Zürcher Hochschule
für Angewandte Wissenschaften

School of Engineering
CAI Centre for Artificial Intelligence

Projektarbeit (Informatik)

Machine Learning-Based Analysis of Data from the ZHAW Movement Analysis Laboratory for Fatigue Detection during Sports Exercises

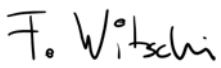
Autoren	Benjamin Stern Florian Witschi
Hauptbetreuung **Nebenbetreuung**	Prof. Dr. Thilo Stadelmann Dr. Eveline Graf

Datum	22. Dezember 2022

Erklärung betreffend das selbstständige Verfassen einer Projektarbeit an der School of Engineering

Mit der Abgabe dieser Projektarbeit versichert der/die Studierende, dass er/sie die Arbeit selbständig und ohne fremde Hilfe verfasst hat. (Bei Gruppenarbeiten gelten die Leistungen der übrigen Gruppenmitglieder nicht als fremde Hilfe.) Der/die unterzeichnende Studierende erklärt, dass alle zitierten Quellen (auch Internetseiten) im Text oder Anhang korrekt nachgewiesen sind, d.h. dass die Projektarbeit keine Plagiate enthält, also keine Teile, die teilweise oder vollständig aus einem fremden Text oder einer fremden Arbeit unter Vorgabe der eigenen Urheberschaft bzw. ohne Quellenangabe übernommen worden sind.

Bei Verfehlungen aller Art treten die Paragraphen 39 und 40 (Unredlichkeit und Verfahren bei Unredlichkeit) der ZHAW Prüfungsordnung sowie die Bestimmungen der Disziplinarmassnahmen der Hochschulordnung in Kraft.

Ort, Datum:	**Studierende:**
	
22. Dezember 2022	Benjamin Stern
	
22. Dezember 2022	Florian Witschi

Zusammenfassung

Im Bewegungslabor des Gesundheitsdepartements der ZHAW wird in klinischen Studien eine Vielzahl von Bewegungsaten erhoben. Bisher wurden diese Daten mit klassischen statistischen Methoden ausgewertet. Da sich maschinelles Lernen gut zur Erkennung von Mustern eignet, stellt sich die Frage, ob es auch zur Detektion der Ermüdung eines Sportlers eingesetzt werden kann. Die verwendeten Daten wurden von 20 Leistungssportler in jeweils 25-minütigen HIIT-Trainings (High Intensity Interval-Training) aufgenommen. Ein Training umfasst klassische Übungen wie Sprünge, Ausfallschritte und Burpees. Die Idee ist eine Veränderung im Bewegungsmuster zu erkennen. Unter der Annahme, dass sich das Bewegungsmuster mit zunehmender Ermüdung verändert. Zu diesem Zweck wird ein Autoencoder verwendet, der die Daten zu rekonstruieren lernt. Allerdings wird er nur mit den Daten zu Beginn des Trainings trainiert, wenn die Sportler noch in guter Verfassung sind. Wenn der Autoencoder die Daten am Ende des Trainings rekonstruiert, sollte der Rekonstruktionsfehler steigen, da sich das Bewegungsmuster verändert hat. Dieser Veränderung wäre auf eine Ermüdung der Sportler zurückzuführen. Als Autoencoder wird zum einen ein RNN mit LSTM-Zellen und zum anderen ein CNN-Autoencoder verwendet. Es hat sich gezeigt, dass der CNN-Autoencoder besser in der Lage ist, die Daten zu erlernen und einen sehr kleinen Rekonstruktionsfehler auf den Trainingsdaten erreicht. Gleichzeit benötigt der CNN-Autoencoder um ein vielfaches weniger Parameter als der LSTM-Autoencoder. Allerdings steigt der Rekonstruktionsfehler während des gesamten Trainings nicht an, und es war nicht möglich, die Müdigkeit anhand der Bewegungsdaten zu erkennen. Diese Resultate können jedoch als Grundlage für weitere Arbeiten in diesem Bereich dienen und sind daher als inkonklusiv zu betrachten.

Abstract

In the movement laboratory of the ZHAW Health Department, a large number of movement data are collected in clinical studies. So far, these data have been analyzed using classical statistical methods. Since machine learning is well suited for detecting patterns, the question arises whether it can also be used to detect fatigue in an athlete. The data used were recorded from 20 competitive athletes in 25-minute high intensity interval training (HIIT) workouts each. A workout includes classic exercises such as jumps, lunges and burpees. The idea is to detect a change in the movement pattern, assuming that the movement pattern changes with fatigue. For this purpose, an autoencoder is used that learns to reconstruct the data. Although, it is only trained with the data at the beginning of the training when the athletes are still in good shape. When the autoencoder reconstructs the data at the end of the training, the reconstruction error should increase since the movement pattern has changed. This change would be due to the fatigue of the athletes. One of the autoencoders used is an RNN with LSTM cells and the other is a CNN autoencoder. It has been shown that the CNN autoencoder is better at learning the data and achieves a very small reconstruction error on the training data. At the same time, the CNN autoencoder requires many times fewer parameters than the LSTM autoencoder. However, the reconstruction error does not increase throughout the training, and it was not possible to detect fatigue from the motion data. Nevertheless, these results can serve as a basis for further work in this area and should therefore be considered as inconclusive.

Vorwort

Diese Arbeit ist unsere Projektarbeit, die als Vorbereitung für die Bachelorarbeit im nächsten Semester dient, in unserem Informatik Studium an der ZHAW School of Engineering. Erste Kenntnisse über neuronale Netze haben wir bereits im Rahmen des Moduls "Maschinelles Lernen und Data Mining" erworben. Dennoch haben wir uns weiteres Wissen über neuronale Netze und insbesondere über verschiedene Architekturen und Modellarten im Verlauf dieser Arbeit angeeignet.

Speziell bedanken möchten wir uns bei unserem Hauptbetreuer Prof. Dr. Thilo Stadelmann und unserer Nebenbetreuerin Dr. Eveline Graf für die Unterstützung während dieser Arbeit.

Ein grosser Dank geht auch an Konstantinos Balaskas und Kostas Siozios, die uns den Code zu ihrer Arbeit zur Verfügung gestellt haben [3].

Inhaltsverzeichnis

1. Einleitung

1.1. Ausgangslage

Die folgende Arbeit wird an der ZHAW unter der Betreuung von Prof. Dr. Thilo Stadelmann des Center for Artificial Intelligence und Dr. Eveline Graf des Institute of Physiotherapy durchgeführt. In dieser Arbeit wird untersucht, wie künstliche Intelligenz eingesetzt werden kann, um zu erkennen, wann die Leistungssportlerin im ExerCube erschöpft ist. Dies ist vor allem in einer späteren Phase einer Rehabilitation von einer Verletzung von Interesse, kurz vor dem Zeitpunkt an dem eine Sportlerin wieder Wettkämpfe bestreiten wird. Zu dieser Zeit soll die Sportlerin möglichst nahe an ihrer Leistungsgrenze trainieren, ohne dabei den zuvor verletzten Körperteil in der Regeneration zu überlasten. Dadurch kann die Athletin in einer kontrollierten Umgebung in kürzester Zeit wieder an das kompetitive Umfeld herangeführt werden.

Bisher wurden die Daten mit klassischen statistischen Methoden in Matlab aufbereitet und ausgewertet. Dabei wurden einzelne Parameter, wie zum Beispiel der maximale Kniegelenkwinkel über die Zeit analysiert. Diese klassischen Ansätze erfordern jedoch zu Beginn eine Auswahl von Merkmalen, die auf ausreichendem Hintergrundwissen und einer subjektiven Meinung beruhen [18]. Da die menschlichen Bewegungsabläufe nicht linear sind, bieten sich datenbasierte Ansätze in Kombination mit Machine Learning-Verfahren zur Auswertung der Daten besonders an [4].

