-Schwelle von $\sim 85.09\%$ nicht überschritten werden. Im Test-Set wurde jedoch eine Überschreitung von $\sim 93.35\%$ um $\sim 1.65\%$ erzielt.

5.3.6 Experiment 6 - Personen Maske

Im Kapitel 3 habe ich über Arbeiten gesprochen, welche eine Sturzerkennung mittels einer Bounding Box um die stürzende Person realisiert [MM12] [ATJ17]. Um zu testen, ob sich diese Lösung auch mit meinem Datensatz realisieren lässt, muss zuerst eine Personen Maske berechnet werden. Dieser Zwischenschritt selbst stellt auch eine eigenständige Datenvariante dar und ist im Kapitel 4.2.6 zur Datenvariante 6 (4.2.6) erklärt. Außerdem haben auch weitere verwandte Arbeiten Daten verwendet, die dieser Variante ähnlich sind [ATJ17] [FRP08] [RMSAR11]. Da in dieser Variante der Hintergrund entfernt wurde, verschwindet auch die Noise der Umgebung, welche zu Problemen beim Training führen kann.
Für die Durchführung des Experiments entschied ich mich dafür nur Sequenzen mit Stürzen zu verwenden und eine Fensterung vorzunehmen, da mit dieser Kombination bisher die besten Ergebnisse erzielt werden konnten, wenn man die Überschreitungen der naiven Erkennungs-Schwellen betrachtet. Das Neuonale Netz blieb ähnlich wie bei Experiment 4 (5.3.4). Als einzige Änderung fügte ich einen Dense-Layer zwischen Reshape-/ und LSTM-Layer ein, um zu sehen, ob der LSTM-Layer den temporalen Zusammenhang besser lernen kann, wenn er statt 1700 Werten pro Frame nur einen als Input bekommt. Die aus den Änderungen resultierenden neuen Ausgangsdimensionen der Layer sind in Tabelle 5.4 dargestellt. Die aus diesem Netz resultierenden trainierbaren Parameter belaufen sich auf 27,327. Das liegt an der Kanalreduktion wie bei Experiment 4 (5.3.4) und dem Dense-Layer vor dem LSTM-Layer. Im Durchlauf mit dem Trainings-Set konnte die naive Erkennungs-Schwelle von $\sim 72.22\%$ um

Layer	Dimension
Input	(200,240,320,1)
Convolutional	(200,17,25,4)
Batch-Norm.	(200,17,25,4)
Dropout	(200,17,25,4)
Reshape	(200,1700)
Dense	(200,1)
LSTM	(200,1)
Dense	(200,1)

Tabelle 5.4: Aus den Layern resultierende Dimensionen (Exp. 6)

$\sim$ 10.8% überschritten werden. Außerdem wies der Durchlauf mit dem Test-Set eine Accuracy von $\sim$ 72.33% auf, was die naive Erkennungs-Schwelle von $\sim$ 68.39% um $\sim$ 3.94% übersteigt.

5.3.7 Experiment 7 - Bounding Box

In diesem Experiment teste ich, ob der Ansatz aus [MM12] und [ATJ17], welche im Kapitel 3 näher erläutert sind, auch mit unseren Daten funktioniert. Diese verwenden nämlich eine Bounding Box um die Person. Ich nutze dafür die Datenvariante 7, welche in Kapitel 4.2.7 näher erläutert ist. Die Datenvariante hat nur zwei Werte für jeden Frame in der Sequenz. Daher kann das Neuronale Netz auch deutlich kleiner sein. Am besten funktionierte hier wieder die Kombination aus Fensterung und nur Sturzsequenzen, bevor die Bounding Boxen berechnet wurden.
Für dieses Experiment baue ich das Neuronale Netz wie folgt auf: Auf den Input-Layer der Dimension (200,2) folgt ein Dense-Layer mit 2 Units als Basis. Danach folgt ein LSTM-Layer mit 64 Units. Das ist deshalb so, weil ich einen deutlich höheren Fokus auf die temporalem Zusammenhänge legte. Danach folgt ein Dense-Layer mit einer Unit und ein Activation-Layer mit der Activation Function ReLu (2.3). In Tabelle 5.5 ist auch hier eine Übersicht zu den aus den Layern resultierenden Dimensionen. Das

Layer	Dimension
Input	(200,2)
Dense	(200,2)
LSTM	(200,64)
Dense	(200,1)

Tabelle 5.5: Aus den Layern resultierende Dimensionen (Exp. 7)

Netz ist hier so simpel wie möglich gehalten. Dennoch legte ich einen großen Fokus auf die temporalen Zusammenhänge der Zeitschritte. Dieses Netz hat aufgrund des großen LSTM-Layers 17,223 trainierbare Parameter
Nach 500 Epochen Training, konnte die naive Erkennungs-Schwelle im Trainings-Set von $\sim$ 77.59% um $\sim$ 9.57%, sowie die Schwelle im Test-Set von $\sim$ 52.3% um $\sim$ 6.7% überschritten werden.

5.3.8 Experiment 8 - Quotient

Ab hier wurden die neuen Daten verwendet, welche mir am 24. Juni 2021 zu den ursprünglichen Datenstücken hinzu gegeben wurden.

