HAW HAMBURG

BACHELORTHESIS
Daniel von Drathen

Neuronale Klassifikation von Bewegungsabläufen mittels Time-of-Flight Kameras

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Daniel von Drathen

Neuronale Klassifikation von Bewegungsabläufen mittels Time-of-Flight Kameras

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

Betreuender Prüfer: Prof. Dr.-Ing. Andreas Meisel
Zweitgutachter: Prof. Dr. Tim Tiedemann

Eingereicht am: 25. Juli 2022

Daniel von Drathen

Thema der Arbeit

Neuronale Klassifikation von Bewegungsabläufen mittels Time-of-Flight Kameras

Stichworte

Klassifizierung, Bewegungsablauf, Time-of-Flight

Kurzzusammenfassung

Im Laufe dieser Arbeit wird die Realisierbarkeit einer Klassifizierung der Bewegungsabläufebei Langhantelübungen anhand des Beispiels der Kniebeuge erforscht. Hierbei wird untersucht, ob mittels Aufnahmen mit einer Time-of-Flight Kamera, ein neuronales Netz trainiert werden kann, welches unterschiedliche Fehler in der Ausführung erkennen und unterscheiden kann. Es soll ein Vergleich aufgestellt werden, welcher zeigt, welche Netztypen besser geeignet sind diese Aufgabe durchzuführen.

Daniel von Drathen

Title of Thesis

Neuronal classification of movement patterns using Time-of-Flight cameras

Keywords

Classification, movement pattern, Time-of-Flight

Abstract

During the course of this work, the feasibility of classifications of movement sequences in barge exercises is investigated using the example of squats. It is investigated whether a neural network can be trained using a Time-of-Flight camera. The resulting neural network is supposed to recognize and distinguish different errors in the execution. A comparison between different network types is made showing which type is better suited for this task.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Die folgenden Kapitel beschreiben die Motivation, sowie die Problemstellung und Ziele die sich hieraus ergeben haben.

1.1 Motivation

Die Motivation dieser Arbeit stellt die Problemstellung der korrekten und fehlerfreien Durchführung von Bewegungsabläufen im dem Bereich der Langhantel dar. Unter diese Übungen fallen unterschiedliche Arten der Kniebeugen, des Kreuzhebens, sowie Military Press's. Diese Übungen erfordern Wissen und in einigen Fällen einen Trainer, der Fehler in der Haltung erkennen und korrigieren kann. Der Aufwand und die Kosten für einen ausgebildeten Fitnesstrainer können dabei erheblich sein. Zudem kann nicht immer ein Trainer vor Ort sein, um bei jeder Übung begleitend tätig zu sein. Der Einsatz von einem Neuronalen Netzwerk in Verbindung mit einem Videosystem kann hier für Abhilfe schaffen. Aus diesem Grund wurde ein kamerabasiertes System überlegt.
Eine Kamera soll hierbei die Ausführung der Übungen aufnehmen. Diese Aufnahme soll von einem trainierten neuronalen Netzwerk klassifiziert und auf diverse Fehler überprüft werden.

1.2 Problemstellung und Ziele

Für den Einsatzzweck eines Neuronalen Netzwerkes im Bereich der Klassifizierung von Bewegungsabläufen, muss man zunächst über die Art des zu verwendenden Netzwerkes nachdenken. Im Vergleich zu normalen Klassifizierungen im Bereich Computer Visions liegt der Hauptunterschied in der Datenmenge, die für jeden einzelnen Datenpunkt benötig wird, sowie die Notwendigkeit mehrere Bilder in Folge als ein zusammenhängendes Ganzes zu betrachten. Bei einer statischen Menge an Bildern pro Datenpunkt wäre dies

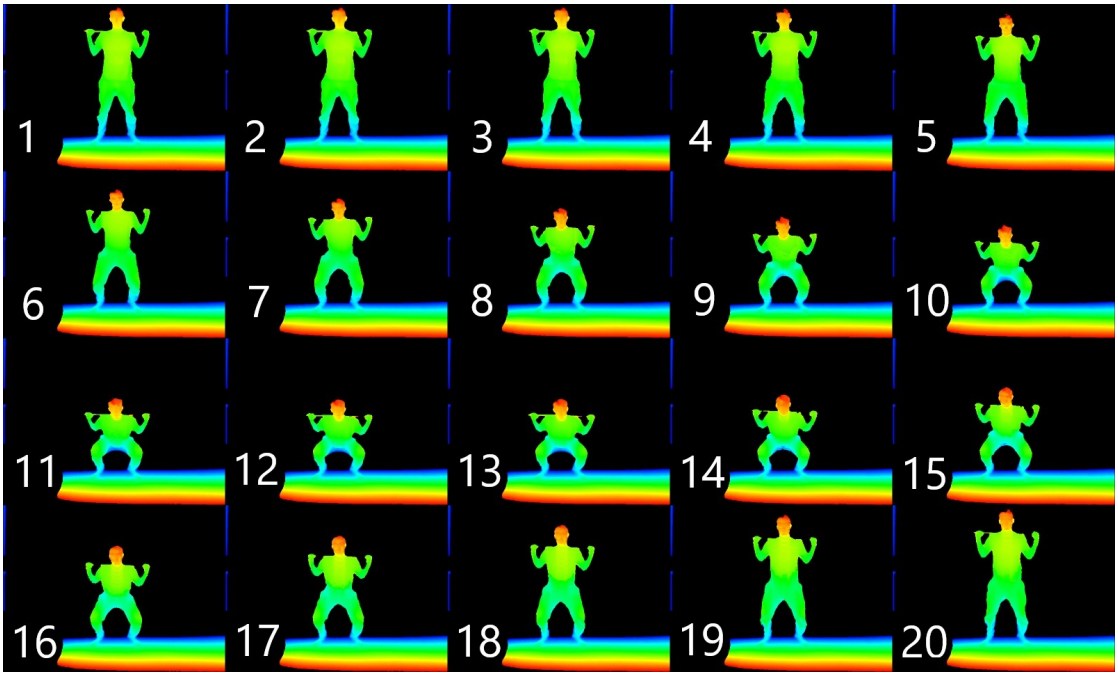


Abbildung 1.1: Bildverlauf einer guten Durchführung der Kniebeuge.

zunächst allerdings zu vernachlässigen, wodurch der Einsatz eines konventionellen CNN möglich ist. Um diesen Ansatz gegen einen dynamischen Versuch vergleichbar zu machen, wird die Untersuchung mit einer festen Anzahl an Bildern durchgeführt. Neben einem CNN auf Basis der Keras Bibliothek wird zudem ein Hybrid Netzwerk von einem CNN und einem LSTM, sowie eine Transformer Architektur verwendet.
Die wichtigste Herausforderung stellt die Ähnlichkeit der verschieden fehlerhaften Durchführungen zu einer guten Form (1.1) dar.
Diese können sich teilweise nur geringfügig von einer korrekten Ausführung unterscheiden (siehe 1.2). Der betreffende Fehler kann dabei zu jeder Zeit einer Kniebeuge entstehen (siehe 1.3). Ebenfalls möglich sind zeitliche Fehler bei Timing und Reihenfolge der Bewegungen. Dies kann zu schnellen oder unsauberen Ausführungen und gegebenenfalls Verletzungen führen. Natürlich kann es bei einer Kniebeuge auch zu mehreren Ungenauigkeiten kommen und somit Kombination von Fehlern
Die verschiedenen Fehler können mit bloßem Auge nur schwer zu erkennen sein, was es für ein neuronales Netzwerk besonders schwierig macht.

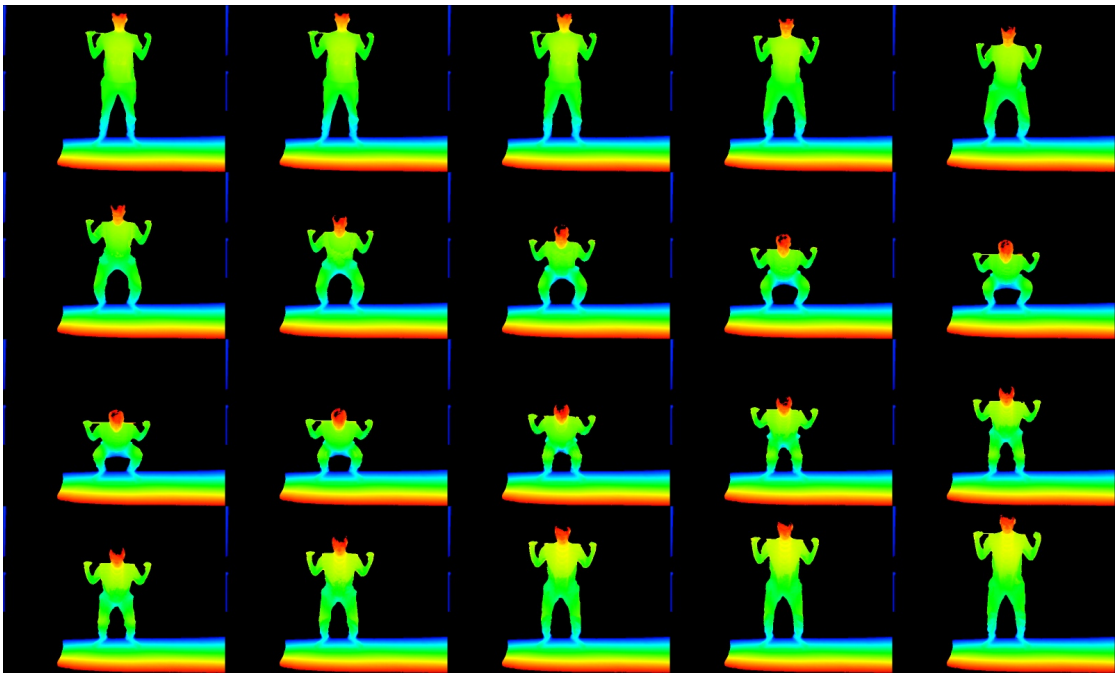


Abbildung 1.2: Bildverlauf einer falschen Durchführung mit zusammengedrückten Knien am untersten Punkt.