In zahlreichen bisherigen Analysen mit datenbasierten Ansätzen wurden repetitive Bewegungsabläufe analysiert [1, 3, 7–9, 12, 20], einige davon verwendeten Daten von EKG- oder IMU-Sensoren [3, 7, 20]. In der vorliegenden Arbeit werden hingegen die räumlichen Ortskoordinaten mit der entsprechenden X-, Y- und Z-Position im Raum als Daten herangezogen. Die Daten enthalten keine Angaben über die aktuelle Müdigkeit des Sportlers und werden daher in einem unsupervised Setting analysiert. Beim Unsupervised Learning werden verschiedene Methoden wie zum Beispiel die Principal Component Analysis (PCA) zur Feature-Extraktion oder neuronale Netze in der Form eines Autoencoders in Verbindung gebracht [9].

In einer vergangenen Arbeit haben Balaskas und Siozios [3] die Müdigkeitserkennung mithilfe eines LSTM-Autoencoders erfolgreich umgesetzt. Die Verwendung von LSTM-Zellen ist auf ihre häufige Anwendung in Zeitreihen zurückzuführen, der Autoencoder hingegen trägt zum Erlernen der normalen Bewegungsabläufe bei. Es wurde bereits in verschiedenen Anwendungsbereichen erfolgreich demonstriert, dass eine neue Gegebenheit von einem LSTM-Autoencoder anhand des Rekonstruktionsfehlers erkannt werden kann. Beispielsweise wurde der Rekonstruktionsfehler dazu verwendet, ein neues akustisches Signal zu erkennen [14]. Die Erkennung von Ermüdung folgt der gleichen Annahme: Wenn ein Autoencoder auf den ersten paar Minuten trainiert wird, in denen der Athlet fit ist, steigt der Rekonstruktionsfehler während des Trainings mit zunehmender Ermüdung, da sich der Bewegungsablauf verändert.

Eine sehr ähnliche Ausgangslage hatten Hernandez u. a. [9], sie verwendeten auch 3D Bewegungsdaten und analysierten die Müdigkeit der Probanden supervised und unsupervised, wenn auch auf Daten von repetitiven Bewegungen. Die Anwendung von Autoencoder scheint ein vielversprechender Ansatz zu sein [3], beim Denoising von 3D Bewegungsdaten wurde die Effektivität bereits gezeigt [30] und daher vermuten wir auch auf variierenden Bewegungsabläufen, wie es bei den Daten vom ExerCube der Fall ist, gute Resultate zu erzielen.

Daher basiert die Baseline auf einem Autoencoder, welcher auf den ersten paar Minuten der 3D Bewegungsdaten trainiert wird. Mit der Annahme, dass im Laufe des Exergames der Rekonstruktionsfehler mit der grösser werdenden Müdigkeit ansteigt.

Die offizielle Aufgabenstellung ist im Anhang A.1 in vollständiger Form vorhanden.

Im folgenden Bericht werden trotz der deutschen Sprache durchgehend englische Begriffe vorkommen. Aufgrund derer breiten Verwendung im Feld der künstlichen Intelligenz wurden diese nicht auf Deutsch übersetzt, damit es nicht zu Verwirrung bei den Lesern kommt.

2. Theoretische Grundlagen

2.1. ExerCube

Der ExerCube ist ein Produkt der Firma Sphery, das körperliches mit mentalem Training verbindet. Er hat drei Wände auf die das Spiel "Sphery Racer" projiziert wird, wie in Abbildung 2.1 ersichtlich ist.

Abbildung 2.1.: Ein Spieler während eines Trainings im ExerCube

Dabei fahren die Spieler entlang einer Strecke mit verschiedenen Aufgaben. Diese sind jeweils mit einem Tor markiert und müssen vor dem Durchfahren absolviert werden. Zu den Aufgaben gehören klassische Fitnessübungen wie Burpees, Lunges, Jumps oder Punches. An Händen und Füssen sind Bewegungstracker angebracht, die feststellen, ob und zu welchem Zeitpunkt die Übungen ausgeführt wurden. So kann der Spieler Punkte sammeln und Levels hochsteigen. Je höher das Level, desto schneller die Abfolge der Übungen. Die Dauer des Trainings ist wählbar und der Endpunktestand wird nach Abschluss des Trainings angezeigt [21].

2.2. Bewegungsanalyse

Das Bewegungslabor vom Institut für Physiotherapie der ZHAW ist ein Raum ausgestattet mit diversen Sensoren und Systemen zur Bewegungsanalyse. Die Daten, welche in dieser Arbeit genutzt werden, wurden vom Motion Capture System Vantage der Firma Vicon erfasst [25]. Dieses besteht aus mehreren fix montierten Kameras (Abbildung 2.2) mit integrierten Infrarot LEDs und reflektierenden Markern, welche von den Probanden getragen werden. Durch Triangulation kann aus den verschiedenen Kamerabildern die Position der Marker, auf wenige Millimeter genau berechnet werden.

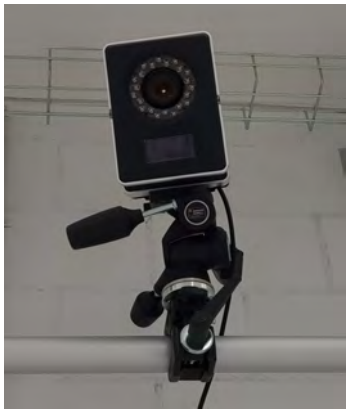


Abbildung 2.2.: Eine Kamera des Motion Capture Systems Vantage der Firma Vicon

2.3. Feedforward Neural Network

Neuronale Netze sind von der Art und Weise inspiriert, wie das menschliche Gehirn Informationen erfasst und verarbeitet. Die Grundlage bilden Neuronen, welche mehrere Inputs erhalten, mit diesen mathematische Berechnungen durchführen und das Resultat als Output weitergeben. Die Abbildung 2.3 beschreibt die Funktionsweise eines Neurons, welche im Grunde aus 3 Schritten besteht.

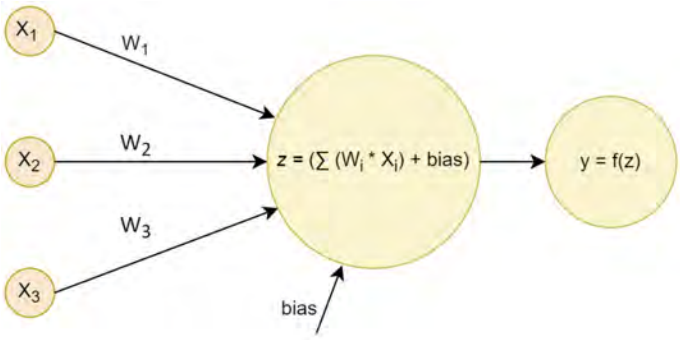


Abbildung 2.3.: Funktionsweise eines Neurons

Im ersten Schritt werden die entsprechenden Inputs mit den dazugehörigen Gewichten multipliziert. Dann wird die Summe z berechnet, indem die gewichteten Inputs aufsummiert werden und ein Bias dazu addiert wird. Zur Berechnung des Outputs y wird die Aktivierungsfunktion f auf die Summe z angewendet. Zahlreiche dieser Neuronen werden miteinander verknüpft und bilden folglich ein neuronales Netz, wie in Abbildung 2.4 abgebildet ist.

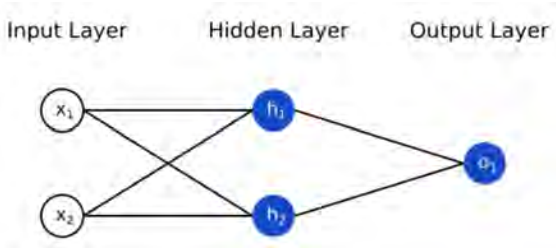


Abbildung 2.4.: Neuronales Netz mit einem Hidden Layer [29]

Dieses neuronale Netz hat zwei Inputs, ein Hidden Layer mit zwei Neuronen und ein Output Layer mit einem Neuron. Typischerweise wird ein neuronales Netz mit einem Hidden Layer als Shallow Neural

Network bezeichnet und sobald die Tiefe des neuronalen Netzes vier oder mehr beträgt, wird es als Deep Neural Network bezeichnet [27].