Da der Versuch, mit weniger komplizierten Daten bessere Ergebnisse zu erzielen, nach den Überschreitungen naiven Erkennungs-Schwellen von Erfolg geprägt war, versuchte ich in diesem Experiment die Daten noch ein letztes mal zu vereinfachen. Dies erfolgte mit der Datenvariante 8, welche in Kapitel 4.2.8 erläutert ist.
Da die zwei Werte pro Frame nun auf einen verringert wurden, habe ich auch das Netz an der Basis geschmälert. Es folgt auf den Input-Layer der Dimension (200,1) nun ein Dense-Layer mit einer Unit. Ansonsten wurde das Netz wieder mit nur Sturz Sequenzen und einer gefensterten Variante trainiert, da auch hier die Kombination dieser Techniken die besten Ergebnisse lieferte. Eine Übersicht der aus den Layern resultierenden Dimensionen ist in Tabelle 5.6 dargestellt. Hier belaufen sich die

Layer	Dimension
Input	(200,1)
Dense	(200,1)
LSTM	(200,64)
Dense	(200,1)

Tabelle 5.6: Aus den Layern resultierende Dimensionen (Exp. 8)

trainierbaren Parameter auf 16,963

Das Training dauerte hier erneut 500 Epochen. Es konnte im Trainings-Set eine Überschreitung der naiven Erkennungs-Schwelle von $\sim 67.24\%$ um $\sim 1.5\%$ erreicht werden. Im Test-Set lag die Überschreitung der Schwelle von $\sim 83.33\%$ bei $\sim 6.84\%$.

5.4 Funktionsfähiger Algorithmus

Aus Kapitel 5.3 lässt sich Schlussfolgern, dass manche Kombinationen aus Datenvarianten zu höheren Überschreitungen der naiven Erkennungs-Schwellen und damit einem besseren Trainings-Effekt führen. Diese Kombinationen sind: Nur Sturzsequenzen verwenden, eine Fensterung der Daten vornehmen und den Hintergrund samt Noise mit der Variante der Personen Maske entfernen (4.2.6). Die Varianten der Bounding Box (5.3.7) und des Quotienten (5.3.8) werden hier trotz Überschreitungen in den Ergebnissen vernachlässigt, da viele verwandte Arbeiten in Kapitel 3 ein Datenset verwenden, welches meiner Variante der Personen Maske stark ähnelt [ATJ17] [FRP08] [RMSAR11] und dies auch bei mir hohe Überschreitungen zeigte.
Für das Training des funktionsfähigen Algorithmus verwende ich nach oben erläuterten Schlussfolgerungen die Datenvariante 6 (4.2.6). In Kapiel 5.3.6 verwende ich ein Neuronales Netz, auf dem dieser Algorithmus aufbaut. In der Abbildung 5.1 ist das verwendete Modell anschaulich dargestellt. Es beginnt mit einem Convolutional-Layer, welcher einen Kernel der Größe Vier und einen Stride von Eins verwendet. Es werden hier wie schon vorher Vier Filter angewandt. Dieser Layer soll simple Features wie Kanten erkennen. Diese sowohl horizontal, vertikal als auch diagonal. In den vorherigen Experimenten folgte auf den Input direkt ein Convolutional-Layer mit einem Kernel der Größe 80. Hier jedoch nutze ich einen im Machine Learning gewohnteren Ansatz, bei dem erst kleinere Kernels und später größere verwendet werden. Das hat zur Folge, dass zu Beginn simplere Features wie Kanten und später komplexere Features wie Personen von den Filtern erkannt werden. Die Convolutional-Layer bauen in diesem Sinne aufeinander auf. Aus diesem Grund ist ein Kernel der Größe Vier eine gute Größe für Kanten.

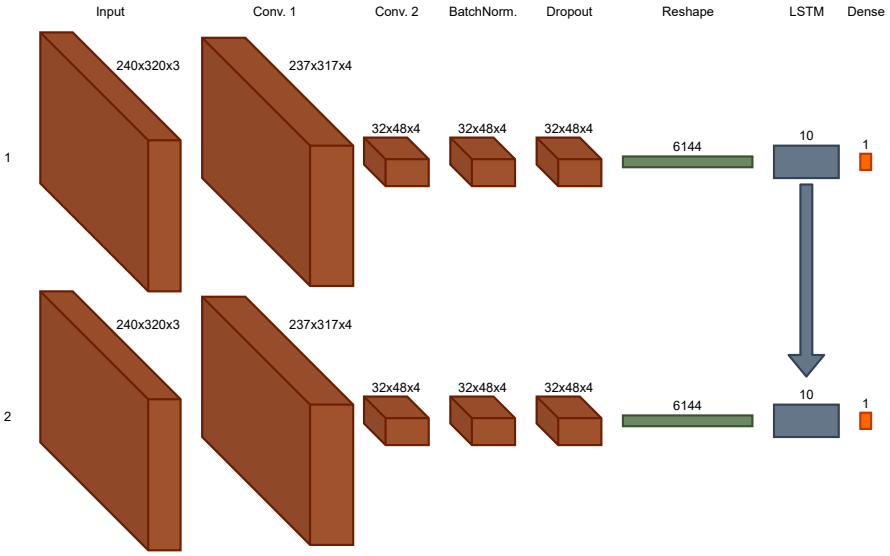

Abbildung 5.1: Eine Visualisierung des funktionsfähigen Algorithmus. Von links nach rechts sieht man die auf die einzelnen Frames angewandten Layer. Von oben nach unten finden die Zeitschritte statt. Der Einfachheit halber sind nur Zeitschritt Eins und Zwei dargestellt. Zu beachten ist, dass der LSTM-Layer des zweiten Schrittes nicht nur vom vorherigen Reshape-Layer, sondern auch vom LSTM-Layer des ersten Zeitschrittes beinflusst wird. Die Activation Functions sind nicht dargestellt. Diese Abbildung wurde mit Hilfe von www.diagrams.net erstellt [JGr]