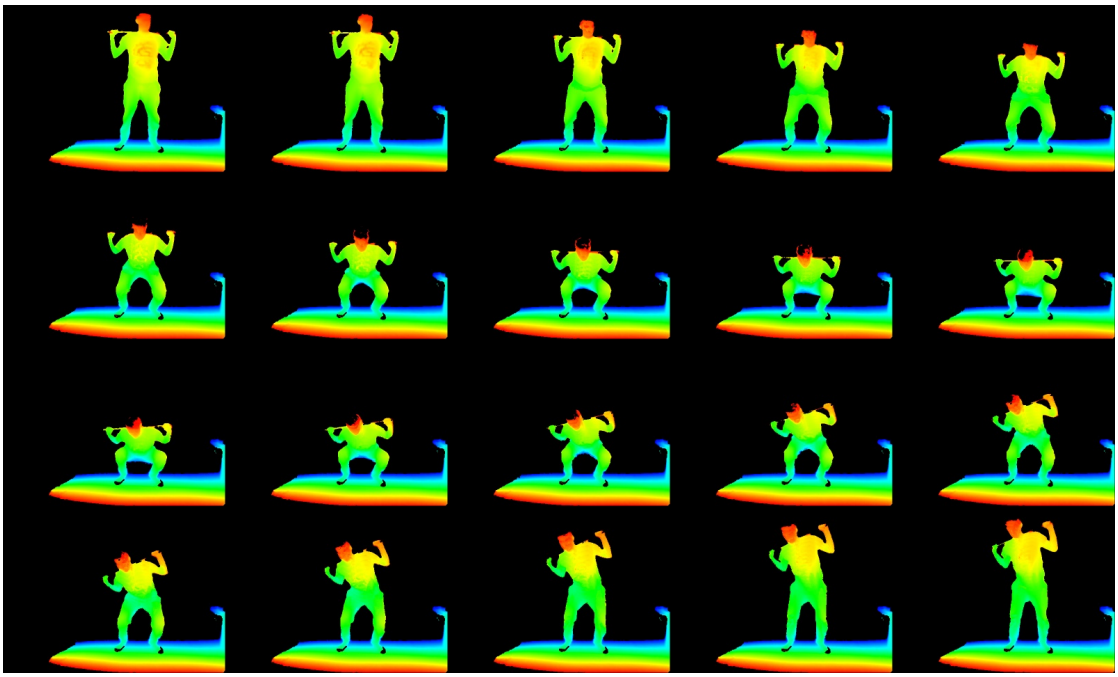


Abbildung 1.3: Bildverlauf einer falschen Durchführung beim Aufstehen aus der Hocke.

2 Grundlagen der eingesetzten Verfahren

2.1 Vortrainierte Netzwerke

Das normale Verfahren zum trainieren eines neuronalen Netzwerkes für den Zweck der Klassifikation eines Datensatzes verläuft durch das Initialisieren der Gewichtungen. Dies geschieht zunächst mit Zufallswerten. Sobald das Training startet werden diese Gewichte verändert, um die Aufgabe mit weniger Fehlern durchzuführen. Sobald das Resultat des Trainings ausreichend ist, werden die Gewichte gespeichert und das Training ist beendet.

Diese gespeicherten Gewichtungen können nun verwendet werden, um bei dem Training mit einem anderen Datensatz die Initialgewichtungen zu bestimmen. Dabei gilt das zuerst trainierte Netz als vortrainiertes Netz. Das zweite Netz wird nun genutzt um die Gewichtungen auf den neuen Datensatz feinzujustieren.

Diese Herangehensweise ist deshalb hilfreich, da die ansonsten zufällig gewählten Gewichtungen zu Beginn des Trainingsvorganges nichts mit der eigentlichen Aufgabe des Netzes zu tun haben. Durch die Nutzung eines vortrainierten Netzwerkes, können die Gewichtungen näher an den optimalen Werten begonnen werden.

Der Datensatz des vortrainierten Netzes kann identisch mit dem Datensatz des zweiten Trainings sein. Es kann sich allerdings auch von diesem unterscheiden. Dabei steht der Erfolg des vortrainierens Netzwerkes in Korrelation mit der Ähnlichkeit der beiden Datensätze, ist aber nicht unbedingt linear zu sehen. Dabei macht es wenig Sinn ein vortrainiertes Netzwerk mit Finanzdaten zu trainieren, um dieses daraufhin für Bildklassifikation zu nutzen.

2.2 Netztypen

Im Bereich der Klassifikation von Datensätzen gibt es verschiedene Netztypen die in Betracht gezogen werden können. Im folgenden Abschnitt werden die möglichen Typen näher erläutert.

2.2.1 CNN

Das menschliche Gehirn verarbeitet Daten auf bildlicher Weise. Dabei werden Muster und Merkmale identifiziert oder klassifiziert die sich in unserem Umfeld befinden. Mit neuronalen Netzwerken versuchen wir diesen Ablauf in maschineller Art nachzubilden.

Convolutional Neural Networks (im folgenden CNN genannt) ist ein neuronales Netz welches versucht diese Merkmale aus einem Datensatz zu extrahieren und zu identifizieren. Dies formt eine der fundamentalen Operationen in der Welt des maschinellen Lernens und findet in einem Großteil der neuronalen Netzwerke im Bereich der Objekterkennung und Bildklassifikation Verwendung. Beispiele hierfür sind GoogleNet und VCC19.

Die Basiskomponenten eines CNN setzen sich zusammen aus:

Die Basiskomponenten eines CNN	
Convolution	Merkmale in Bildern erkennen.
ReLU	Das Bild glätten und Grenzen klarer darstellen.
Pooling	Verzerrung minimieren.
Flattening	Das Bild in eine verarbeitbare Form bringen.
Full connection	Die Daten im neuronalen Netzwerk verarbeiten.

2.2.2 Recurrent Neural Network

Recurrent Neural Networks (im folgenden als RNN abgekürzt), arbeiten unter dem Prinzip den Ausgang eines bestimmten Layer's zu speichern und in den Eingang zurückzuführen. Dies ermöglicht die Vorhersage des Ausganges des layer's.
Abbildung 2.1 zeigt wie aus einem einfachen Feed-foward Netzwerk ein RNN wird. Die Knoten der verschiedenen Layer des Neuronalen Netzwerkes werden zusammengefügt und formen ein einzigen Layer in einem RNN. A,B, und C sind die Parameter des Netzwerks.

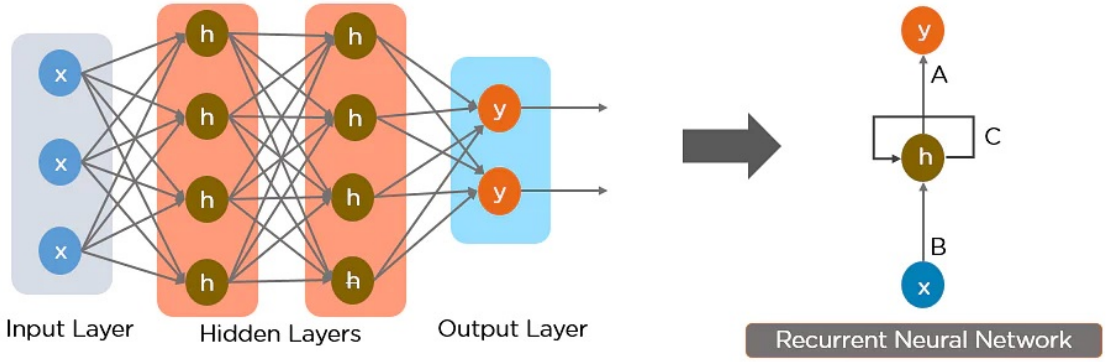


Abbildung 2.1: Darstellung eines einfachen RNN-Netzwerks, im Vergleich zu einem Feed-Foward Neural Network. Von https://www.simplilearn.com/tutorials/deep-learning-tutorial/rnn

Abbildung 2.2 zeigt ein vollständig Verbundenes RNN Netz. x ist das Input-Layer, h das Hidden-Layer und y ist das Ouput-Layer. A, B und C sind die Netzwerkparameter die verwendet werden um den Output des Modells zu verbessern. Zu jedem gegebenen Zeitpunkt t, ist der Input eine Kombination des Inputs *x(t-1)* und *x(t-2)*. Der Output zu jedem Zeitpunkt wird zurückgeschickt um den Output weiter zu verbessern.

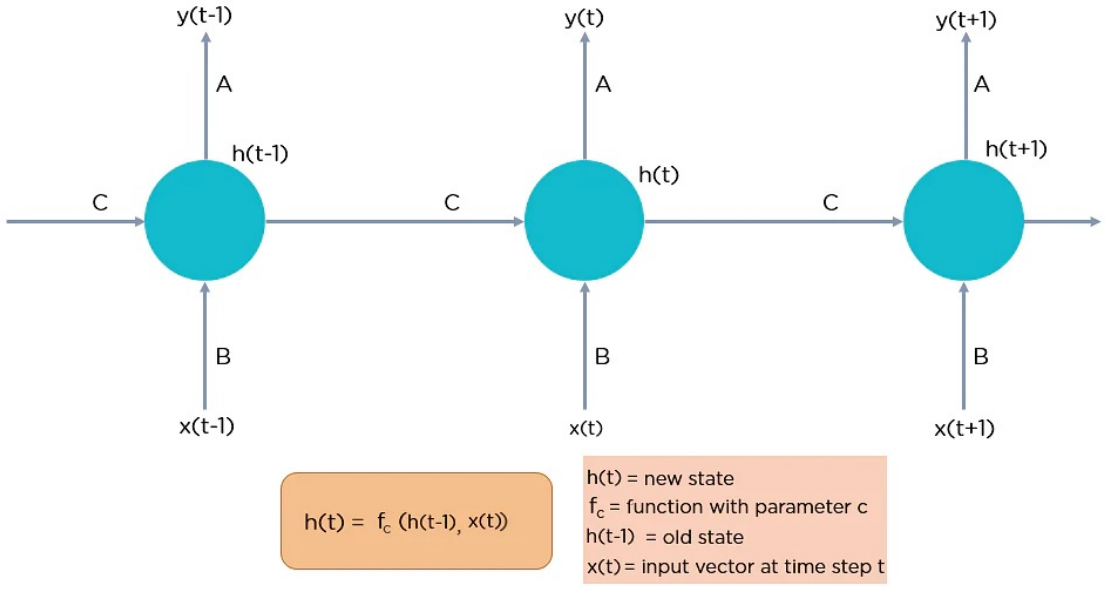
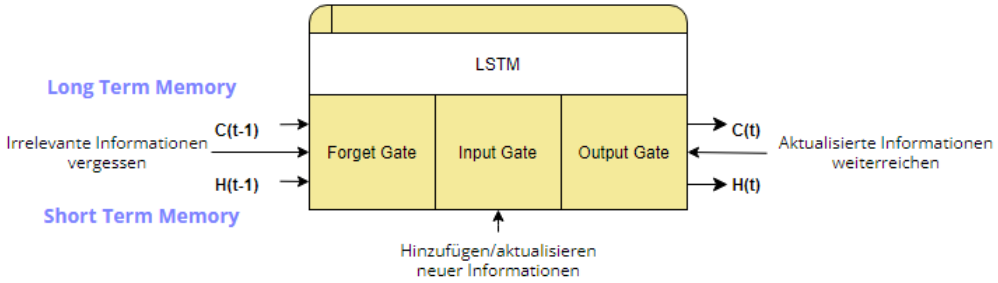


Abbildung 2.2: Darstellung eines vollständig verbundenen RNN-Netzwerks. Abbildung von https://www.simplilearn.com/tutorials/deep-learning-tutorial/rnn

2.2.3 Long short-term memory

Ein Long short-term memory (im folgenden als LSTM abgekürzt) Netzwerk stellt eine Erweiterung zu Recurrent Neural Networks (RNN) dar. Dabei besteht die Struktur aus drei Teilen. Der erste Teil entscheidet darüber, ob die Information aus dem vorherigen Zeitabschnitt wichtig ist oder vergessen werden kann. Der zweite Abschnitt versucht Informationen aus der aktuellen Eingabe zu entnehmen. Der letzte Schritt leitet die aktualisierten Informationen an den nächsten Zeitabschnitt weiter. Diese drei Abschnitte werden auch Gatter(englisch=Gates) genannt. Das erste Gate wird als Forget Gate bezeichnet, das zweite als Input Gate und das dritte als Output Gate.