2.4. Reccurrent Neural Network

Die bereits vorgestellte traditionelle Art von neuronalen Netzen geht davon aus, dass es keine Abhängigkeiten zwischen den einzelnen Inputs gibt und somit können sie keine zeitlichen Zusammenhänge erlernen. Für sequentielle Daten und Zeitreihen, bei denen eine zeitliche Abhängigkeit zwischen den einzelnen Inputs besteht, können Recurrent Neural Networks (RNN) verwendet werden [22]. Sie zeichnen sich durch ihren Speicher aus, in dem Informationen aus vergangenen Inputs gespeichert werden, welche die aktuellen Outputs beeinflussen. Ein solches Netz wird auch als Unidirectional Reccurrent Neural Network bezeichnet, wie in Abbildung 2.5 dargestellt [16]. Das liegt daran, dass die Informationen für die Verarbeitung der Inputs nur aus den bereits verarbeiteten Inputs stammen und zukünftige Inputs keinen Einfluss darauf haben.

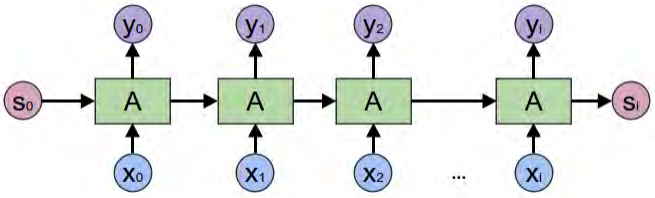


Abbildung 2.5.: Aufbau eines Unidirectional Reccurrent Neural Network [16]

Solche neuronalen Netze werden häufig in einer vereinfachten Form dargestellt, wie in Abbildung 2.6 abgebildet. Auf der linken Seite ist ein Rolled RNN zu sehen, dass das gesamte Netz darstellt und eine beliebig grosse Eingabe haben kann.

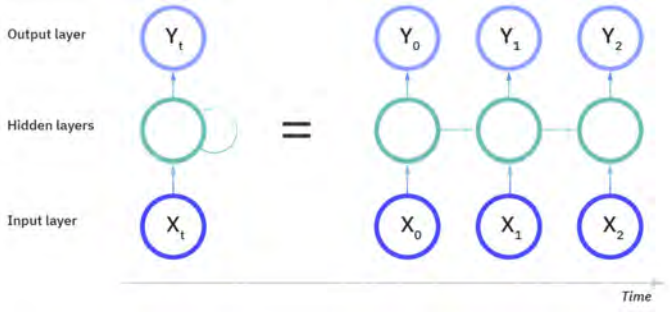


Abbildung 2.6.: Rolled (Rechts) & Unrolled RNN (Links) [26]

Im Unrolled RNN auf der rechten Seite werden die einzelnen Layers für die verschiedenen Inputs angezeigt. Es ist zu beachten, dass das Netz über die Zeit hinweg ausgerollt wird. Während Feedforward-Netze unterschiedliche Gewichte in den einzelnen Neuronen haben, besitzen die Neuronen in einem RNN in jeder Schicht die gleichen Gewichte. Daher auch der Name Reccurrent, da die Neuronen in den Hidden Layer wiederholend sind. Diese Gewichte werden ebenfalls mittels Backpropagation Through Time (BPTT) berechnet, was sich speziell für sequentielle Daten eignet. Im Prinzip funktioniert die Backpropagation genauso wie beim Feedforward Neural Network, nur dass die Fehler für jeden Zeitschritt aufsummiert werden, da die Parameter über den gesamten Layer geteilt werden. Typischerweise können zwei Probleme auftreten, die als exploding und vanishing gradients bekannt sind [26]. Im Falle des vanishing gradients werden die Gewichte so klein, dass sie unbedeutend werden und nicht mehr lernfähig sind. Im Gegensatz dazu werden die Gewichte beim Problem des expoding gradients so gross, dass sie am Ende nicht mehr als Zahl repräsentiert werden können.

2.4.1. Bidirectional Reccurrent Neural Network

Bidirectional RNN verbinden zwei Hidden Layer in unterschiedlicher Richtung mit demselben Output, wie in Abbildung 2.7 ersichtlich. Dies bewirkt, dass der Output aus gespeicherten Informationen von vergangenen und zukünftigen Inputs beeinflusst wird.

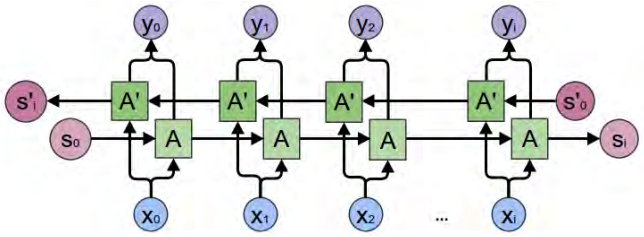


Abbildung 2.7.: Aufbau eines Bidirectional Reccurrent Neural Network [16]

2.5. Long Short Term Memory

Aufgrund des vanishing gradients in einem RNN ist es sehr schwierig, damit langfristige Zusammenhänge zu erfassen und diese zu merken [26]. Als Lösungsansatz haben Hochreiter u. a. [10] eine RNN-Architektur mit Long Short Term Memory (LSTM) Zellen empfohlen. Wie in der Abbildung 2.8 dargestellt, besitzen LSTMs ebenfalls eine kettenartige Struktur, allerdings verfügen die jeweiligen Zellen nicht nur über eine, sondern über vier Aktivierungsfunktionen. Das Herzstück einer LSTM-Zelle ist der

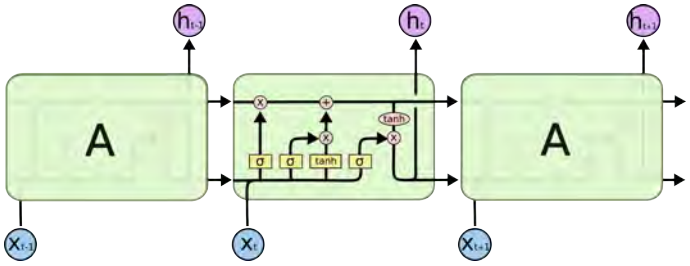


Abbildung 2.8.: Wiederholende Zellen in einem LSTM-Netzwerk mit vier Aktivierungsfunktionen [24]

Cell State C_t, der durch den oberen vertikalen Pfeil dargestellt wird, welcher sich quer durch die Zelle erstreckt. Mit den unterschiedlichen Aktivierungsfunktionen, auch Gates genannt, hat die LSTM-Zelle die Möglichkeit, Informationen zum Cell State hinzuzufügen oder zu entfernen. Der erste Schritt besteht darin, festzulegen, wie viele Informationen aus dem vorherigen Cell State beibehalten werden sollen. Zu diesem Zweck wird im Forget Gate unter Verwendung der Sigmoid-Aktivierungsfunktion ein Wert f_t zwischen 0 und 1 berechnet:

$$f_t = \sigma(W_f * [h_{t-1}, x_t] + b_f)$$

Bei einem Wert von 1 werden sämtliche Informationen aus dem vorherigen Cell State übernommen. Ist der Wert hingegen 0, werden alle Informationen verworfen. Der nächste Schritt besteht darin, zu entscheiden, welche neuen Informationen im Cell State gespeichert werden sollen. Als Erstes entscheidet das Input Gate, welche Werte aktualisiert werden:

$$i_t = \sigma(W_i * [h_{t-1}, x_t] + b_i)$$

Die Tanh-Aktivierungsfunktion erstellt einen Vektor $\tilde{C}_t$ mit neuen Werten, die dem Cell State hinzugefügt werden könnten:

$$\tilde{C}_t = tanh(W_C * [h_{t-1}, x_t] + b_C)$$

Anschliessend wird der alte Cell State C_{t-1} auf den neuen Cell State C_t aktualisiert. Dazu wird der alte Cell State mit f_t multipliziert, um die nicht benötigten Informationen zu entfernen. Danach wird $i_t * \tilde{C}_t$ addiert, um die neuen Informationen zum Cell State hinzuzufügen:

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Zum Schluss muss der Output erzeugt werden, der auf dem Cell State basiert. Die Sigmoid-Aktivierungsfunktion entscheidet, welche Teile des Cell States ausgegeben werden:

$$o_t = \sigma(W_o * [h_{t-1}, x_t] + b_o)$$

Um Werte zwischen -1 und 1 zu erhalten, wird der Cell State durch eine Tanh-Aktivierungsfunktion gegeben und danach mit dem Output der Sigmoid-Aktivierungsfunktion multipliziert, damit nur der gewünschte Output ausgegeben wird:

$$h_t = o_t * tanh(C_t)$$

2.6. Convolutional Neural Network

Convolutional Neural Networks (CNN) welche Bekanntheit erlangt haben durch ihre Stärke im Umgang mit Bildern, wie zum Beispiel bei der ImageNet Competition [13], werden mittlerweile in verschiedensten Anwendungsbereichen genutzt. Besonders dort, wo viele Input Features vorhanden sind, da CNNs besonders gut damit umgehen können, in dem sie schon von ihrer Grundstruktur her wenig Verbindungen zwischen den Layer erstellen. Der Name kommt von der mathematischen Operation der Faltung (Convolution), dabei werden die Filter, welche die zu trainierenden Parameter sind, mit den Daten in den jeweiligen Layern gefaltet [2].

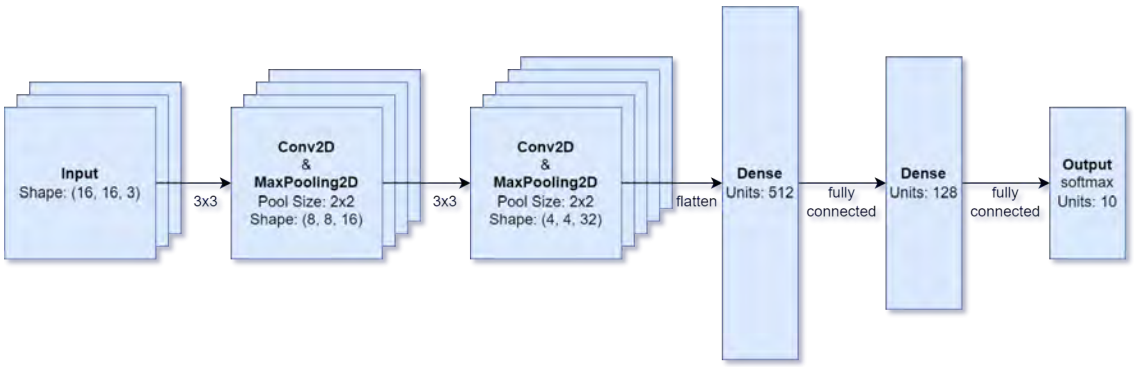


Abbildung 2.9.: Architektur eines einfachen CNNs das ein Bild mit einer Auflösung von 16x16 Pixel und drei Farbkanälen klassifiziert

In der Abbildung 2.9 ist die Architektur eines einfachen CNNs zu sehen. Das Bild kommt mit den drei Farbkanälen hinein, geht durch den ersten Convolutional Layer mit 16 3x3-Filter, woraus 16 verschiedene Bilder resultieren. Diese werden mit Max Pooling in beiden Dimensionen halbiert, weshalb aus einem 2x2 Block ein Pixel wird. Beim Max Pooling wird jeweils der höchste Wert übernommen (Abbildung 2.10). Dies wiederholt sich noch einmal mit 32 Filtern und danach gibt es einen Übergang in ein herkömmliches Fully Connected Neural Network mit einem entsprechenden Softmax Output Layer am Ende. Welcher für das Bild jede der zehn Klassen mit einer Wahrscheinlichkeit bewertet.

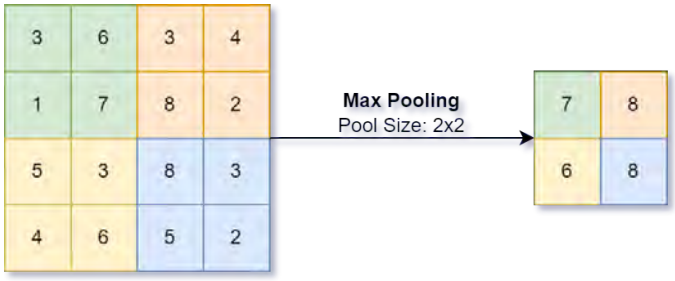


Abbildung 2.10.: Ein Beispiel von Max Pooling mit einer Pool Size von 2x2 Pixel. Dasselbe gibt es auch mit dem Mittelwert (Average Pooling) oder dem Minimum (Min Pooling).

2.7. Autoencoder

Autoencoder sind eine Art von neuronalen Netzen, die zur Erkennung von Mustern in Daten durch Unsupervised Learning eingesetzt werden und diese Daten in einer niedrigeren Dimension darstellen. Eine Besonderheit des Autoencoders liegt darin, dass der Input und der Output Layer jeweils die gleiche Dimension haben und dieselbe Menge an Information enthalten. Dies ist darauf zurückzuführen, dass der Autoencoder versucht, die Eingabedaten zu reproduzieren und diese Kopie der Eingabedaten nach einem Durchlauf durch das Netzwerk wieder ausgibt. Dazu bildet das Netzwerk jedoch nicht einfach den Input direkt auf den Output ab, sondern versucht, die Struktur und die wichtigsten Features der Daten zu lernen. Ein entsprechender Autoencoder ist in der Abbildung 2.11 ersichtlich. Der Autoencoder besteht

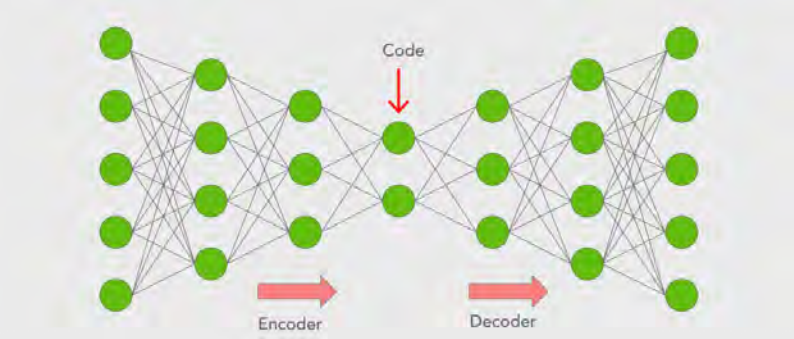


Abbildung 2.11.: Architektur eines undercomplete Autoencoder [23]

im Wesentlichen aus drei Komponenten: Der erste Bestandteil eines Autoencoders ist der Encoder, der die Eingabedaten entgegen nimmt und sie komprimiert, um sie in einer kleineren Dimension darzustellen. Der Code Layer ist der zweite Teil und der Flaschenhals des Autoencoders, welcher nur die wichtigsten Informationen der Daten enthält, um die ursprünglichen Daten rekonstruieren zu können. Das Ziel des Code Layers ist es, zu bestimmen, welche Daten für die Rekonstruktion erhalten bleiben müssen und welche verworfen werden können. Der letzte Bestandteil ist der Decoder, in dem die komprimierten Daten aus dem Code Layer entgegengenommen werden und daraus versucht wird, die ursprünglichen Daten zu rekonstruieren. Wie von Chen u. a. [5] beschrieben, wird der Rekonstruktionsfehler in einem Autoencoder minimiert, um die optimale Rekonstruktion der Daten zu erreichen. Dieser Rekonstruktionsfehler kann beispielsweise auf wie folgt berechnet werden:

$$\epsilon(x_i, x_i') = \sum_{i=1}^{d} (x_i - x_i')^2$$

Die Trainingsdaten umfassen die folgenden Daten $\{x_1, x_2, , ..., x_n\}$, von denen jeder ein d-dimensionaler Vektor darstellt. Daraus erzeugt der Autoencoder den folgenden Output $\{x_1', x_2', ..., x_n'\}$. Anschliessend wird die Differenz zwischen Input und Output gebildet, welche danach quadriert und für alle Werte aufsummiert wird.