Darauf folgt ein Convolutional-Layer, welcher einen Kernel mit einer Größe von 80 und einem Stride von Fünf hat. Wie in Kapitel 5.3 erklärt, soll dieser Layer die Person in ihrer Gesamtheit während dem Sturz umfassen. Auch hier wurden Vier Filter verwendet, da für einen Sturz der Übergang von vertikalen- über diagonale- zu horizontalen Kanten erkannt werden soll. Die Größe 80 ist hier wieder aufgrund der in den Videos auftretenden Personen zu begründen und hat sich in vorherigen Experimenten bewährt.
Darauf folgen die bereits verwendeten Layer zur Regularisierung, BatchNormalization und Dropout. Der Dropout-Layer wurde hier wieder mit einer Drop-Rate von 0.2 verwendet. Auf diese Layer folgt ein Activation-Layer, welcher die Activation Function ReLu verwendet (2.3). Diese dient dazu, dass keine negativen Werte aus den Convolutional-Layern in die rekurrente Schicht der LSTMs weiter getragen werden.
Als nächstes folgt ein Reshape-Layer, welcher die Dimension des Netzes durch Frameweise Konkatenation anpasst, um sie für den LSTM-Layer vorzubereiten. Dieser kann die höherdimensionalen Outputs der Convolutional-Layer nicht direkt annehmen.
Es folgt der LSTM-Layer, welcher mit Zehn Units ausgestattet ist. Die Anzahl der Units ergibt sich hier aus einer schrittweisen Verringerung von experimentellen 100 Units. Die Verringerung veränderte die Ergebnisse nicht, sorgte jedoch für eine deutliche Verringerung der trainierbaren Parameter und somit einem geringeren Rechenaufwand.
Es folgt ein Dense-Layer mit einer Unit und ein Activation-Layer mit erneut der ReLu Activation Function. Der Dense-Layer sorgt hier dafür, dass der Output des LSTM-Layers von (200,10) auf den Output mit (200,1) passt. Eine Übersicht über die aus den Layern resultierenden Dimensionen ist in Tabelle 5.7 zu sehen. Bei dieser Netzarchitektur belaufen sich die trainierbaren Parameter auf 348.691. Das Netz wird

Layer	Dimension
Input	(200,240,320,1)
Convolutional	(200,237,317,4)
Convolutional	(200,32,48,4)
Batch-Norm.	(200,32,48,4)
Dropout	(200,32,48,4)
Reshape	(200,6144)
LSTM	(200,10)
Dense	(200,1)

Tabelle 5.7: Aus den Layern resultierende Dimensionen (Funktionsfähiger Algorithmus)

in 20 Epochen trainiert. Es übersteigt beim Training die naive Erkennungs-Schwelle von $\sim 72\%$ mit $\sim 96.59\%$ um $\sim 24.59\%$. Bei den ungesehenen Test-Daten kann das Netz die naive Erkennungs-Schwelle von $\sim 75.25\%$ mit $\sim 91.28\%$ um $\sim 16.03\%$ überbieten. Eine Visualisierung des Verlaufs des Trainings kann der Abbildung 5.2 entnommen werden.

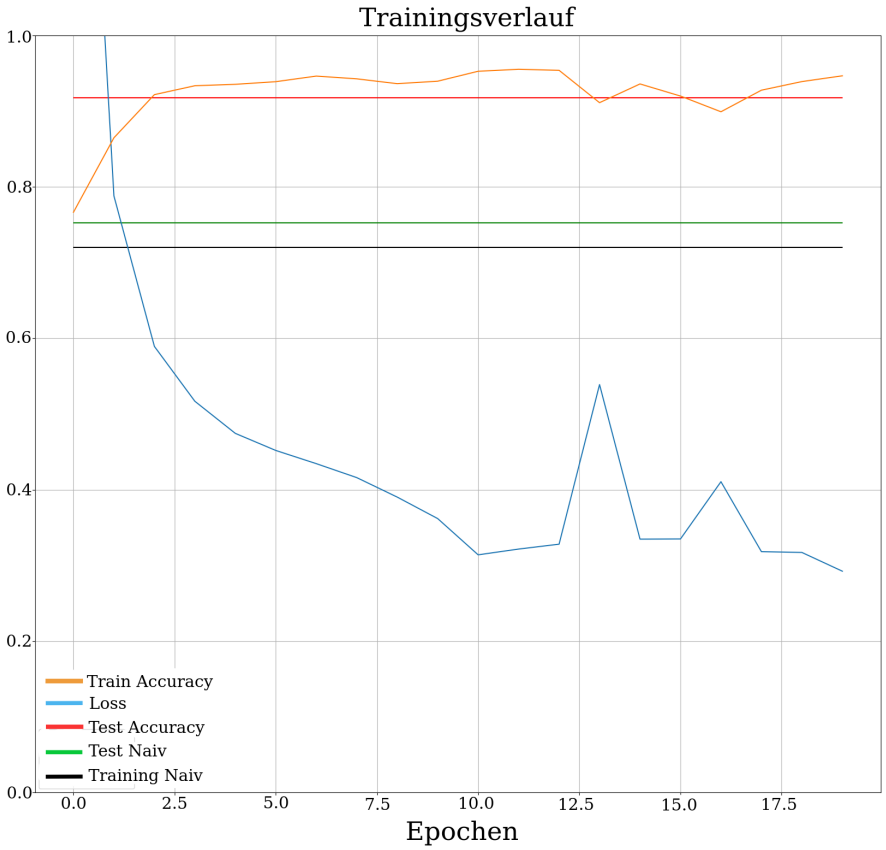

Abbildung 5.2: Der Trainingsverlauf des funktionsfähigen Algorithmus. Es ist die Veränderung des Losses(blau) und der Accuracy(orange) über die Anzahl der Epochen in X-Richtung zu sehen. Außerdem sind die naiven Erkennungs-Schwellen des Trainings-(schwarz) und des Test-Sets(grün), sowie der erreichten Test-Set Accuracy(rot) unabhängig der Epochen als Konstanten angegeben. Diese Grafik wurde mit Hilfe von Matplotlib erstellt [Hun07].