LSTM's besitzen versteckte States H. H(t-1) repräsentiert den versteckten State des vorherigen Zeitabschnitts und H(t) den State des aktuellen Abschnitts. Dies wird anders als bei herkömmlichen RNN's jedoch durch einen Cell State C(t-1) und C(t) komplementiert. Visualisiert sieht das nun wie folgt aus.



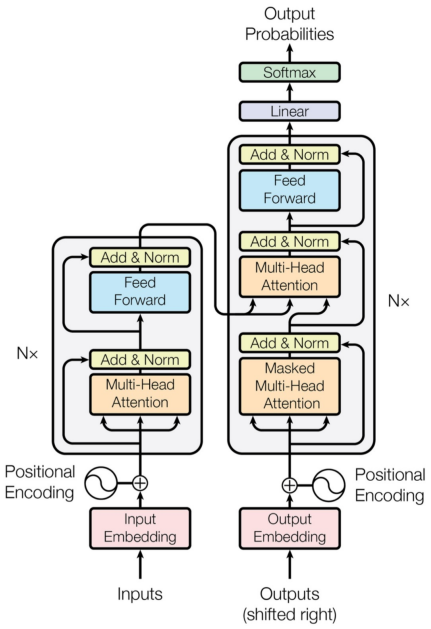
(a) LSTM Visualisierung.

2.2.4 Transformer

Die Wissenschaftliche Ausarbeitung „Attention Is all You Need [4]“ führt die neue Transformer Architektur ein. Wie LSTM's, können Transformer verwendet werden, um Sequenzielle Daten zu lernen, allerdings ohne dabei auf RNN's zurückzugreifen.
Transformer Modelle finden dabei besonders bei Übersetzungen von Texten ihren Einsatz. Abbildung **??** zeigt ein Transformer-Modell mit seinen drei wichtigen Komponenten:

1. Der Encoder, welcher die Eingangssequenz in State-Representationsektoren Encodiert.
2. Ein Attention Mechanismus, das dem Transformer erlaubt, den richtigen Aspekt der Sequenz zu Fokusieren. Dies findet Anwendungen in Encoder, sowie Decoder.
3. Ein Decoder, welcher den State-Representationsvektor dekodiert um ein Ziel Ausgang zu generieren.

Der Encoder auf der linken Seite der Transformer-Architektur, bildet die Eingangssequenz als eine Kontinuierliche Representation ab, welche der Decoder verarbeiten muss. Der Decoder auf erhält den Ausgang des Encoder's zusammen mit dem Ausgang des Decoder's einen Zeitschritt vorher, um eine Sequenz zu erzeugen.



(a) Aus 'Attention Is All You Need' von Vaswani et al. [4]

2.2.5 Grundlegende Einschränkungen

Die verwendeten Netztypen beschränken die Anzahl der Bilder pro Durchführung auf eine statische Menge. Hierfür gibt zwei mögliche Herangehensweisen. Man könnte so entweder auf auf eine feste Menge an Bildern pro Übung Padden oder bereits bei den Aufnahmen auf eine feste Bildzahl achten. Das Aufstocken auf eine bestimmte Bildzahl ist dabei der weniger effektive Weg und somit wird die gezielte Aufnahme mit einer gleichbleibenden Länge vorgezogen. Es hat sich während der Aufnahmen dabei eine durchschnittliche Bildanzahl von 23 Bildern ergeben. Bei nachfolgenden Ausführungen wurde daraufhin ein striktes Timing eingeführt, welches zu dieser festen Bildanzahl führt. Dabei ist die strikte Einhaltung von 23 Bildern pro Durchführung lediglich als erster Ansatz einer besseren Vergleichbarkeit zu verstehen und muss in weiteren Ausführungen in dem Bereich der Klassifizierung von diesen Bewegungsabläufen weiter untersucht und durch den Einsatz einer Dynamischen Architektur abgelöst werden. Aufgrund der eingeschränkten Datensatzmenge und fehlender Diversifikation der Probanden, ist zudem die Allgemeingültigkeit dieses Versuchsaufbaus lediglich als Indikation einer Machbarkeit einzuschätzen. Alle Aufnahmen für den Datensatz wurden mit einer einzigen Person durchgeführt, um die Datenerfassung zu erleichtern. Die vollständige Erstellung eines Datensatzes ist in dem Umfang dieser Arbeit nicht erreichbar.

3 Design

Das folgende Kapitel zeigt die Systemarchitektur und dessen Komponenten auf.

3.1 Hardware

Das System welches zum Einsatz kommt, setzt sich aus Hardware- und Softwarekomponenten zusammen. Der nachfolgende Abschnitt wird zunächst die Hardwarekomponente erläutern.

3.1.1 Basler Blaze 101

„Die Time-of-Flight-Kamera blaze bietet präzise 3D-Bilder in Echtzeit dank der DepthSense™-Sensortechnologie von Sony und der integrierten Tiefenbildverarbeitung.

Die industrielle 3D-Kamera Basler blaze arbeitet nach dem Time-of-Flight-Prinzip. Sie ist ausgestattet mit Laserdioden, die im NIR-Bereich (940 nm) arbeiten, und generiert mit einem Multipart-Bild 2D- und 3D-Daten in einer Aufnahme, bestehend aus Entfernungs-, Intensitäts- und Konfidenzkarten.

Die Basler blaze bietet VGA-Auflösung, hohe Präzision und Genauigkeit sowie leistungsstarke Funktionen zu einem attraktiven Preis. Damit ist die Basler blaze ideal für eine Vielzahl von 3D-Vision-Anwendungen in der Logistik und Fabrikautomation.“

“Der leistungsfähige Sony Depth-Sense™ IMX556-Sensor in der Basler blaze erlaubt eine präzise optische Messung (+/- 5 mm) von Objekten mit bis zu 30 Bildern pro Sekunde. Dank seiner Backside-Illuminated-(BSI-)Architektur kann dieser moderne CMOS-Sensor einfallendes Licht besser aufnehmen. In Kombination mit der Sony-eigenen ToF-

Pixeltechnologie CAPD (Current Assisted Photonics Demodulation) lässt sich eine deutlich höhere Genauigkeit der 3D-Tiefendaten im Vergleich zu herkömmlichen ToF-Kameras erreichen. Die eingebauten leistungsstarken VCSEL-Dioden sowie die ToF-optimierte Optik sind auf den IMX556-Sensor abgestimmt und unterstützen die präzise optische Messung mit der Lichtlaufzeitmethode (Time-of-Flight).
Die Basler blaze bietet ein großes Sichtfeld (FOV 67° x 51°) sowie einen flexiblen Arbeitsabstand von 0,3 m bis maximal 10 m. So kann die Kamera die Tiefendaten auch großer Objekte und ganzer Szenen auf einmal erfassen und optimal bei Pick-und-Place-Applikationen, Volumenmessungen und Palettieraufgaben im Logistik- und Produktionsumfeld unterstützen.
Mit bis zu 30 Bildern pro Sekunde eignet sich die Basler blaze hervorragend für Anwendungen, bei denen die Schnelligkeit des bildgebenden Systems entscheidend ist. Pakete auf einem Förderband können „on-the-fly“ detektiert und ihr Volumen vermessen werden, um die Frachtkosten zu bestimmen. Bei Roboter-Greifaufgaben erkennt die blaze Lage, Größe und Greifposition der Objekte in Echtzeit. Die Verarbeitung der Tiefendaten erfolgt dabei bereits in der Kamera, um die Rechen- und Datenlast zu reduzieren, wichtig z.B. bei AGV-Anwendungen.
Für die Entfernungsmessung wird nur das kameraeigene Licht benötigt. Deshalb hat die blaze Kamera einen optischen Bandpassfilter, der genau auf die Wellenlänge ihrer Lichtquelle abgestimmt ist. Die blaze arbeitet mit Lichtpulsen von 940 nm, die für den Menschen unsichtbar sind. Da das Sonnenlicht bei 940 nm größtenteils von der Atmosphäre absorbiert wird und somit den Kamerabetrieb kaum stört, ist die blaze eine tageslichtrobuste Kamera. Dies ist besonders bei Anwendungen im Außenbereich von Vorteil, wie bei Container-Handling- oder Depalettieraufgaben, und überall dort, wo sich künstliches Licht mit Sonnenlicht mischt.
Die Basler blaze bietet Kamera-Features für den störungsfreien synchronen Betrieb mehrerer ToF-Kameras in derselben Anwendung, selbst wenn sich diese in unterschiedlichen Netzwerken befinden. Der Einsatz mehrerer Kameras kann für die Erfassung großer Objekte oder Szenen sowie für sich bewegende Objekte, wie AGVs in einem Lager, erforderlich sein.
“[**Basler**]

3.1.2 Trainingsumgebung für die neuronalen Netze

Die neuronalen Netze werden auf einer Jupyter-Notebooks Umgebung trainiert. Der Jupyter-Notebook hat Zugriff auf eine Workstation Grafikkarte die sich im Rechenzentrum der HAW befindet. Dabei handelt es sich um die NVIDIA V100s mit 5120 CUDA-Kernen und 32GB VRAM [**v100**]
Ebenfalls zum Einsatz gekommen ist ein Rechner mit verbauter NVIDIA GTX 1080 für lokale Berechnungen. Dies ist besonders am Beginn der Durchführungen geschehen, um kurze Testdurchläufe mit weniger als 30 Perioden zu testen.

3.2 Software

In dem folgenden Abschnitt wird die Funktionalität der einzelnen Softwarekomponenten erläutert und die Gründe für deren Auswahl genannt. Es wird außerdem über die nötigen Schritte für der Aufnahmen diskutiert.

3.2.1 Trainingsskript

Das Trainigsskript welches als Grundlage für das Trainieren der Netze dient, stammt aus dem von den Keras-Examples auf der Seite [**https://keras.io/examples**]. Ein selber geschriebenes Script erweitert die Funktionalität für den angelegten Datensatz. Zudem wurden folgende weitere Modifikationen vorgenommen. Die Möglichkeit eine Heatmap der erkannten Abschnitte in den Bildern zu visualisieren, sowie LSTM Komponenten anstatt der RNN-Layer zu verwenden.

3.2.2 Benötigter Datensatz

Der benötigte Datensatz mit ToF Aufnahmen von gewünschten Bewegungsabläufen muss speziell angefertigt werden. Aufgrund der Einmaligkeit dieser Aufgabe wurde der Datensatz mithilfe der Basler Blaze 101 manuell angefertigt. Dabei haben sich Schwierigkeiten in der Bildwiderholungsrate, sowie in Genauigkeit ergeben. Während Tests mit den Aufnahmen hat sich zudem gezeigt, dass der Winkel der Aufnahme ebenfalls von Bedeutung ist, da bei gewissen Positionen wichtige Körperteile verdeckt werden (siehe abb. **??**a)
Die Verwendung von Position Estimation mittels OpenCV wird ebenfalls in Betracht

gezogen. Dies führt zu einer klareren Abbildung für die Klassifizierung (siehe 3.2) zeigt aber auch deutlicher die Wichtigkeit richtiger Positionierung. Des Weiteren wurde beobachtet, das dies zu einer zusätzlichen Fehlerquelle bei den Aufnahmen führen kann, wodurch dieser Ansatz wieder eingestellt wurde.

(a) Diagonale Ausrichtung zur Kamera. (b) Frontale Ausrichtung zur Kamera.