Je nach Anwendung gibt es verschiedene Architekturen von Autoencodern. Bei der vorangegangenen Erklärung von Autoencodern handelt es sich um einen undercomplete Autoencoder. Wie bereits erläutert, zeichnet sich dieser durch die Komprimierung der Daten im Code Layer aus. Dieses Verfahren kann zum Beispiel zur Entfernung von Störungen in den Daten genutzt werden, da diese beim Komprimieren verloren gehen und somit im rekonstruierten Output nicht mehr vorhanden sind. Im Gegensatz dazu gibt es noch overcomplete Autoencoder. Hier geschieht im Grunde genau das Gegenteil und der Code Layer hat dieselbe Dimension oder wird sogar vergrössert. Dieser wird aber nicht mehr weiter erläutert, da er für unsere Arbeit nicht relevant ist.

2.8. Openstack Cluster der ZHAW

Da für das Trainieren des neuronalen Netzes in einer sinnvollen Zeit viel Rechenleistung benötigt wird, stellt die ZHAW für solche Projekte die Ressourcen ihres Openstack Clusters zur Verfügung [28]. Für das Trainieren der neuronalen Netze konfigurierten wir unsere virtuellen Maschinen mit 8 VCPUs, 16 GB RAM und einer Nvidia T4 Grafikkarte. Das effektive Trainieren dauerte nie mehr als 4 Stunden, der CNN-Autoencoder ist innerhalb von wenigen Minuten und der LSTM-Autoencoder innerhalb von drei Stunden trainiert.

2.9. Eingesetzte Software

Die Datenanalyse sowie auch die Implementation der neuronalen Netze wurden in Python umgesetzt mit den beiden Software-Bibliotheken Keras und TensorFlow. Damit die zur Verfügung gestellten Server-Ressourcen gut genutzt werden konnten, arbeiteten wir mit Jupyter Notebooks auf den virtuellen Maschinen. Die C3D-Dateien vom Motion Capture System haben wir mit der EZC3D-Bibliothek für Python ausgelesen [15].

3. Vorgehen

Die Gliederung des Projekts umfasst mehrere Phasen. Der erste Schritt war die Literaturrecherche, die dazu bestimmt ist, Wissen und Verständnis für die Domäne aufzubauen. Darauf folgt in diesem Kapitel die explorative Datenanalyse, bei der die Daten strukturiert und aufbereitet wurden. Anschliessend wurde auf der Grundlage der Literaturrecherche die Baseline erstellt. In einem zweiten Schritt haben wir eine optimierte Variante implementiert.

3.1. Datenerfassung

Die Datenerfassung wurde vom Insititut für Physiotherapie der ZHAW zwischen November 2021 und Juni 2022 durchgeführt. Es wurden Daten von 20 Probandinnen und Probanden erhoben, die ein 25-minütiges Training im ExerCube absolvierten. Zusätzlich zu den Trackern am Hand- und Fussgelenk für das Spiel, trugen sie die Marker, die zur Aufzeichnung der Bewegungen dienten. Diese sind in Abbildung 3.1 als weisse Punkte zu erkennen. An den Hüften, den Oberschenkeln und den Unterschenkeln ist je

Abbildung 3.1.: Ein Proband während eines Trainings im ExerCube mit den angebrachten Markern für das Motion Tracking System

eine Platte mit jeweils 4 Markern angebracht. Ausserdem befinden sich an beiden Schuhen jeweils 5 Marker von denen vier hinten am Schuh und einer an der Fussspitze angebracht ist. Diese Datenpunkte wurden mit einer Abtastfrequenz von 240 Hz mittels dem Motion Capture Systems Vantage der Firma Vicon aufgezeichnet [25]. Dabei beinhaltet ein Sample auf der X-, Y- und Z-Achse jeweils einen Wert für jeden Marker.

Um die Daten auswerten zu können haben wir ein Array mit allen verfügbaren Daten erstellt, das die folgenden Dimensionen hat:

$$D = (20, 84, 360‘000)$$

Die erste Dimension bezieht sich auf die Anzahl der Probanden, die zweite auf die Anzahl der Marker und ihre drei Achsen (28 Marker x 3 Achsen = 84 Feature) und die vierte auf die Anzahl der Frames (25 Minuten x 60 Sekunden x 240 Hz = 360'000).

3.2. Explorative Datenanalyse

3.2.1. Datenbereinigung

In den Rohdaten sind einige Lücken vorhanden, was darauf zurückzuführen ist, dass zu gewissen Zeitpunkten zu wenige Kameras gleichzeitig oder gar keine Kamera einen Marker sehen konnte. Eine mögliche Erklärung dafür ist, dass der entsprechende Marker von der Hand oder einem anderen Körperteil verdeckt wurde und die Position zum entsprechenden Zeitpunkt nicht bestimmt werden konnte. In solch einem Fall werden die X-, Y- und Z-Koordinaten dieses Markers auf NaN (Not a Number) gesetzt. Um einen Gesamtüberblick zu gewinnen, sind sämtliche NaN-Werte der 20 Probanden in der Abbildung 3.2 visualisiert.

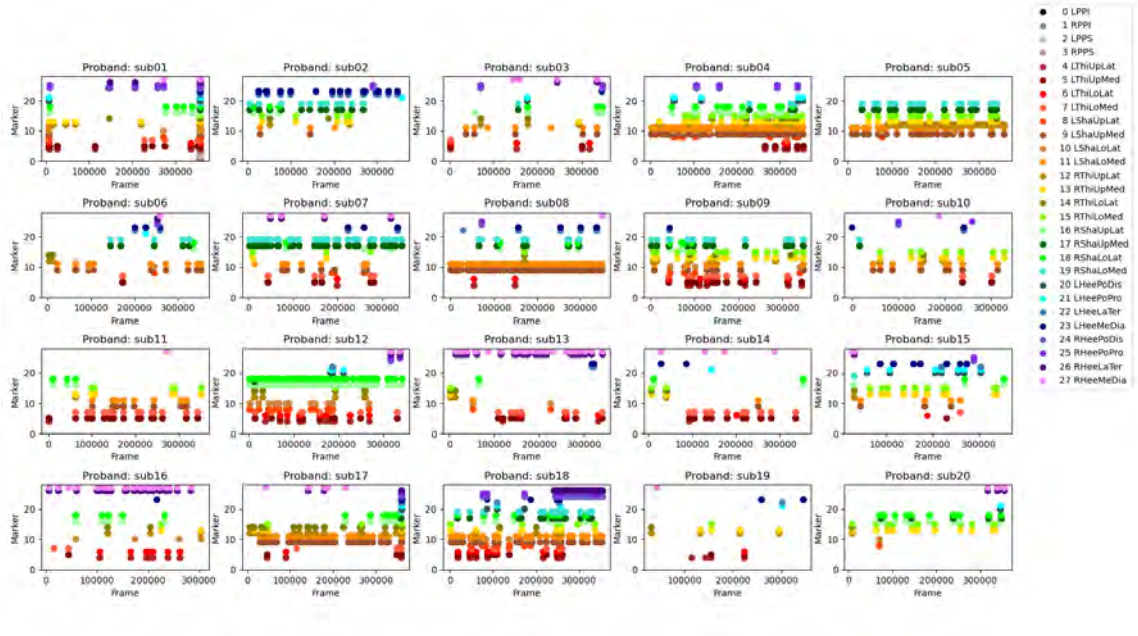


Abbildung 3.2.: Visualisierung der NaN-Werte über alle Probanden, jeden Marker und jedes Frame

Die verschiedenen Farben stehen für die jeweiligen Marker, wie aus der Legende hervorgeht. Diese Marker sind auf der Y-Achse abgebildet. Die X-Achse spiegelt die einzelnen Zeitschritte wider, die mit dem Motion Capture System aufgezeichnet wurden. Anhand der Visualisierung ist bereits zu erkennen, dass bei jedem Probanden Marker fehlen. Ausserdem ist ersichtlich, dass die Anzahl der fehlenden Marker zwischen den einzelnen Probanden variiert, wie zwischen den Probanden 18 und 19 gut zu sehen ist.