6. Ergebnisse

In Kapitel 5.4 über den funktionsfähigen Algorithmus wurde bereits die Accuracy am Ende des Trainings für das Trainings-Sets erwähnt. Diese konnte die naive Erkennungs-Schwelle von $\sim 72\%$ mit $\sim 96.59\%$ um $\sim 24.59\%$ übersteigen. Beim Test-Set, also den Daten, auf denen das Netz nicht trainiert hat, konnte außerdem ein Überschreiten der naiven Erkennungs-Schwelle von $\sim 75.25\%$ mit $\sim 91.28\%$ um $\sim 16.03\%$ erreicht werden. Um die Performanz des Algorithmus allerdings im nächsten Kapitel bewerten zu können, müssen noch weitere Metriken hinzugezogen werden.
Zur Erinnerung, die Forschungsfragen lauten: „Lässt sich eine automatische Erkennung von Stürzen mit Hilfe von Rekurrenten Neuronalen Netzen in Tiefenkarten-Videos realisieren? Wenn ja, können alle Stürze erkannt werden?“.
Die erste Frage lässt sich beantworten, sobald ein True Positive in den Vorhersagen über das Test-Set enthalten ist. Ein True Positive beschreibt ein korrekt vorhergesagtes positives Ereignis im Bezug auf eine Klasse. In unserem Fall ist ein True Positive ein erkannter Sturz. Für die zweite Frage kann ich die häufig verwendete Metrik Recall verwenden. Da die Anwendung in unserem Fall nicht trivial ist, definiere ich zwei Herangehensweisen an einen für mein Problem passenden Recall.
Als Erstes kann der Sturz und die dazu gehörige Annotation als Event betrachtet werden. Es kann beispielsweise ein Sturz von Frame 60-140 annotiert sein. Der Sturz ist dann 80 Frames lang. Sagt man jetzt, dass der Sturz erkannt wurde, wenn sich in der Vorhersage eine Folge von 8 korrekt vorhergesagten Frames befindet, kann die Vorhersage als True Positive definiert werden.
Ein weiterer Ansatz wäre es zu sagen, dass eine Vorhersage ein True Positive ist, wenn beispielsweise unsere 80 zu einem Sturz annotierten Frames zu 8% von korrekt vorhergesagten Frames überlappt werden.
Die Schwellwerte hierfür werden aus einer Statistik über die Daten berechnet. So ist in der verwendeten Datenvariante der Personen Maske (4.2.6) beim funktionsfähigen Algorithmus beispielsweise der kleinste Sturz von der Länge 10 annotierte Frames. Der durchschnittliche Sturz ist hier 100 Frames lang. Möchte man nun, dass der kürzeste Sturz erkannt wird, sollte der Schwellwert für ein True Positive hier mit kleiner Toleranz bei 8 liegen. Um eine in etwa vergleichbare Metrik mit dem prozentualen Ansatz zu haben, wähle ich dort 8%, da 8 Frames beim Event basierten Ansatz 8%

des Durchschnittlichen Sturzes sind.
Beim anwenden dieser Metriken auf meinen Algorithmus mit der Datenvariante der Personen Maske (4.2.6), komme ich zu dem Ergebnis, dass im Trainings-Set ein Event basierter Recall von 87.5% und ein prozentual basierter Recall von 95.83% erreicht werden kann. Im Test-Set kann ein Recall von 70% bei beiden Metriken erreicht werden. Die False Negatives in den Vorhersagen über das Test-Set treten hier immer dann auf, wenn in der Vorhersage kein einziges, oder nur sehr wenig Frames als zu einer Sturzsequenz gehörig vorhergesagt wurde. Ein False Negative beschreibt ein fälschlich nicht vorhergesagtes positives Ereignis im Bezug auf eine Klasse. In unserem Fall ist ein False Negative ein nicht erkannter Sturz.
Des Weiteren verwende ich eine traditionelle Berechnung von scikit-learn[PVG+11], welche in Form eines Classification Reports die Hauptsächlich verwendeten Metriken zur Bewertung eines Klassifikators übersichtlich zusammenfasst. Dieser liefert die Precision, den Recall und den F-Score in Bezug auf die Klassen Null und Eins. Der F-Score beschreibt die Performanz eines Algorithmus bei der Klassifikation eines Datensatzes. Er wird oft als das harmonische Mittel von Precision und Recall betrachtet.
Zur Erinnerung, Null bedeutet hier, dass ein Frame nicht zu einer Sturzsequenz gehört und Eins bedeutet, dass ein Frame zu einer Sturzsequenz gehört. Ich betrachte aufgrund der Forschungsfrage hier nur die Klasse der Einsen.
Im Trainings-Set konnte hier eine Precision von 93%, ein Recall von 94% und ein F-Score von 94% erreicht werden. Im Test-Set erreichte der Algorithmus eine Precision von 90%, einen Recall von 73% und einen F-Score von 81%.