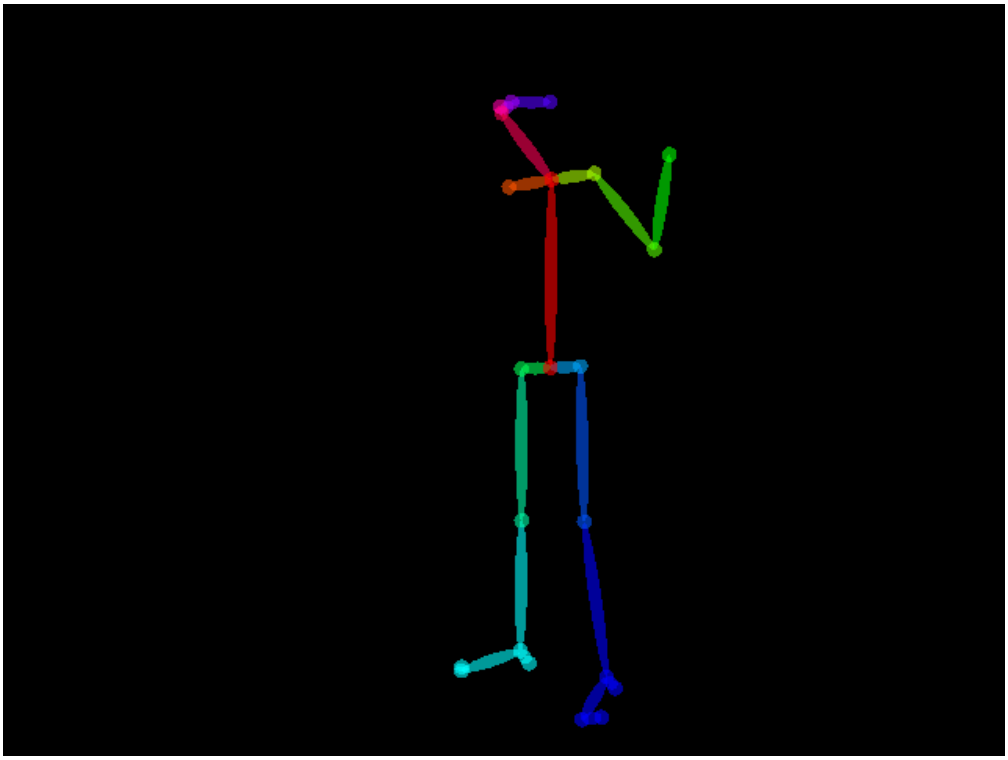


Abbildung 3.2: Darstellung mittels OpenCV Pose Estimation.

Auf der Abbildung kann man erkennen, dass der hintere Arm von dem Körper verdeckt und so nicht mehr von OpenCV einbezogen wird.
Eine weitere Einstellung der Aufnahmen wurde ebenfalls unternommen, um Nutzen aus der ToF Fähigkeit des Kamerasensors zu machen. So wurde der Output der Bilddateien auf RGB gestellt und die Auflösung auf 640 x 480 gesetzt (siehe 3.3). Dabei wird die Entfernung zum Sensor mittels RGB Werten widergespiegelt. Zudem wurde die maximale Distanz auf einen Korridor gelegt der das Ziel der Aufnahme umfasst, allerdings gleichzeitig den Hintergrund ausblendet, um dem neuronalen Netzwerk weniger Datenpunkte zu liefern.

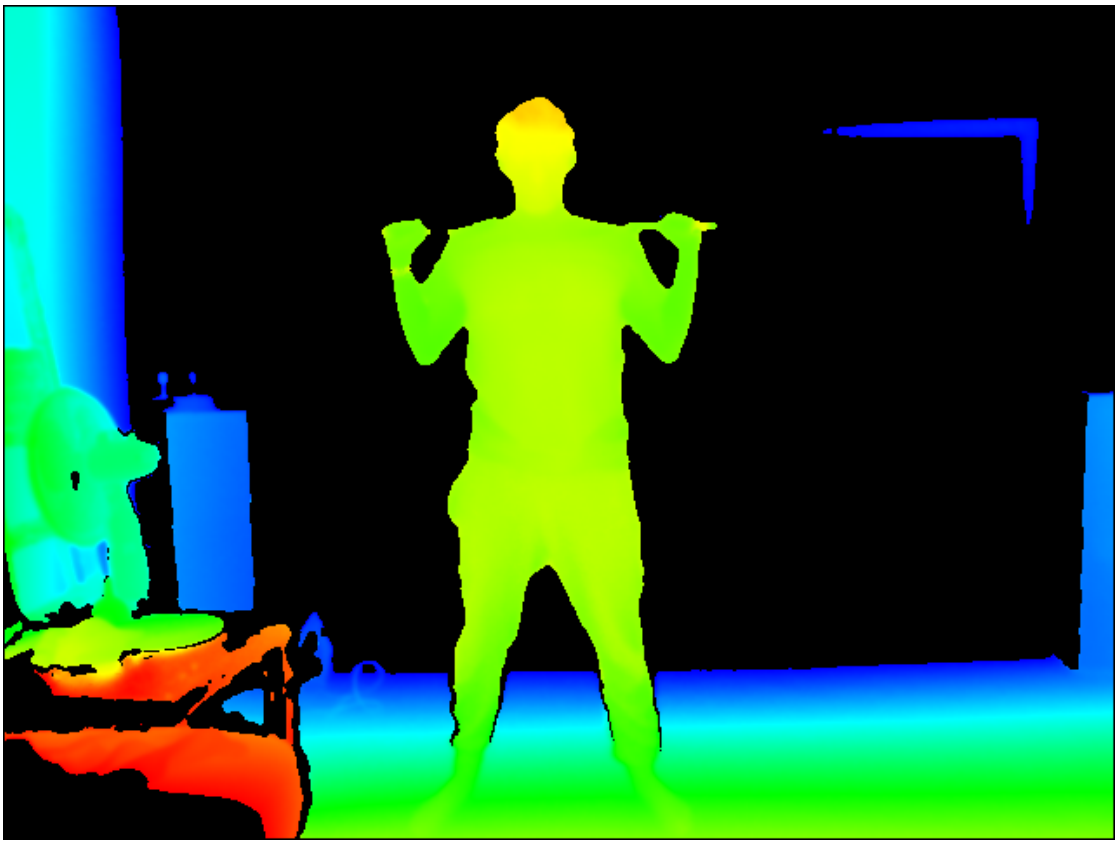


Abbildung 3.3: RGB Aufnahme aus der frontalen Ansicht.

4 Implementation

Im folgenden Kapitel werden die einzelnen Besonderheiten, Schritte und die Vorbereitung für die Implementierung dieser Arbeit beschrieben. Unter anderem das Trainieren der neuronalen Netze, die Inbetriebnahme des Basler Blaze 101, die visuelle Ausgabe der neuronalen Netze im Betrieb und mit welcher Metrik die Netze verglichen werden.

4.1 Hardware Einrichtung der Basler Blaze 101

Die ordnungsgemäße Inbetriebnahme der Basler Blaze 101 wird ausführlich über die Dokumentation online beschrieben. Zunächst sollte die Kamera in der vorgesehenen Position am Aufnahmeort fixiert werden. Daraufhin verbindet man den M12 8-Pin Stecker des mitgelieferten Gigabyte Ethernet Kabels mit dem dafür vorgesehenen Anschluss, sowie das RJ45 Ende mit einem verfügbaren Port des verwendeten Computers. Nun kann der M12 8-Pin Anschluss des Stromkabels mit der Kamera und abschließend mit einer Steckdose verbunden werden. Dabei kam es bei der späteren Übertragen der Daten mittels der Basler Software (siehe Sektion 4.2) aufgrund der Ethernetschnittstelle im verwendeten Computer zu Verbindungsschwierigkeiten. Über die Dokumentation konnte ich dabei feststellen, das für eine bessere und flüssigere Verbindung eine Netzwerkkarte der Serie Intel PRO 1000, I210, I340, und I350 verwendet werden sollte. Ein entsprechendes Modell wurde somit nachgekauft und in Betrieb genommen, woraufhin die Verbindungsschwierigkeiten reduziert wurden.

4.2 Software Einrichtung der Basler Blaze 101

Für die Inbetriebnahme der Kamera wird Baslers eigene Software `Pylon Camera Suite` verwendet. Für die bestmögliche Verbindung zur Kamera gibt Basler zudem mehrere

Punkte zur Netzwerkkonfiguration bereit. Die IP der Kamera wird hierbei über den `Pylon IP Configurator` entweder Manuell, oder Automatisch vergeben. Der Status der Verbindung kann ebenfalls über den Configurator ermittelt werden. Die folgenden Einstellungen wurden in der `Pylon Camera Suite` vorgenommen um die Aufnahmen Konsistent und bei einer Framerate von 20 Bildern pro Sekunde zu halten.

AcquisitionFrameRateEnable Das aktivieren des AcquisitionFrameRateEnable ermöglicht die Begrenzung der Framerate für den nächsten Parameter.

AcquisitionFrameRate Dieser Parameter bestimmt die Framerate und wurde auf 20 gestellt um eine gleichbleibende Aufnahme zu garantieren.

AmbiguityFilter Hierdurch können unsichere Entfernungsmessungen gefiltert werden, die durch Staubpartikel oder ungünstige Lichteinstrahlung verursacht werden können.

AmbiguityFilterThreshold Hierdurch kann die Filterung der Ungenauigkeiten geregelt werden. Die Stärke der Filterung wurde für den Aufnahmeort abgestimmt und ermöglicht eine Erkennung falscher Abstandsmessungen ohne gute Werte zu entfernen.

OperatingMode Der Modus der Aufnahmen kann zwischen `Short Range` und `Long Range` gewechselt werden

- `Short Range` Begrenzt die Reichweite der Aufnahmen auf 0.3 - 1,5 m und verbessert Temporal Noise auf $<$2 mm
- `Long Range` Stellt die Reichweite der Aufnahmen auf 0,3 - 10 m aber die Temporal Noise Grenze auf $<$5 mm

DepthMin DepthMax Die Einstellung dieser Parameter begrenzt die Entfernung mit der bei der Aufnahme gearbeitet wird. Um eine möglichst leichte Verarbeitung der Daten zu erlauben wurde der Aufnahmebereich auf einen Korridor minimiert, in dem lediglich das Ziel der Übung zu erkennen ist. Die Parameter wurden dazu zwischen 1900 und 3500 gelegt. Dies führt zu Aufnahmen im Bereich von 1,9 m und 3,5 m. Dadurch wird jedweder Hintergrund aus den Aufnahmen entfernt.

4.2.1 Optimierungen der Netzwerkeinstellungen

Verschiedene Optimierungen der Netzwerkeinstellungen wurden mithilfe der Dokumentation eingerichtet.

1. Zunächst werden in den Windows Netzwerkadaptereinstellungen jegliche Einstellung gelöscht und der pylon GigE Vision Driver und das Internet Protocol Version 4 werden aktiviert.
2. In den erweiterten Einstellungen des Netzwerktreibers werden die `Receive Descriptors` auf den maximalen Wert gesetzt.
3. Die CPU Interrupts mit der Beschreibung `Interrupt Moderation Rate` werden auf 1000 beschränkt.
4. Der Speed und Duplex Modus wird auf `Auto Negotiation` gestellt.