3.2.2. Entfernte Marker

Bei vier Probanden fielen während des Trainings einer oder beide Marker an den Fussspitzen ab. Von diesem Zeitpunkt an gibt es für den Rest des Trainings keine Daten mehr für die Fussspitzen. Aus diesem Grund wurden diese komplett für alle Probanden entfernt und im weiteren Verlauf der Arbeit nicht mehr beachtet. In der Abbildung 3.2 wurden diese bereits entfernt und sind in der Grafik nicht enthalten.

3.2.3. Rekonstruktion

Ohne die Marker auf den Fussspitzen sind nun alle Marker auf einer Platte (oder am Schuh) fest montiert. Pro Platte sind es jeweils vier Marker. In der Hälfte aller Frames auf denen ein Marker fehlt, sind die anderen drei Marker auf derselben Platte vorhanden. Somit kann die Position des fehlenden Markers, aufgrund der festen Position relativ zu den anderen Marker auf derselben Platte, mittels eines Referenz Frames rekonstruiert werden. Durch dieses Verfahren können rund die Hälfte aller NaN-Werte mit den korrekten Werten befüllt werden.

3.2.4. Restliche Lücken

Natürlich ist es möglich, die restlichen Lücken mit NaN-Werten in den Zeitreihen zu interpolieren. Aus Zeitgründen wurde darauf verzichtet und ein einfacherer Ansatz für den Umgang mit den fehlenden Werten gewählt. Pratama u. a. [19] haben drei konventionelle Möglichkeiten zum Umgang mit fehlenden Werten in Daten aus Zeitreihen beschrieben. Das Löschen der Daten ist für unsere Anwendung nicht geeignet, da entweder ein Grossteil der Marker entfernt oder der Verlauf einer Zeitreihe verändert würde. Ansonsten gibt es noch die Möglichkeiten, den Mittelwert für die fehlenden Werte einzufügen oder sie zu ignorieren. Wir haben uns dafür entschieden, für jeden NaN-Wert den Mittelwert einzusetzen, da dies die besten Ergebnisse lieferte. Der Mittelwert wird jeweils für jede Kombination von Marker und Achse separat berechnet und eingesetzt.

3.3. Aufbereitung der Trainingsdaten

3.3.1. Downsampling

Das markerbasierte Trackingsystem stellt mit einer Framerate von 240 Hz eine sehr hohe Auflösung zur Verfügung. Diese wird benötigt, damit die Marker von Frame zu Frame auch bei schnellen Bewegungen eindeutig verfolgt werden können. Für das neuronale Netz bedeutet die hohe Framerate jedoch viel Rechenaufwand beim Training und speziell bei der Verwendung von LSTM-Zellen, dass das zeitliche Gedächtnis der Zellen mit fast gleichen Daten überfüllt wird. Daher wird die Framerate von 240 Hz mit dem Faktor $\frac{1}{5}$ auf 48 Hz reduziert. Diese Frequenz sollte durchaus ausreichen, um auch schnelle Bewegungen zu erfassen und daraus Rückschlüsse auf die Müdigkeit zu ziehen, was von Dr. Eveline Graf bestätigt wurde.

3.3.2. Sliding Window

Die Trainingsdaten werden mittels eines Sliding Windows aufbereitet, die Sliding Windows sind 9 Sekunden lang und überlappen sich jeweils um 3.6 Sekunden. Diese Werte stammen aus der sehr ähnlich aufgebauten Arbeit von Balaskas und Siozios [3], wo sie sich als gut erwiesen haben. Ein einzelnes Trainingssample beinhaltet somit, bei einer Framerate von 48 Hz, 432 aufeinander folgende Frames. Jedes dieser Frames hat pro Marker drei Werte, für jede Dimension einen. Dies ergibt mit den 28 Markern, 84 Features pro Frame. Der Aufbau eines Trainingssamples ist in der Abbildung 3.3 skizziert.

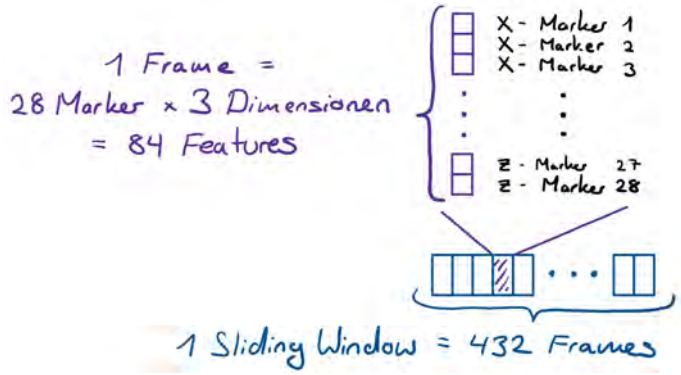


Abbildung 3.3.: Aufbau eines Trainingssample

3.3.3. Normalisierung

Für die Normalisierung wird auf dem Trainingset pro Achse von jedem Marker der Mittelwert und die Varianz berechnet und mit diesen dann die Trainings-, sowie auch die Validation- und Testdaten normalisiert. Dadurch funktioniert der Optimierungsalgorithmus Gradient Descent besser.

3.3.4. Train/Test/Validation Aufteilung

Die Trainingsdaten werden wie in der Tabelle 3.1 dokumentiert, in ein Trainings-, Test- und ein Validationset aufgeteilt. Die Aufteilung geschieht auf der Ebene Proband und nicht auf Trainingsamples, weil geprüft werden soll, ob das Modell nicht nur auf neuen Samples, sondern auch auf Samples von neuen Probanden gleich gut funktioniert.

Set	**Probanden**
Training	1 - 12
Validation	13 - 16
Test	17 - 20

Tabelle 3.1.: Train/Validation/Test Aufteilung

3.4. Baseline LSTM-Autoencoder

Die Arbeit von Balaskas und Siozios im Paper "Fatigue Detection Using Deep Long Short Term Memory Autoencoders"[3] ist im Blick auf unsere Aufgabenstellung ein sehr interessantes Paper. Sie erkennen die Müdigkeit aufgrund von Bewegungsdaten. Ihre Bewegungsdaten enthalten ebenfalls keine Labels für die Müdigkeit, weshalb das Modell die Daten auch unsupervised lernen muss. Diese Daten wurden von 14 Probanden während jeweils 20 Minuten gesammelt, im Falle des ExerCubes sind es 25-minütige Trainings von 20 Probanden.

Dabei differenziert sich ihre Arbeit von unserer in dem ihre Daten von IMU-Sensoren kommen und die Trainings aus einem gleichmässigen Lauftraining mit repetitiven Bewegungen stammen. Beim ExerCube gibt es einige verschiedene Übungen, welche ganz unterschiedlich kombiniert werden, was grosse Variabilität in die Daten bringt.