7. Diskussion

In diesem Kapitel soll es um die Beantwortung der Forschungsfragen gehen. Zur Erinnerung, die Forschungsfragen lauten: „Lässt sich eine automatische Erkennung von Stürzen mit Hilfe von Rekurrenten Neuronalen Netzen in Tiefenkarten-Videos realisieren? Wenn ja, können alle Stürze erkannt werden?“.
Die Antwort auf die erste Frage fällt leicht. Es lässt sich ein True Positive in den Vorhersagen des funktionsfähigen Algorithmus finden. Wäre dies nicht so, könnte der Recall, welcher in Kapitel 6 besprochen wurde, nicht höher als 0% sein. Es konnte jedoch mit jeder besprochenen Variante des Recalls ein höherer Wert aufgezeigt werden und damit lässt sich sagen, dass eine automatische Erkennung von Stürzen mit Hilfe von Rekurrenten Neuronalen Netzen in Tiefenkarten-Videos möglich ist.
Um die Zweite Frage zu beantworten kann erneut der Recall zur Argumentation verwendet werden. Wenn jeder Sturz erkannt werden soll, muss der Recall in zumindest einer Variante die 100% erreichen. Dies konnte jedoch trotz des höchsten Recalls auf ungesehenen Daten von 73% (Classification Report von scikit-learn [PVG+11]) nicht erreicht werden.
Das könnte daran liegen, dass der Datensatz verhältnismäßig klein ist, da mehr Beispiele einen größeren Teil der möglichen Szenarien von Sturzsequenzen abdeckt. Eine Anpassung der Daten könnte auch zu besseren Ergebnissen führen, da die Variante der Personen Maske (4.2.6) beispielsweise schon deutlich bessere Ergebnisse gegenüber der unveränderten Datenvariante (4.2.1) aufweisen konnte.
Des Weiteren kann es daran liegen, dass das Netz für die Lösung des Problems noch nicht ausgereift und nicht fein justiert genug ist. Es lässt sich mit den aktuellen Informationen nicht sagen, ob beispielsweise eine Anpassung des Netzes, oder das Sammeln von mehr Daten zu einem Recall von 100% führt. Ich überlasse eine eindeutige Beantwortung der Frage der zukünftigen Arbeit. Im Bezug auf meinen Algorithmus kann ich allerdings sagen, dass mit Argumentation des erreichten Recalls von bisher 73% auf ungesehenen Daten nicht alle Stürze erkannt werden konnten.
Beim Vergleich meines Algorithmus mit denen aus der Verwandten Arbeit (3), fällt auf, dass hier keine leichte Aussage über die Performanz getroffen werden kann. Da alle Arbeiten einen anderen Datensatz verwendet haben, besteht für einen Vergleich untereinander keine Basis.
Möchte man jedoch die Performanz der Arbeiten auf den jeweiligen verwendeten

Datensatz vergleichen, kann man sagen, dass die Arbeit von Georgios Mastorakis und Dimitrios Makris [MM12] einen Recall von 100% erreicht und damit eine für ihren Datensatz perfekte Performanz erreicht. Die Arbeit von Anahita Shojaei-Hashemi, Panos Nasiopoulos und James J. Little [SHNLP18] erreicht bei ihrem Datensatz einen Recall von 96.12%. Weiter konnte die Arbeit von Subhash Chand Agrawal, Rajesh Kumar Tripathi und Anand Singh Jalal [ATJ17] einen Recall für ihren Datensatz von 95.24% erreichen. Zu nächst erreichte die Arbeit von Homa Foroughi, Alireza Rezvanian und Amirhossien Paziraee [FRP08] einen Recall von 90.28% auf ihren eigenen Daten. Und zuletzt konnten die Arbeiten von Caroline Rougier, Jean Meunier, Alain St-Arnaud, und Jacqueline Rousseau einen Recall von 99.6% [RMSAR11] und 66.67% [RMSAR06] auf den jeweiligen Daten erreichen.
Im Vergleich zu den Arbeiten zum jeweiligen Datensatz schneidet meine Arbeit mit einem Recall von 73% nicht besonders gut ab, allerdings könnte eine Weiterführung der Arbeit in der Zukunft dieses Ergebnis erheblich verbessern.

8. Fazit & Ausblick

Zusammenfassend lässt sich sagen, dass der Ansatz eine TWR in Kombination mit Rekurrenten Neuronalen Netzen zu verwenden, mit Argumentation der Ergebnisse, funktioniert hat. Es wurden Stürze erkannt, jedoch konnten leider nicht alle als Solche erkannt werden. Aufgrund dieser Ergebnisse wäre eine Weiterfühung der Arbeit in der Zukunft wünschenswert. Da die Technik der TWR aus der Spracherkennung kommt, kann die Anwendung auf das Problem der Sturzerkennung als Transferleistung betrachtet werden.
Aus Kapitel 7 geht hervor, dass die Forschungsfrage, ob alle Stürze erkannt werden können, noch nicht endgültig beantwortet werden kann. Um sich dieser anzunehmen, kann versucht werden, mit Hilfe von Anpassungen des Neuronalen Netzes ein besseres Ergebnis beim Recall zu erreichen. Es könnten hier beispielsweise mehr Convolutional-Layer hinzugefügt, oder die Parameter der einzelnen Layer abgeändert werden.
Weitergehend kann die Datenvielfalt durch das Hinzufügen neuer Sequenzen ausgebaut werden, damit mehr mögliche Situationen von Sturzsequenzen abgebildet werden. Eine weitere Möglichkeit das Datenset auszubauen besteht in der Data Augmentation. Dies ist eine Technik, bei der einzelne Daten leicht abgeändert dem Datensatz als Variationen ihrer selbst hinzugefügt werden. Sie erweitert den Datensatz und führt zu Normalisierung der Daten, was einem Overfitting von wenigen Daten entgegenwirkt. Hier kann beispielsweise das Spiegeln an der vertikalen Achse neue mögliche Sequenzen erzeugen.
Der vorgestellte Algorithmus löst das Problem der Sturzerkennung nicht vollständig, allerdings kann er als Grundlage für weiterführende Arbeit dienen.