4.3 Datensatzerzeugung

Über die `Pylon Camera Suite` können die Komponenten, die durch die Kamera gesendet werden, ausgewählt werden.

- **Range** Die `Range` Komponente stellt die Bilder als entweder Depth Map oder Point Cloud dar. Die Depth Map stellt die Distanz als Grau- bzw. RGB-Werte dar, während die Point Cloud jeden Pixel als x,y,z Koordinate speichert.
- **Intensity** Die `Intensity` Komponente fügt jedem Daten Frame ein Intensity Bild hinzu. Dieses zeigt die Helligkeit der reflektierten Lichtimpulse als 16-Bit Integerwert pro Pixel. Dies spiegelt die nahen Infrarotanteile des Lichtes wieder, da die Hintergrundlichter rausgefiltert werden.
- **Confidence** Die `Confidence` Komponente fügt jedem Datensatz eine Confidence Map hinzu. Diese stellt eine visuelle Repräsentation der Zuverlässigkeit der Depth Map dar. Die Menge an Licht pro Pixel korreliert dabei mit dem Konfidenzwert.

Jede aktivierte Komponente erzeugt bei der Übertragung der Bilddaten zusätzlichen Netzwerkaufwand. Für eine gleichmäßige Aufnahme von 20 fps können mit der genutzten Aufnahmehardware lediglich zwei Komponenten gleichzeitig aktiv sein. Bei der Auswahl der richtigen Komponente muss die Weiterverarbeitung berücksichtigt werden. So stellt

die Nutzung der Point-Cloud Dateien für die Verarbeitung und die zwischenzeitliche Einsicht der Bilder ein Problem dar. Die verwendeten Speichertypen sind schwer einzusehen und benötigen extra angefertigte Funktionen, um in ein neuronales Netz eingefügt zu werden. Leichter erweist sich die Nutzung der Depth Map mit RGB-Werten. Dies erlaubt zum einen die erzeugten Bilder ohne spezielle Funktionen zu überprüfen, was die Handhabung während der Testphasen erleichtert, als auch die Bilder in ein neuronales Netzwerk einzulesen und in anderen Bibliotheken zu bearbeiten.
Der Nachteil bei der Nutzung der Depth Map im Vergleich zur Point-Cloud stellt der höhere delay bei der Übertragung dar. Die Nutzung der Point-Cloud Daten kann somit in einer höheren Framrate erfolgen, als die Aufnahme mit Depth Map Daten.
Dennoch wird in diesem Versuch die Nutzung der Depth Map vorgezogen, da dies die Komplexität verringert.

4.3.1 Ordnerstruktur der Trainingsdaten

Für die Verwendung der Rohdaten als Trainingssatz, müssen die ungeordneten und ungefilterten Dateien in eine geordnete Struktur gebracht werden. Zunächst wurde jede Durchführung manuell separiert und in 23 Frames unterteilt. Eine Durchführung geht dabei von einem aufrechten Stand, über zu einer Kniebeuge und wieder zurück zu einem statischen, aufrechten Stand. Diese 23 Bilddaten wurden dann in eine Ordnerstruktur mit aufsteigender Nummerierung eingepflegt und zwischen den jeweiligen Arten aufgeteilt. Anschließend werden die Daten auf die Jupyter-Umgebung hochgeladen *dataset.ipynb* [2].

4.3.2 Data Augmentation

Die Menge und die Varianz der Daten ist einer der wichtigsten Punkte im Machine Learning. Neuronale Netze benötigen sehr große Datensätze, welche häufig durch bereits vorhandene Bibliotheken gegeben sind. Nur dann können genaue Resultate erwartet werden.
Der Einsatz eigener Aufnahmen schränkt die Menge der Daten für diese Arbeit ein. Lediglich eine Person wurde für die Aufnahmen in einem Raum aufgenommen. Aufgrund der Natur der Aufnahmen können zudem nur eine begrenzte Anzahl an Durchführungen am Stück und am Tag durchgeführt werden, bis Erschöpfung bei der Testperson eintritt. Hierdurch ergibt sich eine eingeschränkte Menge an Daten für den Datensatz. An dieser

Stelle kommt `Data Augmentation` zum Einsatz.
`Data Augmentation` ist eine Technik zur künstlichen Erweiterung des Datensatzes. Hierbei werden dem Datensatz zusätzliche, leicht modifizierte Daten hinzugefügt. Dabei kann jede einzelne Einheit mehrmals unterschiedlich modifiziert und dem Datensatz hinzugefügt werden.
Data Augmentation ist ein integraler Bestandteil von Machine Learning. Es gibt somit bereits mehrere Bibliotheken, welche man hierfür verwenden kann. In dieser Arbeit wurde auf die Open Source Bibliothek Albumentation zurückgegriffen [1].
Für den Einsatz von Augmentationen in Verbindung mit Video-Dateien ist es wichtig, dass jeder Frame in einem zusammenhängenden Durchlauf, mit denselben Modifizierungen versehen wird. So muss darauf geachtet werden, dass es die Möglichkeit gibt, jeden Frame mit dem gleichen Parametern zu behandeln. Es ist zudem imperativ sicherzustellen, welche Operationen für diesen Anwendungsfall möglich sind. So ist das Spiegeln der Bild-Dateien um die Y-Achse ohne weiteres möglich, um die X-Achse jedoch nicht, da ein Mensch die Übungen nicht auf dem Kopf durchführen kann. So wurde eine Liste mit möglichen Parametern erstellt

HorizontalFlip Spiegelt das Bild Horizontal um die Y-Achse

HueSaturationValue Ändert zufällig Farbton, Sättigung und Wert des Eingangsbildes, abhängig von den Argumentparametern die eingegeben werden:

- hue_shift_limit=((int, int) oder float) Limitierung der Veränderung des Farbtons. Parameter : (-20, 20).
- sat_shift_limit=((int, int) oder int) Keine Änderung der Sättigung.
- val_shift_limit=((int, int) oder int) Keine Veränderung der Werte.

ShiftScaleRotate Verschiebt und Rotiert den Eingang in Abhängigkeit zu den Parametern:

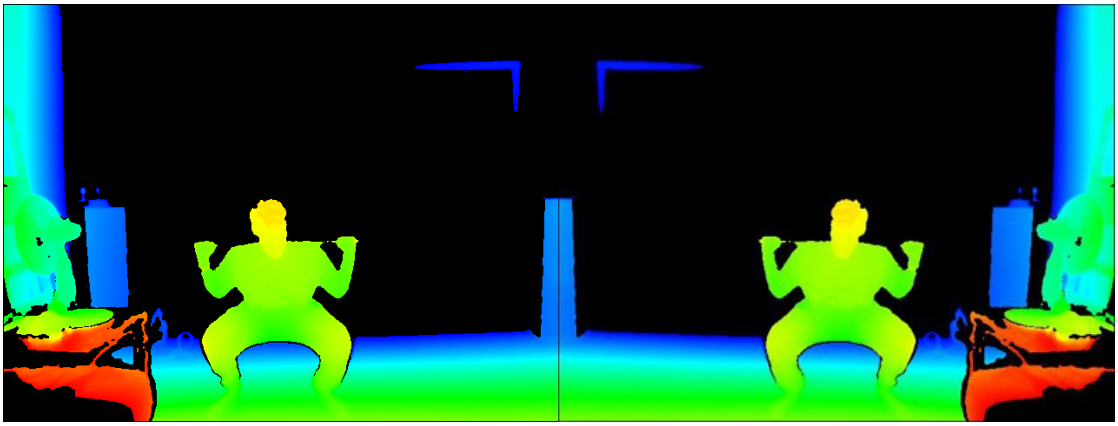
- shift_limit ((float, float) or float): Verschiebungsfaktor für die Y- und X-Achse. Die absoluten Werte sollten zwischen [0, 1] liegen. Parameter: (-0.06, 0.06).
- scale_limit ((float, float) or float): Skalierungs Faktor. Parameter: (-0.1, 0.1).

- rotate_limit ((int, int) or int): Rotationslimit des Bildes in Grad. Dieser Wert darf in diesem Anwendungsfall allerdings nur geringfügig ausfallen. Parameter: (-5, 5).

CoarseDropout Fügt Ausfallbereiche anhand der Parameter hinzu:

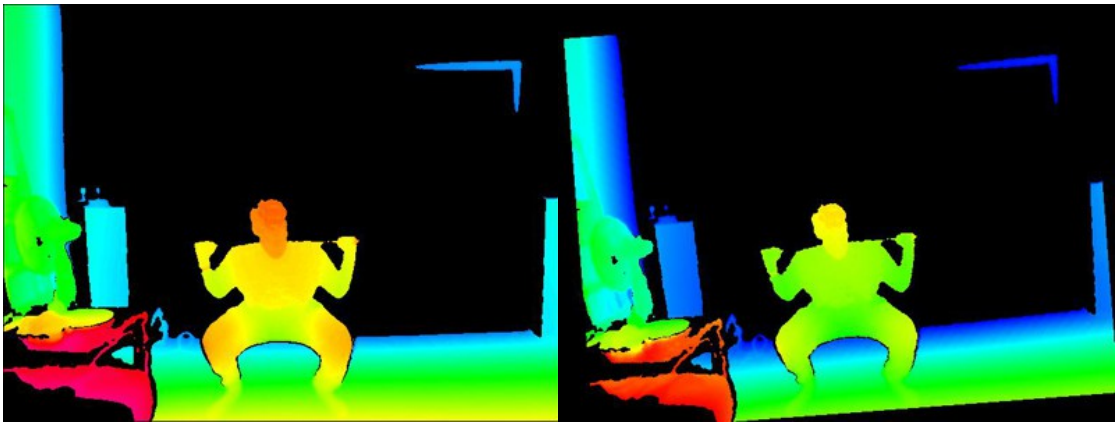
- **max_holes** (int): Maximale Anzahl der Fehlerbereiche. Parameter : (8)
- **max_height** (int): Maximale Höhe des Fehlerbereiches. Parameter : (8)
- **min_width** (int): Maximale Breite des Fehlerbereiches. Parameter : (8)

ElasticTransform Elastische Deformation der Bilder.



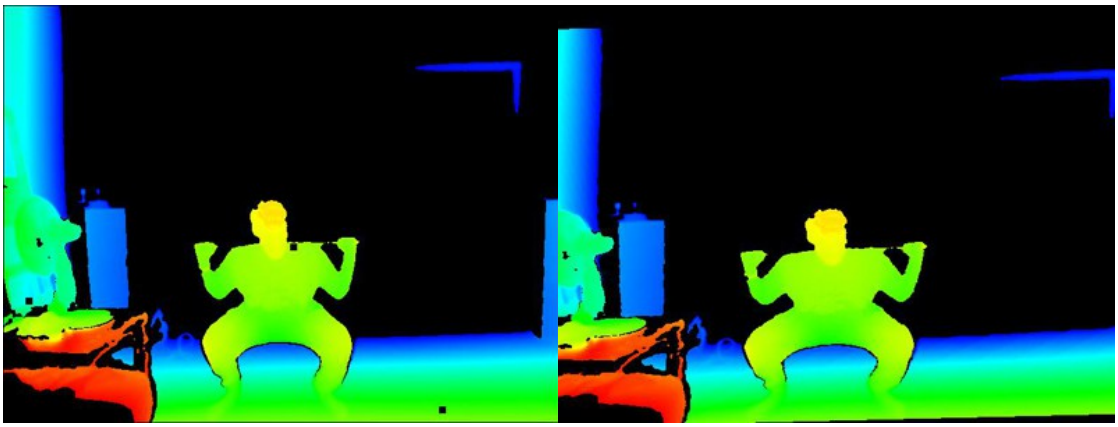
(a) Frame ohne Augmentation.