Aufgrund der vielen Ähnlichkeiten haben wir uns entschieden, die Baseline anhand ihrer Arbeit aufzubauen: Einen undercomplete Autoencoder mit LSTM-Zellen.

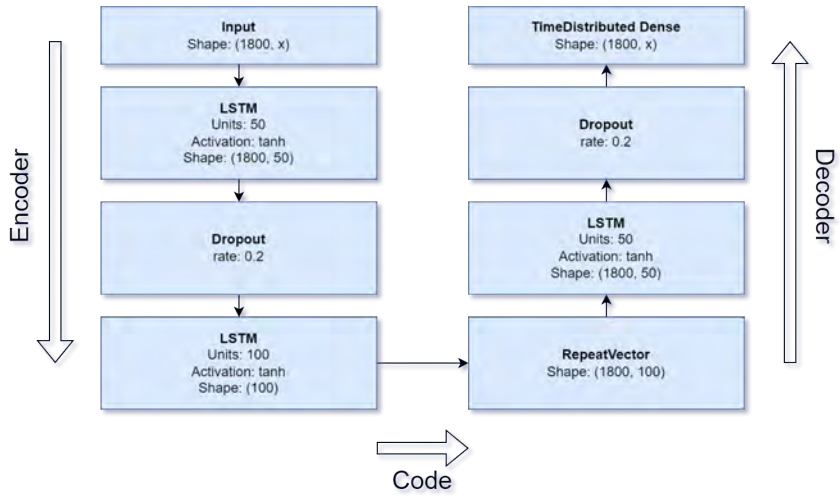


Abbildung 3.4.: Architektur des LSTM-Autoencoders von Balaskas und Siozios [3]

3.4.1. Implementation

Konstantinos Balaskas und Kostas Siozios waren so freundlich und haben uns ihren Python Code vom Aufbau ihres Autoencoders zur Verfügung gestellt. Auf diese Weise erhielten wir alle Details zu ihrem neuronalen Netz, auch solche, die in ihrem Paper nicht beschrieben sind. Das unveränderte Modell (Abbildung 3.4) ist nicht in der Lage, unsere Daten zu lernen. Der Grund dafür ist, dass unsere Daten eine wesentlich grössere Variabilität und mehr Dimensionen aufweisen.

Das Erweitern mit mehr Neuronen reicht aus, den LSTM-Autoencoder einem einzelnen Probanden anzupassen, jedoch nicht wenn er auf mehreren Probanden gleichzeitig trainiert wird. Die Anpassungen müssen dem Netz mehr Flexibilität geben, um unsere Daten zu erlernen. Wie von Pascanu u. a. [17] beschrieben, basiert Deep Learning auf der Annahme, dass tiefere Netze besser geeignet sind als flache Netze, um komplexere Funktionen zu erlernen. Beispielsweise wurde bereits von İrsoy u. a. [11] gezeigt, dass sich ihre Ergebnisse verbesserten, als sie von einem bidirektionalen RNN mit einem Layer zu einer Architektur mit mehreren Layern wechselten. Aufgrund dieser positiven Erfahrungen mit tieferen Netzen haben wir uns entschieden, die Architektur um zusätzliche Layer zu erweitern.

3.4.2. Architektur

Die hier beschriebene Architektur basiert noch auf dem Paper von Balaskas und Siozios [3] wurde aber um zusätzliche Layer und LSTM-Zellen erweitert. Encoder und Decoder wurden durch zusätzliche LSTM-Layer zu Deep Encoder und Decoder ergänzt. Die resultierende Architektur ist in der Abbildung 3.5 dargestellt.

Der Encoder besteht aus vier LSTM-Layer, die ihre Fähigkeiten nutzen, um die zeitlichen Zusammenhänge zwischen den Frames innerhalb der Sliding Windows zu erfassen. Der letzte Layer im Encoder übergibt die Daten flach ohne zeitliche Dimension als Code dem Decoder. Dieser besteht wiederum aus vier LSTM-Layern, um die Daten aus dem Code zu rekonstruieren.

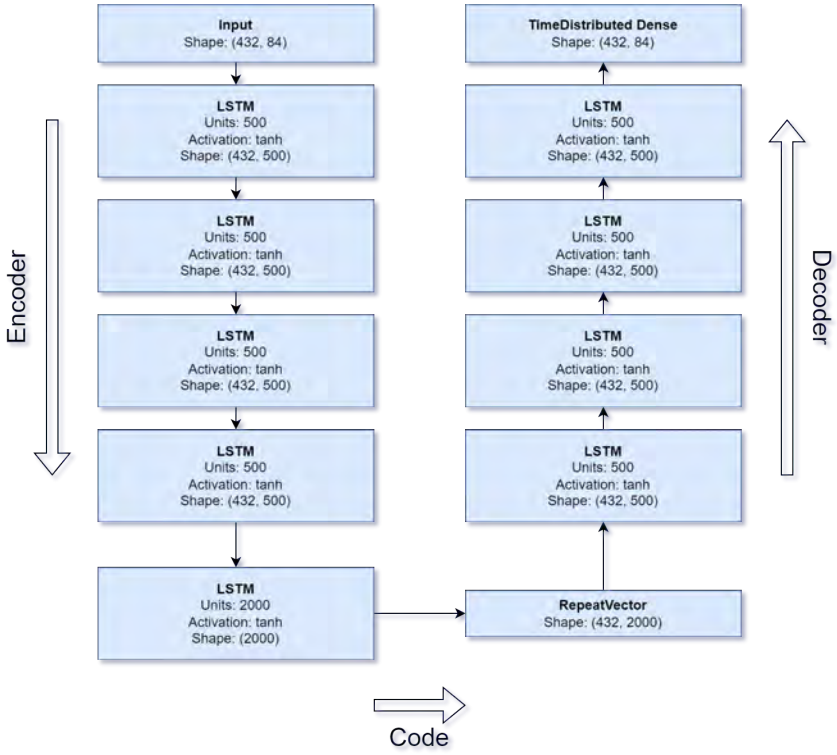


Abbildung 3.5.: Architektur des LSTM-Autoencoders

3.5. CNN-Autoencoder

Nach dem mässigen Erfolg mit dem LSTM-Autoencoder haben wir auf Hinweis unseres Betreuers Prof. Dr. Thilo Stadelmann einen zweiten Ansatz mit einem CNN-Autoencoder implementiert. Die CNNs haben von ihrer Grundstruktur her mit den Filtern nur wenige Verbindungen, beziehungsweise Parameter, zwischen den Layern. Deshalb können sie gut mit vielen Input Features, wie es auch bei unseren Daten der Fall ist, umgehen.

3.5.1. Implementation & Architektur

Als Starthilfe und Inspiration für die Architektur diente der Convolutional Autoencoder aus dem Keras Blogpost zu Autoencodern von Francois Chollet [6]. Die Aufbereitung der Daten erfolgt analog zum LSTM-Autoencoder, indem zuerst die Marker und ihre Achsen zusammengeführt und dann in den zwei Dimensionen Zeit x Marker (Abbildung 3.3) an den CNN-Autoencoder übergeben werden. Wie in der Abbildung 3.6 sichtbar, hat der Encoder mehr Layer als der Decoder, damit der Autoencoder nicht einfach für jedes Sample einen separaten Code lernt und nichts über die Daten selbst in Erfahrung bringt. Die Daten werden im Encoder nur in der Dimension der Zeit zusammengestaucht, da zwischen den Markern, welche nebeneinander liegen, keine starken Zusammenhänge bestehen. Für das Downsampling wird der Average Pooling Layer verwendet, da die Werte sich kontinuierlich über die Zeit verändern und durch das Mitteln der Werte weniger Information verloren geht. Aus demselben Grund wurde im Decoder der UpSampling Layer mit bilinearer Interpolation verwendet. So werden die kontinuierlichen Werte am besten und viel schneller als mit der bicubic Interpolation angenähert.