A. Glossar

Accuracy Die Accuracy beschreibt wie präzise ein Machine Learning Modell die designierten Label der zugehörigen Daten vorhersagen konnte. 6, 7, 28, 29, 32, 35, 37

Activation Function Eine Activation Function ist eine nicht-lineare mathematische Funktion, welche auf die Outputs eines Layers angewandt wird. Beispiele für solche Funktionen sind „Tanh“, „Sigmoid“ und „ReLU“. 5, 9, 27–29, 32, 34

Backpropagation Die Backpropagation beschreibt einen Prozess beim Training eines Neuronalen Netzes, bei dem die Gewichte der Kanten zwischen den Layern aufgrund ihres höheren oder niedrigeren Einflusses auf die Loss-Funktion angepasst werden. Geschieht nach der Fowardpropagation. 7

Batch Als Batch wird die Menge an Samples bezeichnet, welche gleichzeitig an Forwardpropagation und Backpropagation teilnehmen kann, um die Rechenleistung effizienter auszunutzen. 7

Bounding Box Eine Bounding Box ist rechteckig und umfasst ein Objekt in einem Bild. 13, 14, 23, 24, 31–33

Classification Report Der Classification Report fasst die hauptsächlich verwendeten Metriken zur Bewertung eines Klassifikators übersichtlich zusammen. Diese sind Recall, Precision und F-Score. 38, 39

Curse of dimensionality Curse of dimensionality beschreibt ein Problem, bei dem hochkomplexe Daten mit ihren vielen features für Machine Learning Modelle zunehmend schwerer zu erlernen und zu unterscheiden sind. 5

Data Augmentation Ist eine Technik, bei der einzelne Daten leicht abgeändert dem Datensatz als Variationen ihrer selbst hinzugefügt werden. Sie erweitert den Datensatz und führt zu Normalisierung der Daten, was einem Auswendig lernen von wenigen Daten entgegenwirkt. 41

Epoche Eine Epoche beschreibt einen Schritt beim Training eines Neuronalen Netzes. Sie besteht aus Forward- und Backward-Propagation mit Hilfe des Trainings-Sets. 7, 28–32, 35

F-Score Der F-Score beschreibt die Performanz eines Algorithmus bei der Klassifikation eines Datensatzes. Er wird oft als das harmonische Mittel von Precision und Recall betrachtet. 38

False Negative Ein False Negative beschreibt ein fälschlich nicht vorhergesagtes positives Ereignis im Bezug auf eine Klasse. In unserem Fall ist ein False Negative ein nicht erkannter Sturz. 38

False Positive Ein False Positive beschreibt ein falsch vorhergesagtes positives Ereignis im Bezug auf eine Klasse. In unserem Fall ist ein False Positive eine Handlung, welche fälschlich als Sturz erkannt wurde. 13–15

Feature Features sind Merkmale von Daten. So kann ein Apfel beispielsweise das Feature: Farbe - Rot haben. 5, 10, 14, 23, 27, 33

Fensterung Eine Fensterung beschreibt einen Prozess, bei dem Ausschnitte aus Daten betrachtet werden. Diese Fenster überlappen oft und werden über eine feste Länge „weiter geschoben“ . 21

Fitting Fitting wird beim Machine Learning als Begrifflichkeit dafür benutzt, wie gut ein Machine Learning Modell zwischen Klassen von Objekten unterscheiden kann. Dabei wird zwischen Under- und Overfitting unterschieden. 6

Forwardpropagation Die Forwardpropagation beschreibt einen Prozess beim Training eines Neuronalen Netzes, bei dem die Trainingsdaten vom Input-Layer, über die Hidden-Layer zum Output-Layer verrechnet und anschließend mit dem designierten Ausgabewert verglichen werden. Die Abweichung von diesem Wert wird dann als Loss bezeichnet. 6

FPS-Rate Frames-Per-Second-Rate. 17, 29

Frame Als Frame wird ein Bild in aus einer Videosequenz bezeichnet. Er hat bis auf den Ersten und Letzten Frame einen Vorgänger und einen Nachfolger. 14, 15, 18–21, 23, 25–29, 31–34, 37, 38

Framework Ein Framework stellt ein Grundgerüst dar, mit Hilfe dessen Entwickler aus vorprogrammierten Bausteinen ein Programm konstruieren können. v, 1, 26

GMM Gaussian Mixture Model. 15

Gradient Descent Gradient Descent ist eine Optimierungsfunktion aus dem Machine Learning mit Hilfe dessen die Gewichte eines Machine Learning Modells so angepasst werden, dass die Loss-Funktion bestenfalls zu einem globalen Minimum findet. Während diese zu einem globalen Minimum konvergiert, wird versucht die Accuracy zu maximieren. Für die Umsetzung wird eine Learning Rate verwendet, welche die Schrittgröße bestimmt. 6, 7, 28

GRU Gated Recurrent Unit. 11, 28

Hyperparameter Hyperparameter beeinflussen mit ihren Werten das Training. Anders als trainierbare Parameter sind sie vom Entwickler bestimmt. 6, 28

Hyperplane Die Hyperplane beschreibt im Machine Learning eine n-dimensionale Grenze welche die durch ein Machine Learning Modell getrennten Klassen separiert. 6