(b) Frame mit HorizontalFlip.



(c) Frame mit HueSaturationValue.

(d) Frame mit ShiftScaleRotate.



(e) Frame mit CoarseDroptout.

(f) Frame mit ElasticTransform.

Abbildung 4.1: Beispielhafte Darstellung der verwendeten Augmentationen mittels der Albumentation Bibliothek.

Mittels des Python Skripts *dataset\augmentation.ipynb* [2] wird jede Übung mit einem separaten Seed versehen und Augmentiert. Dies geschieht 5 mal, wodurch der Datensatz verfünffacht wird. Anschließend werden die einzelnen Frames einer Übung zu separaten mp4 Daten zusammengeführt und in Abhängigkeit des Ordnernamens umbenannt. Abschließend wird der komplette Datensatz zufällig bei einer Rate von 5 zu 1 in den Train- und Testordner sortiert.

```
── /collapsingKnee                                  ── /collapsingKnee
|    |                                             |    |
|    ── /1                                         |    ── /1.mp4
|         |                                        |    |    2.mp4
          ── /squat-collapse-001.png               |    | ...
|         |    squat-collapse-002.png              |
|         | ...                                    |
|    ── /2                                         |
|         |                                        |
          ── /squat-collapse-001.png               |
|         |    squat-collapse-002.png              |
|         | ...                                    |
   | ...                                           |
── /slumpedShoulder                                 ── /slumpedShoulder
|    |                                             |    |
|    ── /1                                         |    ── /1.mp4
|         |                                        |    |    2.mp4
          ── /squat-slumped-001.png                |    | ...
|         |    squat-slumped-002.png               |
|         | ...                                    |
|    ── /2                                    ->   |
|         |                                        |
          ── /squat-slumped-001.png                |
|         |    squat-slumped-002.png               |
|         | ...                                    |
   | ...                                           |
── /good                                            ── /good
|    |                                             |    |
|    ── /1                                         |    ── /1.mp4
|         |                                        |    |    2.mp4
          ── /squat-good-001.png                   |    | ...
|         |    squat-good-002.png                  |
|         | ...                                    |
|    ── /2                                         |
|         |                                        |
          ── /squat-good-001.png                   |
|         |    squat-good-002.png                  |
|         | ...                                    |
   | ...                                           |
```

Auflistung 4.1: Beispiel der Dateiumbenennung anhand der Ordnerstruktur.

4.3.3 Extraktion des Ground Truths

Um den Datensatz für die weitere Verarbeitung in den genutzten neuronalen Netzen bereitzustellen, wird das Skript *LSTM\formatDataset.ipynb* [2] verwendet. Dieses verwendet die gegebene Ordnerstruktur um die Daten in eine CSV Datei zu überführen.

4.3.4 Aufbereitung des Datensatzes

Eine Herausforderungen des Trainings eines Video Klassifizierers, ist es die Video-Dateien für das Netzwerk verfügbar zu machen. Da ein Video eine geordnete Sequenz von Frames darstellt, könnte man hierfür die einzelnen Bilder in ein 3D Tensor übertragen. Wie allerdings bereits in Kapitel 1.2 angesprochen, könnte ein Video und auch ein Übungsdurchlauf aus einer variablen Menge an Bildern bestehen. Dies erschwert die einfache Nutzung von 3D Tensoren. Es muss deshalb folgendes durchgeführt werden:

- Aufnahme der einzelnen Bilder eines Videos.
- Entnahme der Frames aus dem Video bis eine maximale Anzahl erreicht wurde.
- Falls ein Video keine ausreichende Zahl an Bilder enthält, den Rest des Videos mit Nullen auffüllen.

4.4 Trainieren der neuronalen Netze

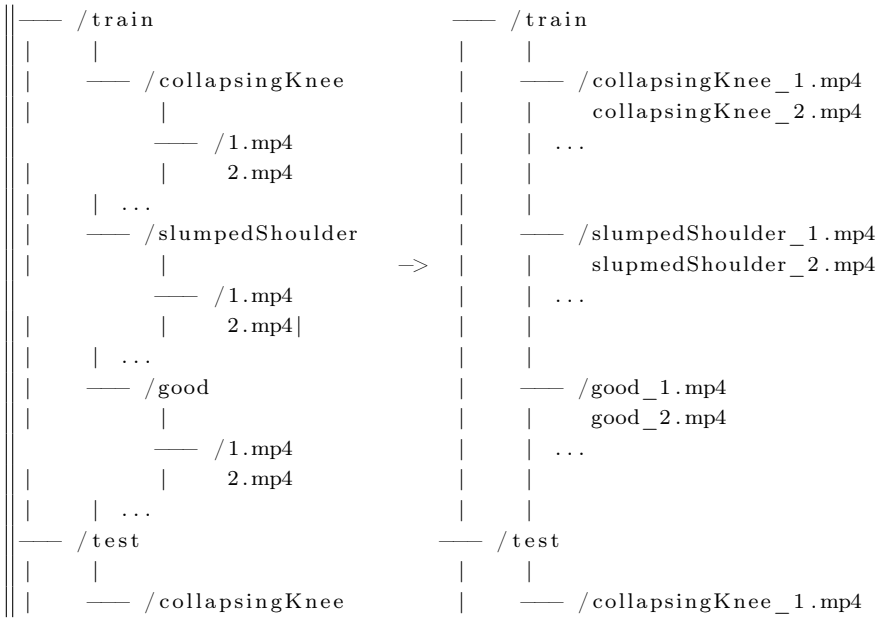
Die neuronalen Netze werden, wie im Kapitel 3.1.2 geschildert, auf einer Jupyter-Notebook Umgebung trainiert. In den folgenden Kapiteln werden die drei unterschiedlichen Netzwerke näher besprochen. Um eine Vergleichbarkeit zwischen den Netzen besser abbilden zu können, siehe Kapitel 1.2, beschränken wir uns bei den Aufnahmen auf eine Konstante Framelänge von 23 Bildern. Die Netzwerkarchitektur des Transformers und des LSTM's sind dabei größtenteils identisch, wodurch diese beiden Netze bis auf Ausnahmen gleich bearbeitet und hier beschrieben werden.

4.4.1 Hyperparameter der neuronalen Netze

Zunächst wurden die Hyperparameter für den weiteren Verlauf des Netzwerkes festgelegt. Obwohl ein LSTM in der Lage ist mit variablen Videolängen zu lernen, wird eine maximale Länge für Videos benötigt. Diese wird mittels des Parameters *MAX_SEQ_LENGTH* auf 23 gesetzt. Die Anzahl an Epochen die für das Training verwendet werden, wird dabei mittels *EPOCHS* auf 50 angesetzt. Dies stellt eine gute Grundlage dar um Overfitting vorzubeugen, da es bei der Größe des Datensatzes ansonsten zu Problemen führen kann.
Für die Netze wurde die *sparse_categorical_crossentropy* Lossfunktion und der *adam* Optimierer von Keras verwendet. *Sparse_categorical_crossentropy* wird verwendet, um den crossentropy Verlust zwischen den Labels und den Vorhersagen zu bestimmen wenn es mehr als zwei Labelklassen gibt.

4.4.2 Formatieren und beschriften des Datensatzes

Um den in Kapitel 4.3 beschriebenen Datensatz für die weitere Verarbeitung mit den beiden Netzen LSTM und Transformer vorzubereiten, werden die Videos mittels des Skriptes *formatDataset.ipynb*[2] gelabelt. Das Skript benennt dabei zunächst alle Dateien anhand des Ordners in dem sich diese befinden um.



```
|          |                          |     |     collapsingKnee_2.mp4
           -- /1.mp4                  |     |  ...
|          |     2.mp4                |     |
|     | ...                          |     |
|     -- /slumpedShoulder            |     -- /slumpedShoulder_1.mp4
|          |                    ->    |     |     slupmedShoulder_2.mp4
           -- /1.mp4                  |     |  ...
|          |     2.mp4|               |     |
|     | ...                          |     |
|     -- /good                       |     -- /good_1.mp4
|          |                         |     |     good_2.mp4
           -- /1.mp4                  |     |  ...
|          |     2.mp4                |     |
|     | ...                          |     |
```

Auflistung 4.2: Beispiel der Dateiumbenennung anhand der Ordnerstruktur.

	video_name	tag
1	good_15.mp4	good
2	collapsingKnee_1.mp4	collapsingKnee
3	slumpedShoulder_9.mp4	slumpedShoulder
...	...	...
1238	collapsingKnee_66.mp4	collapsingKnee

Tabelle 4.1: Aufbau der CSV-Dateien welcher den Dateipfad mit dem zugehörigen Label(tag) verbindet.

Daraufhin werden zwei CSV-Dateien erstellt, welche anhand des Namens der Datei eine Liste mit allen Dateien und deren Label erstellt. Es wird eine Liste mit allen *Test* und allen *Train* Datensätzen erzeugt, welche wie folgt aussehen:

	video_name	tag
1	good_15.mp4	good
2	collapsingKnee_1.mp4	collapsingKnee
3	slumpedShoulder_9.mp4	slumpedShoulder
...	...	...
1238	collapsingKnee_66.mp4	collapsingKnee

Tabelle 4.2: Aufbau der CSV-Dateien welcher den Dateipfad mit dem zugehörigen Label(tag) verbindet.

Anders als für die beiden Netze LSTM und Transformer, werden die Daten für das CNN als Bilderstapel abgelegt. Dabei wird jede Übung im Datensatz zusammengeführt und als Bildmontage abgelegt (siehe Abbildung 4.2).

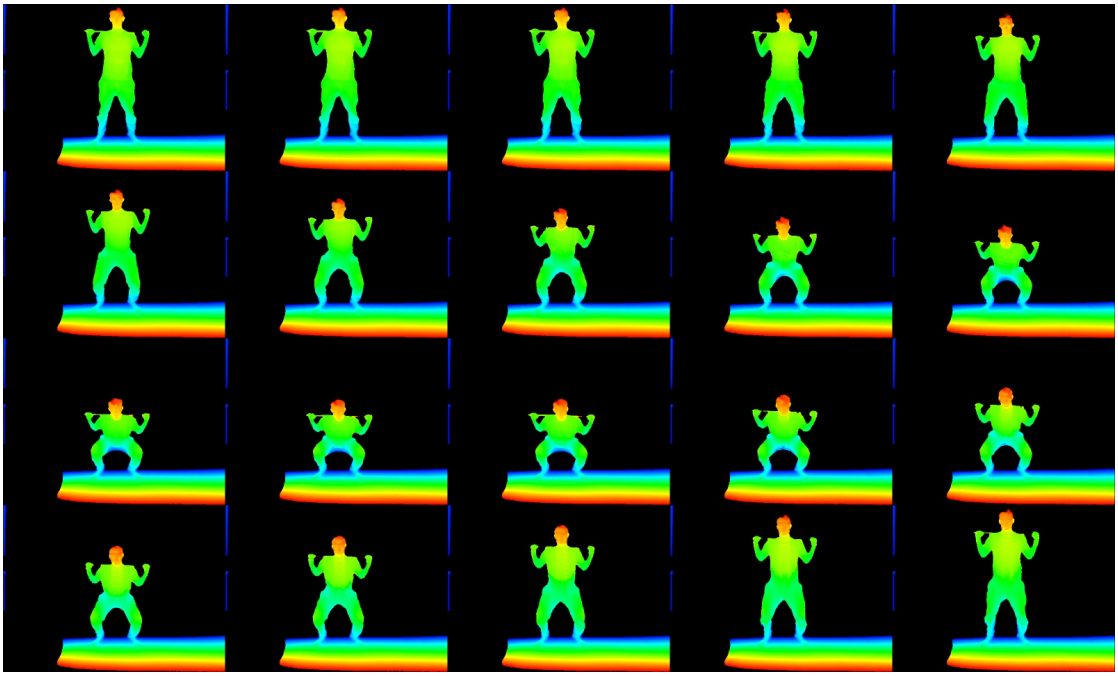


Abbildung 4.2: Datensatzbeispiel für das CNN Netz.