Label Als Label wird im Machine Learning die Zuordnung eines Datums zu einer Klasse bezeichnet. So kann Beispielsweise ein Apfel das Label Eins und eine Birne das Label Zwei haben. 6, 7, 11, 19, 28

Layer Gebräuchlicher Begriff für eine Schicht in einem Neuronalen Netz. 5–7, 10, 11, 13, 14, 27–35

Learning Rate Die Learning Rate bestimmt die Schrittgröße bei einer Optimierungsfunktion wie dem Gradient Descent. 6, 28

Library Eine library bezeichnet in der Informatik eine Programmierbibliothek, welche Methodiken für Lösungen von Problemen enthält. 14, 26

Loss Der Loss beschreibt die Differenz zwischen disigniertem Output und einer per Fowardpropagation der Trainingsdaten errechneten Vorhersage, mit Hilfe des zu trainierenden Neuronalen Netzes. Die Kombination der einzelnen Losse der Trainingsdaten wird dann Loss-Funktion genannt. 6, 7, 35

Loss-Funktion Die Loss-Funktion beschreibt die Kombination der einzelnen Losse, welche durch Forwardpropagation der Trainingsdaten erreicht werden. 6, 7, 28

LSTM Long Short-Term Memory. 3, 8, 9, 13, 27–29, 31, 32, 34

Noise Beschreibt ein Rauschen der Werte in den Daten, welches zu falschen Ergebnissen führen kann. 18, 19, 22, 31, 33

OpenNI Open Natural Interaction. 13

Overfitting Beschreibt den Zustand, bei dem ein Machine Learning Modell sich zu sehr an die features der Trainingsdaten angeglichen hat und somit nicht mehr wirklich über neue Daten generalisieren kann. 6, 27, 41

Padding Beschreibt das Erweitern, oder auch Auffüllen von Daten mit einem ausgewählten Wert, sodass diese in eine gewünschte Form gebracht werden können. 19–21, 26, 27

Precision Die Precision beschreibt, wie hoch der Anteil der korrekt vorhergesagten positiven Ereignisse an den vorhergesagten positiven Ereignissen, im Bezug auf eine Klasse ist. 14, 15, 38

Recall Der Recall beschreibt, wie hoch der Anteil der korrekt vorhergesagten positiven Ereignisse an den insgesamt positiven Ereignissen, im Bezug auf eine Klasse ist. v, 14, 15, 37–41

ReLu Rectified Linear Unit. 5, 10, 27–29, 32, 34

RGB Rot-Grün-Blau. 3, 14, 17, 26

Sample Ein Sample ist ein Datum aus einem Datensatz. In unserem Beispiel ist ein Sample eine Videosequenz. 7, 27

Sturzsequenz Beschreibt in unserem Beispiel eine Folge von Frames, in der eine Person beim Sturz zu sehen ist. Dieser Sturz besteht aus: nach dem Stehen hinfallen und liegen bleiben. 14, 17, 19, 26, 32, 38, 39, 41

Tiefenkarte Eine Tiefenkarte (engl. Depth Map) gibt an, wie weit das Sichtfeld von der Kamera entfernt ist. Dabei werden die Entfernungen pixelweise unterteilt. v, 1, 3, 4, 13, 17, 21, 22, 26, 37, 39

TOF Time-of-Flight. 3

trainierbare Parameter Trainierbare Parameter sind die Gewichte zwischen den Layern. Anders als die Hyperparameter werden sie bis auf bei der Initialisierung nicht vom Entwickler beeinflusst. 28–30, 32

Training Der Begriff Training beschreibt den Prozess in dem die Gewichte des Neuronalen Netzes iterativ an eine Mapping-Funktion von Input-Daten zu Output-Label angepasst werden. 6, 7, 11, 14, 17, 20, 27–33, 35, 37, 38

Transfer Learning Transfer Learning ist eine Technik des Machine Learnings, bei dem gelerntes aus einem Machine Learning Modell auf ein Anderes, meist ähnliches Problem übertragen wird. 13

True Positive Ein True Positive beschreibt ein korrekt vorhergesagtes positives Ereignis im Bezug auf eine Klasse. In unserem Fall ist ein True Positive ein erkannter Sturz. 37, 39

TWR Trigger Word Recognition. v, 3, 19, 25–28, 41

Underfitting Beschreibt den Zustand, bei dem ein Machine Learning Modell sich zu wenig an die features der Trainingsdaten angeglichen hat und somit nicht wirklich zwischen den Klassen unterscheiden kann. 6

Unit Units sind die einzelnen Bausteine, aus denen sich beispielsweise Layer in einem Neuronalen Netz zusammensetzen. 5, 6, 8, 10, 27, 29, 32–34

Danksagung

Zuerst möchte ich mich recht herzlich bei Tanja Schultz für die Möglichkeit bedanken, diese Bachelorarbeit zu schreiben. Dann gilt mein Dank Joachim Clemens für die Übernahme der Pflichten des Zweitgutachters.
Weiter möchte ich Yale Hartmann und Hui Liu für die Betreuung danken. Gerade in den schwierigen Phasen konnte ich auf ihre Motivation und Tipps, sowie Ideen bauen.
Danke außerdem an meine Eltern Klaus und Katja Lüdemann, die mir das Studium ermöglicht haben und mit dessen Unterstützung ich immer rechnen konnte.
Hilfe bei der Korrektur erhielt ich von meinem Bruder Kevin Lüdemann, sowie Mirco Meyer, bei denen ich mich nicht nur für diese, sondern auch für die generelle Unterstützung während des Studiums bedanken möchte.
Zu guter Letzt gilt mein Dank der Gruppe von Kommilitonen, mit der ich so Vieles gemeinsam erledigen konnte. Wir haben in den unterschiedlichsten Kursen auf die Unterstützung der anderen bauen können. Danke Clara Maria Odinius, Dennis Riemer, Timo Hoheisel und Niklas Masemann.