4.4.3 Analyse der gelernten Merkmale

Ein neuronales Netz lernt verschiedene Merkmale in einem Bild zu erkennen. Ohne einen weiteren Blick auf die Schwerpunkte zu nehmen, kann somit nicht unmittelbar festgestellt werden, welche Muster zu der Entscheidung eines neuronalen Netzes beigetragen haben.

So könnte es sein, dass bei einer bestimmten Übung immer ein Stift im Aufnahmebereich liegt und das Netz nun den Stift als einfache Markierung nutzt, um die Übung einfach zu kategorisieren, anstatt die Übung selbst zu beachten. Dies ist besonders deshalb möglich, da der hier verwendete Datensatz in Intervallen aufgenommen wurde und somit viele gleiche Übungen an einem Stück absolviert wurden. Bereits eine leicht andere Ausgangsstellung kann dem neuronalen Netz einen Hinweis auf die Art der Übung geben. Um dieses Risiko zu minimieren wurde, wie bereits in Abschnitt **??** erwähnt, eine Anzahl von Bildaugmentationen unternommen, die das Bildmaterial in unterschiedliche Richtungen verschiebt und die Tiefenwerte leicht ändert, um einen unterschiedlichen Abstand und

eine unterschiedliche Orientierung zum Objektiv zu simulieren.

Mittels *Grad-CAM Klassen Aktivierung* wird visualisiert, welche Bereiche der Daten für die Klassifizierung der Durchführung von den Netzen betrachtet werden. Die hierdurch erhaltene Heatmap 4.3 kann mit dem Ausgangsbild überlagert werden, wodurch man erkennen kann, welche Bereiche besonders viele Aktivierungen haben (Abbildung 4.4). Dies wurde sowohl mit der einfachen CNN-Architektur sowie auch mit dem LSTM-Netz durchgeführt (Abbildung 4.5). Bei der LSTM Architektur wurde dabei jeder einzelne Frame mit der Heatmap überlagert. Die Einzelbilder wurden hier für eine bessere Übersicht zusammengeführt. [3]

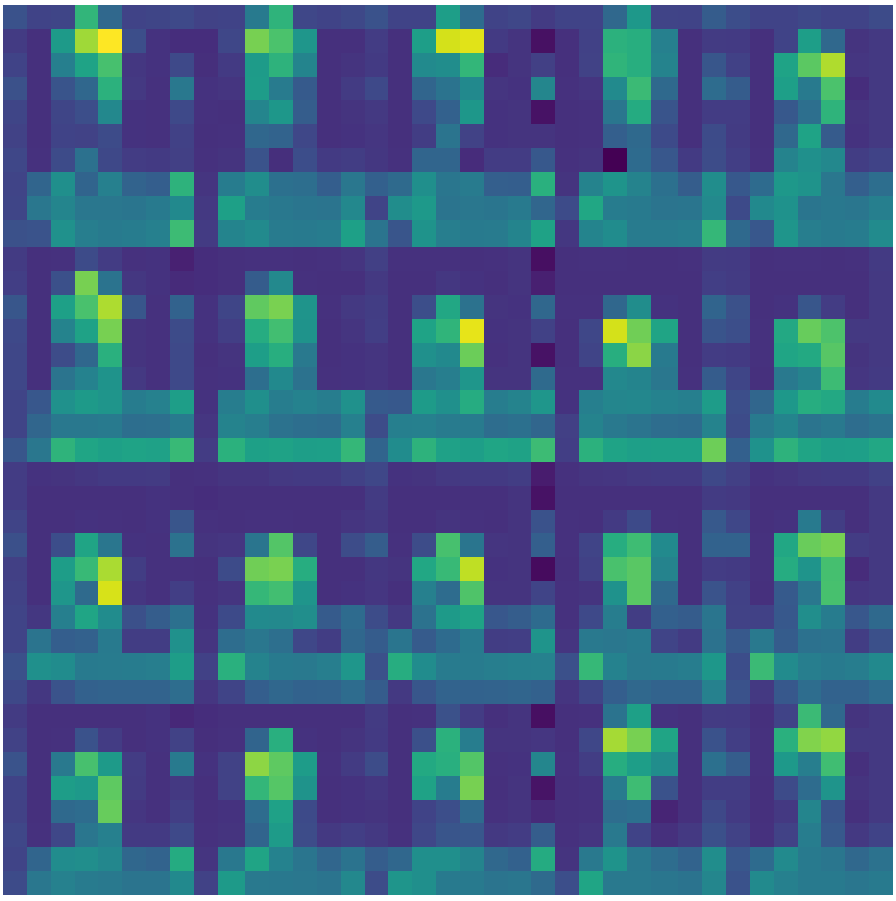


Abbildung 4.3: Heatmap einer fehlerhaften Ausführung bei der CNN Architektur.

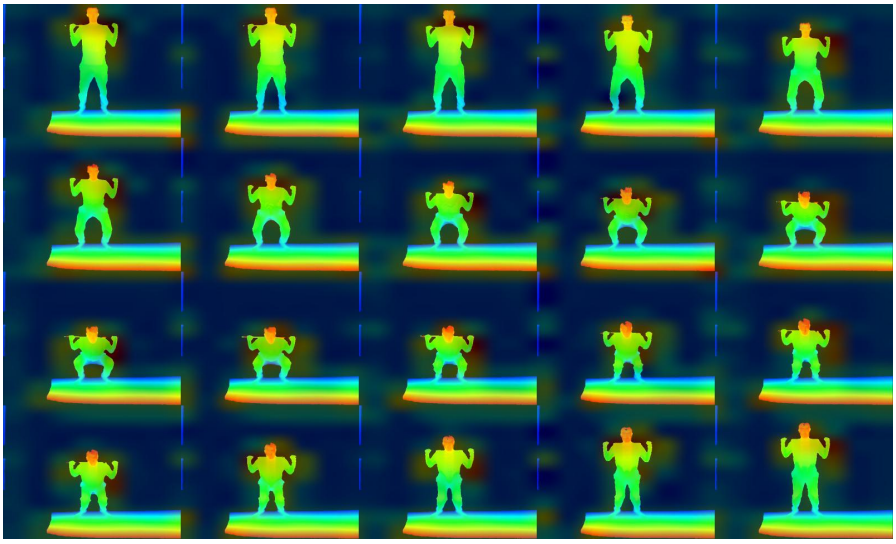


Abbildung 4.4: Heatmap kombiniert mit dem originalen Bild einer fehlerhaften Ausführung bei der CNN Architektur.

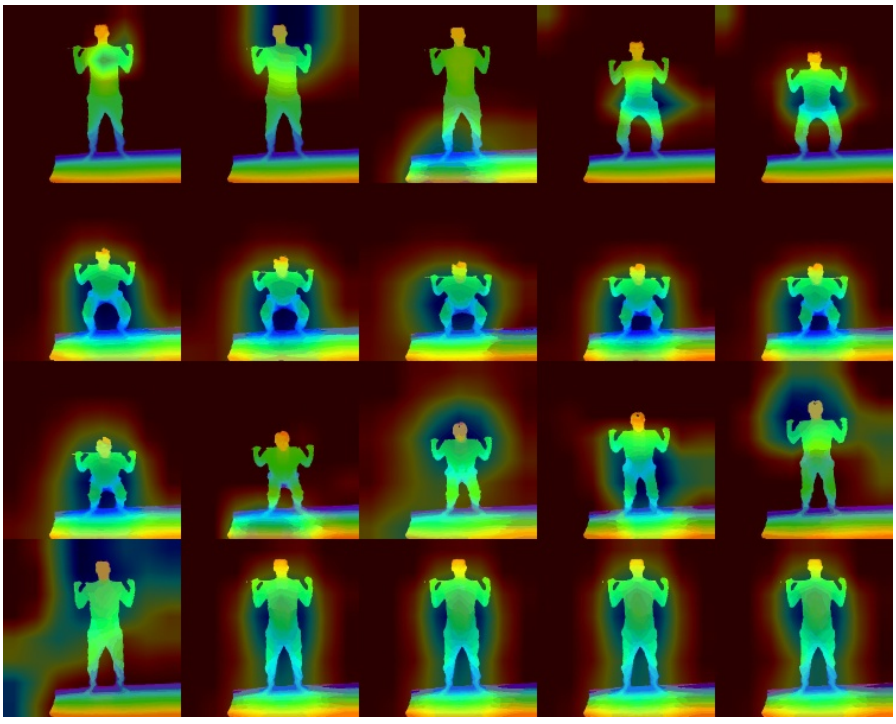


Abbildung 4.5: Heatmap kombiniert mit dem originalen Bild einer fehlerhaften Ausführung bei der LSTM Architektur.

4.5 Lernkurven

Die Ergebnisse der drei verschiedenen Architekturen werden im folgenden vorgestellt. Anhand der Abbildung 4.6, sowie 4.7 kann eine gute Accuracy Rate erkannt werden. Hierdurch kann man ableiten, dass die beiden Netze LSTM und Transformer nicht besonders von Overfitting betroffen sind und eine hohe Genauigkeit aufweisen. Das CNN Netzwerk schafft es nicht eine zutreffende Klassifizierung zu erstellen (siehe 4.8). Die maximale Genauigkeit liegt hierbei bei 55%. Besonders auffällig ist der Unterschied zwischen den Trainingsdaten und den Validierungsdaten. Dies deutet eine zu hohe Varianz zwischen den Trainings und den Validierungsdaten an. Während das Netzwerk zuverlässiger in der Lage ist mit den Testdaten umzugehen sind die Validierungsdaten zu unterschiedlich und führen zu hohen Schwankungen und Spitzen in den Epochen.

Der Vergleich zwischen dem LSTM und dem Transformer Netzwerk ist dagegen weniger extrem. Es ist allerdings zu erkennen, dass das Transformer-Netz schneller lernt und sich bei einem Wert von über 95% Genauigkeit einpendelt. Der Genauigkeitswert des LSTM Netzes liegt leicht über dem maximalen Wert des Transformer Netzes. Dies ist aus der Grafik nicht unmittelbar zu erkennen. Die Extremen Spitzen bei der Validierungsgenauigkeit deutet auf lokale Maxima hin.

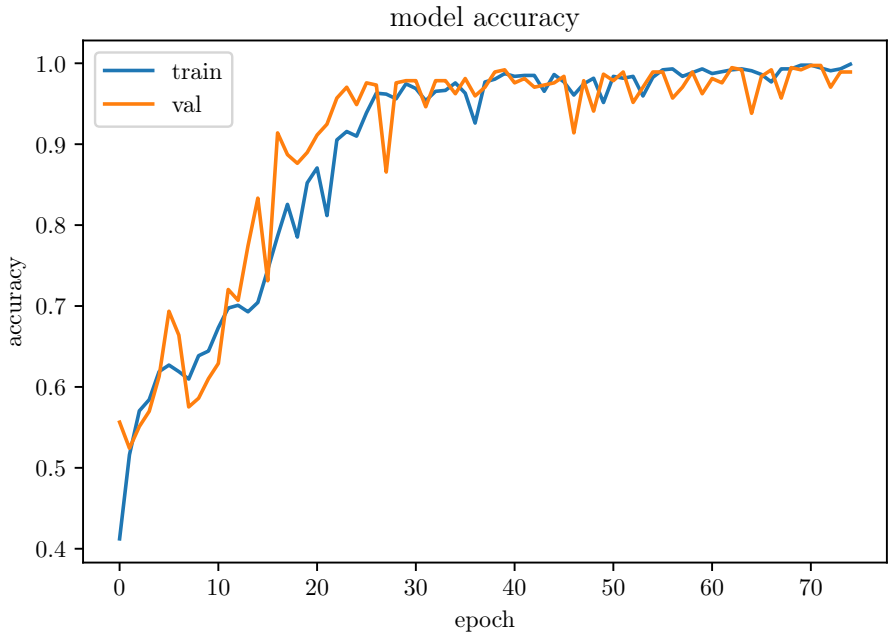


Abbildung 4.6: LSTM-Model Genauigkeit.

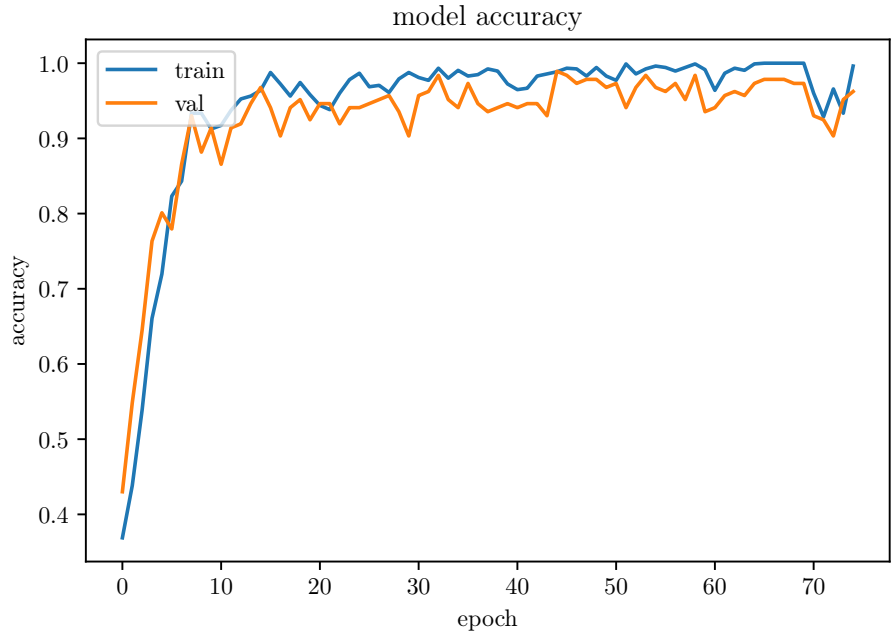


Abbildung 4.7: Transformer-Model Genauigkeit.

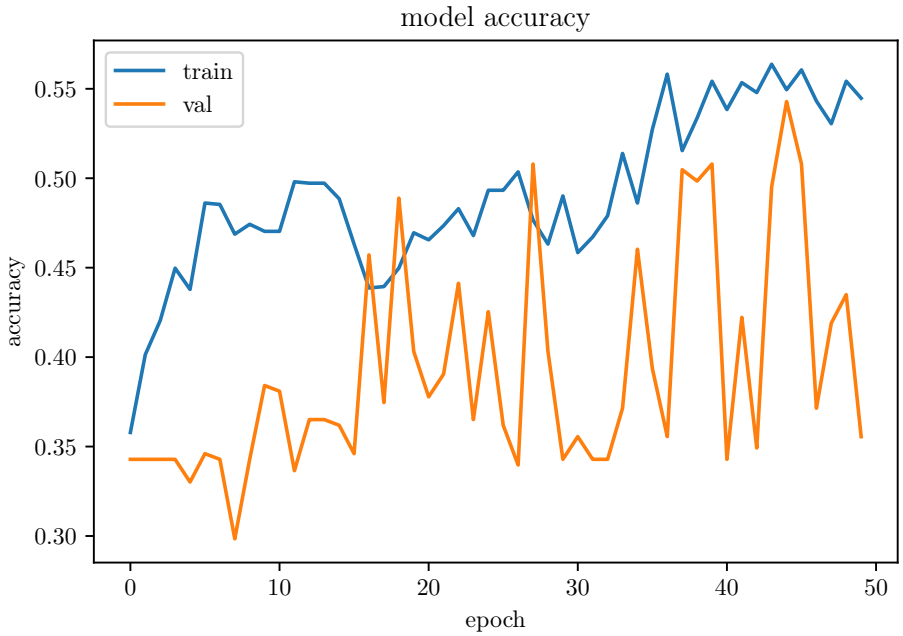


Abbildung 4.8: CNN-Model Genauigkeit.

5 Fazit

Das Ziel dieser Arbeit bestand darin, zu ermitteln, ob die Erkennung bestimmter Bewegungsabläufe mittels 3D Datenmaterial und neuronalen Netzwerken möglich ist. Ein besonderer Faktor war in diesem Fall die geringe Unterscheidung der Bewegungsabläufe in unterschiedliche Fehlerkategorien und der Einsatz eines selbst erstellten Datensatzes. Um den Aufwand zur Erzeugung eines Datensatzes zu minimieren, wurde für die Machbarkeitsstudie zunächst nur eine Übung mit 2 Fehlerquellen, sowie eine ordnungsgerechte Variante aufgenommen und untersucht. Der Datensatz wurde mittels einer Basler Blaze 101 in 640 x 480 Pixeln aufgenommen und spiegelt lediglich eine Person in immer dem gleichen Raum dar. Um den Datensatz dennoch gegen Overfitting zu schützen kam Data-Augmentation zum Einsatz.

Es wurden drei verschiedene Netze gegeneinander getestet. So wurde zunächst ein Standard CNN mit den Videodaten als Bilderstapel trainiert. Daraufhin wird eine CNN-LSTM Architektur verwendet, wobei das verwendete InceptionV3 Model mittels ImageNet-1k dataset vortrainiert ist und features für das LSTM extrahiert. Abschließend wurde ein Netzwerk auf Basis eines Transformers trainiert. Alle Netze wurden dabei mit der selben Anzahl an Epochen trainiert um einen bestmöglichen Vergleich anstellen zu können.

Die entstandenen Resultate verdeutlichen dabei die Überlegenheit der LSTM und Transformer Architekturen gegenüber einem einfachen CNN Netzwerk. Diese beiden Netze weisen jeweils eine Genauigkeit von über 90% auf und sind in der Lage die Übung erfolgreich zu klassifizieren.

5.0.1 Ausblick und Verbesserungsansätze

Aufgrund des beschränkten Umfanges einer Bachelorarbeit kam es bei dieser Arbeit um einige Einspaarungen. So wurde die Sequenzlänge auf 23 Bilder gehalten um einen bes-

seren Vergleich zwischen CNN und LSTM/Transformer Architektur zu schaffen und die Daten vereinfacht verarbeiten zu können. Die Arbeit ist als ersten Schritt zu sehen, um die Machbarkeit der Unterscheidung minimaler Unterschiede aufzuzeigen. Die gefundenen Daten zeigen dabei deutlich einen positiven Ausblick, wodurch in einem nächsten Schritt mit dynamischen Sequenzen experimentiert werden kann.
Eine weitere Einschränkung dieser Arbeit stellt der Datensatz dar. So wurde nur eine Person in einem limitierten Verlauf aufgenommen und mit 3 verschiedenen Fehlerstufen gearbeitet. Eine Erweiterung ist notwendig, wobei mehrere Personen in unterschiedlichen Umgebungen, bei einer Mehrzahl an unterschiedlichen Übungen mit jeweiligen Fehlerformen aufgenommen und die Netze trainiert müssen.

6 Quellen

https://arxiv.org/abs/1512.00567
https://keras.io/examples/vision/video_classification/
https://www.baslerweb.com/de/produkte/kameras/3d-kameras/basler-blaze/blaze-101/
https://docs.baslerweb.com/hardware-installation-(blaze)
https://towardsdatascience.com/cnn-heat-maps-class-activation-mapping-cam-614180e360d5
https://blog.coast.ai/five-video-classification-methods-implemented-in-keras-and-tensorflow-99cad29cc0b5
https://www.analyticsvidhya.com/blog/2021/06/building-a-convolutional-neural-network-using-tensorflow-keras/
https://www.datarobot.com/blog/introduction-to-dataset-augmentation-and-expansion/
https://sefiks.com/2017/12/10/transfer-learning-in-keras-using-inception-v3/
https://builtin.com/machine-learning/loss-functions
https://www.baeldung.com/cs/ml-loss-accuracy
https://towardsdatascience.com/lstm-networks-a-detailed-explanation-8fae6aefc7f9 https://www.theaidream.com/post/transformer-neural-network-in-deep-learning-explained

Literatur

[1] Alexander Buslaev u. a. “Albumentations: Fast and Flexible Image Augmentations”. In: *Information* 11.2 (2020). ISSN: 2078-2489. DOI: 10.3390/info11020125. URL: https://www.mdpi.com/2078-2489/11/2/125.

[2] Daniel von Drathen. *Neuronale Klassifikation von Bewegungsabläufen mittels Time-of-Flight Kameras*. URL: https://github.com/S41nts4w/BA.

[3] Ramprasaath R. Selvaraju u. a. “Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization”. In: *International Journal of Computer Vision* 128.2 (2019), S. 336–359. DOI: 10.1007/s11263-019-01228-7. URL: https://doi.org/10.1007%2Fs11263-019-01228-7.

[4] Ashish Vaswani u. a. *Attention Is All You Need*. 2017. DOI: 10.48550/ARXIV.1706.03762. URL: https://arxiv.org/abs/1706.03762.

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Gemäß der Allgemeinen Prüfungs- und Studienordnung ist zusammen mit der Abschlussarbeit eine schriftliche Erklärung abzugeben, in der der Studierende bestätigt, dass die Abschlussarbeit „— bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit [(§ 18 Abs. 1 APSO-TI-BM bzw. § 21 Abs. 1 APSO-INGI)] — ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt wurden. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen."

Quelle: § 16 Abs. 5 APSO-TI-BM bzw. § 15 Abs. 6 APSO-INGI

Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: _____

Vorname: _____

dass ich die vorliegende Bachelorarbeit – bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit – mit dem Thema:

Neuronale Klassifikation von Bewegungsabläufen mittels Time-of-Flight Kameras

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

_____	_____	_____
Ort	Datum	Unterschrift im Original