Literaturverzeichnis

[ATJ17] Subhash Chand Agrawal, Rajesh Kumar Tripathi, and Anand Singh Jalal. Human-fall detection from an indoor video surveillance. *8th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, pages 1–5, 2017.

[Bun20] Bundesministerium für Gesundheit. Stürze bei älteren Menschen. https://gesund.bund.de/stuerze-aeltere-menschen, 2020. Aufgerufen: 06.08.2021.

[Cor19] Intel Corporation. Intel realsense d400 series product family. https://www.intel.com/content/dam/support/us/en/documents/emerging-technologies/intel-realsense-technology/Intel-RealSense-D400-Series-Datasheet.pdf, Januar 2019. Aufgerufen: 06.08.2021.

[FRP08] Homa Foroughi, Alireza Rezvanian, and Amirhossien Paziraee. Robust Fall Detection Using Human Shape and Multi-class Support Vector Machine. *Sixth Indian Conference on Computer Vision, Graphics & Image Processing*, pages 413–420, 2008.

[HMvdW+20] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020. Aufgerufen: 16.08.21.

[Hoc91] Josef Hochreiter. Untersuchungen zu dynamischen neuronalen Netzen, Juni 1991. Diplomarbeit.

[HS97] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, November 1997.

[Hun07] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[IS15] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *Pro-*

ceedings of the 32nd International Conference on Machine Learning, PMLR, 37:448–456, 2015.

[JGr] JGraph Ltd Artisans'. diagrams.Net. https://www.diagrams.net/. Aufgerufen: 03.09.2021.

[KB15] Diederik P. Kingma and Jimmy Lei Ba. Adam: A Method for Stochastic Optimization. *Conference paper at 3rd International Conference for Learning Representations, San Diego*, 2015.

[Ker] Keras. Keras. https://keras.io/. Aufgerufen: 16.08.2021.

[LBH15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 512:436–444, Mai 2015.

[Mar20] Marco. Was ist ein IR-Cut Filter (Infrarotfilter)? https://www.wlan-kamera.info/artikel/was-ist-ein-ir-cut-filter-infrarotfilter, Juli 2020. Aufgerufen: 31.08.2021.

[Mit97] Tom M. Mitchell. *Machine Learning.* McGraw-Hill Science/Engineering/Math, März 1997.

[MM12] Georgios Mastorakis and Dimitrios Makris. Fall detection system using kinect's infrared sensor. *Journal of Real-Time Image Processing*, 9:635–646, März 2012.

[Ola15] Christopher Olah. Understanding LSTM Networks. https://colah.github.io/posts/2015-08-Understanding-LSTMs/, August 2015. Aufgerufen: 18.08.2021.

[ON15] Keiron O'Shea and Ryan Nash. An introduction to convolutional neuronal networks. https://arxiv.org/pdf/1511.08458.pdf, 2015. Aufgerufen: 11.08.2021.

[Ope] OpenCV. OpenCV. https://opencv.org/. Aufgerufen: 12.08.2021.

[Ope12] OpenNI. OpenNI GitHub. https://github.com/OpenNI/OpenNI, Mai 2012. Aufgerufen: 12.08.2021 - Version 1.5.4.0.

[PVG+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[PyT] PyTorch. PyTorch. https://pytorch.org/. Aufgerufen: 16.08.2021.

[RAR+11] Caroline Rougier, Edouard Auvinet, Jacqueline Rousseau, Max Mignotte, and Jean Meunier. Fall Detection from Depth Map Video Sequences. *9th International Conference on Smart Homes and Health Telematics, ICOST 2011 Montreal, Canada,*, pages 121–128, Juni 2011.

[RMSAR06] Caroline Rougier, Jean Meunier, Alain St-Arnaud, and Jacqueline Rousseau. Monocular 3D Head Tracking to Detect Falls of Elderly People. *2006 International Conference of the IEEE Engineering in Medicine and Biology Society*, pages 6384–6387, 2006.

[RMSAR11] Caroline Rougier, Jean Meunier, Alain St-Arnaud, and Jacqueline Rousseau. Robust Video Surveillance for Fall Detection Based on Human Shape Deformation. *IEEE Transactions on Circuits and Systems for Video Technology*, 21(5):611–622, Mai 2011.

[SDVS20] Kalyanam Supriya, Anemu Divya, Balaga Vinodkumar, and Gedala Ram Sai. Trigger word recognition using lstm. *International Journal of Engineering and Technical Research (IJERT)*, 9(6), Juni 2020.

[SHNLP18] Anahita Shojaei-Hashemi, Panos Nasiopoulos, James J. Little, and M. T. Pourazad. Video-based human fall detection in smart homes using deep learning. *IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5, 2018.

[Wag06] Christoph Wagner. Kantenextraktion - Klassische Verfahren. http://www.mathematik.uni-ulm.de/stochastik/lehre/ws05_06/seminar/wagner.pdf, Januar 2006. Aufgerufen: 04.09.2021.

[Öz20] Ahmet Özlü. Long Short Term Memory (LSTM) Networks in a nutshell. https://ahmetozlu93.medium.com/long-short-term-memory-lstm-networks-in-a-nutshell-363cd470ccac, Juni 2020. Aufgerufen: 22.08.2021.