Die Anzahl der Layer von Encoder und Decoder, sowie die Anzahl der Filter pro Layer wurden heuristisch evaluiert. Werden die Daten mit zusätzlichen Layern noch mehr komprimiert, wird die Rekonstruktion gesamthaft schlechter und somit das Endresultat auch nicht besser. Werden mehr Filter pro Layer eingesetzt, enthält das Netz mehr Parameter, es wird aber keine erheblich bessere Performance erreicht. Daher wurde die einfachere Architektur mit weniger Parametern ausgewählt.

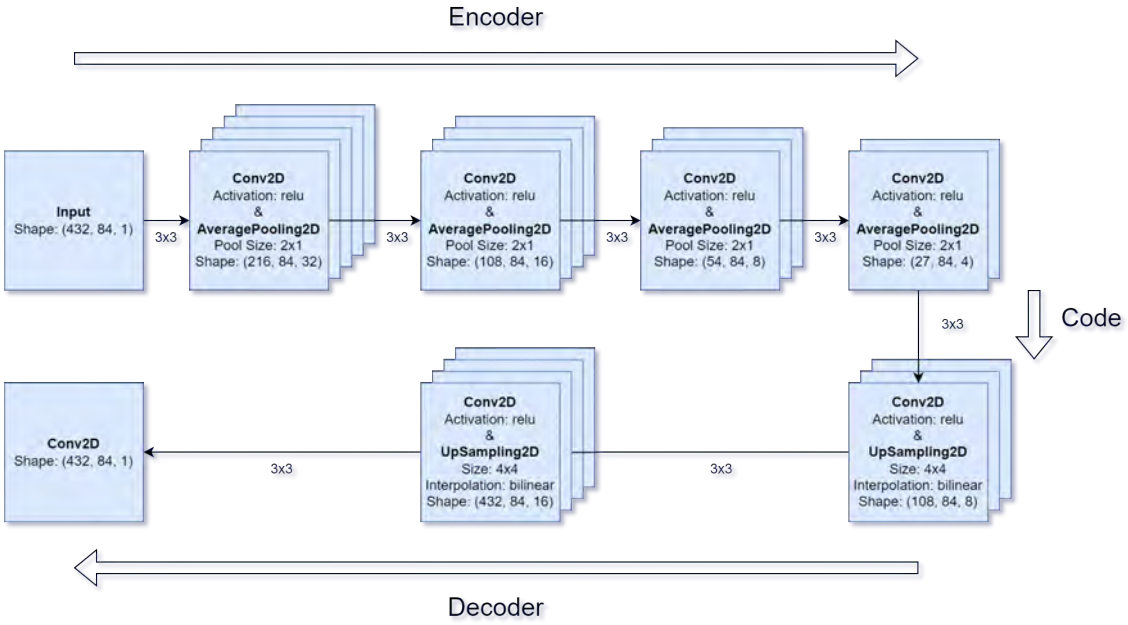


Abbildung 3.6.: Architektur des CNN-Autoencoders

Die Implementation dieses Autoencoders war im Verhältnis zum LSTM-Autoencoder einfacher, da es besser verständlich ist, was genau im neuronalen Netz abläuft. Dadurch, dass er mit rund 8'000 Parametern um ein Faktor 4'800 weniger Parameter braucht, ist er dementsprechend auch viel schneller beim Trainieren.

4. Resultate

4.1. LSTM-Autoencoder

Der Rekonstruktionsfehler des LSTM-Autoencoders beträgt am Ende des Trainings auf dem Trainingsset 0.248 und auf dem Validationset 0.794. Somit overfittet das Modell auf dem Trainingsset, dabei sehen die rekonstruierten Bewegungen in der Abbildung 4.1 nur bei langsamen Bewegungen gut aus. Sobald es sich um Marker und Übungen handelt, die schnelle Bewegungen enthalten, ist der Autoencoder überfordert und kann nur noch mit grossen Abweichungen rekonstruieren. Dementsprechend sehen die Stichproben aus dem Validationset (Abbildung 4.2) nochmals einiges schlechter aus. Bereits langsame Bewegungen werden mit einem grossen Fehler rekonstruiert.

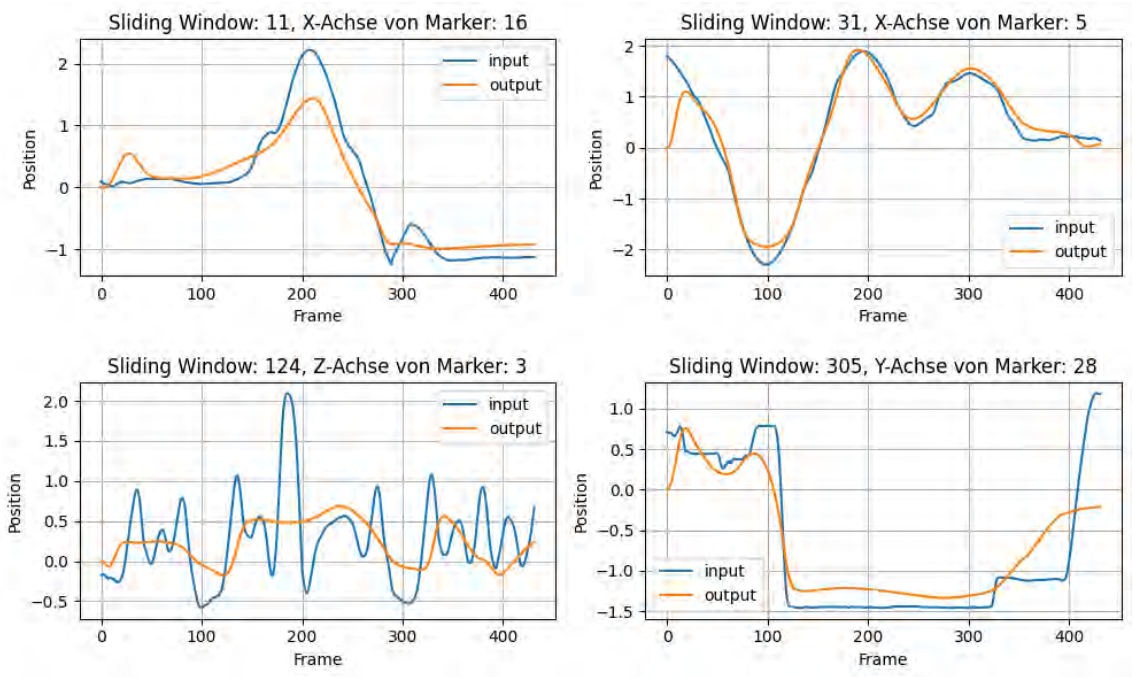


Abbildung 4.1.: Stichproben aus dem Trainingsset mit der jeweiligen Ausgabe aus dem LSTM-Autoencoder

In der Abbildung 4.3 ist die Endauswertung des LSTM-Autoencoders zu sehen mit dem Rekonstruktionsfehler über das gesamte 25-minütige Training einzelner Probanden. Im Anfangsbereich der Probanden aus dem Trainingsset (Abbildung 4.3a) ist der Bereich, auf welchem der Autoencoder trainiert wurde, rot hinterlegt. In diesem Bereich ist der Fehler klein und steigt danach sprunghaft an. Je nach Probanden geht der Fehler im Verlauf des Trainings leicht hoch, bei anderen etwas runter oder bleibt auf einem konstanten Niveau. Eine Müdigkeit, die bei allen Probanden im Verlauf des Trainings den Rekonstruktionsfehler ansteigen lässt, ist nicht sichtbar. Bei den Probanden aus dem Validationset (Abbildung 4.3) ist, ausser im Anfangsbereich, dasselbe Verhalten des Rekonstruktionsfehler zu beobachten.

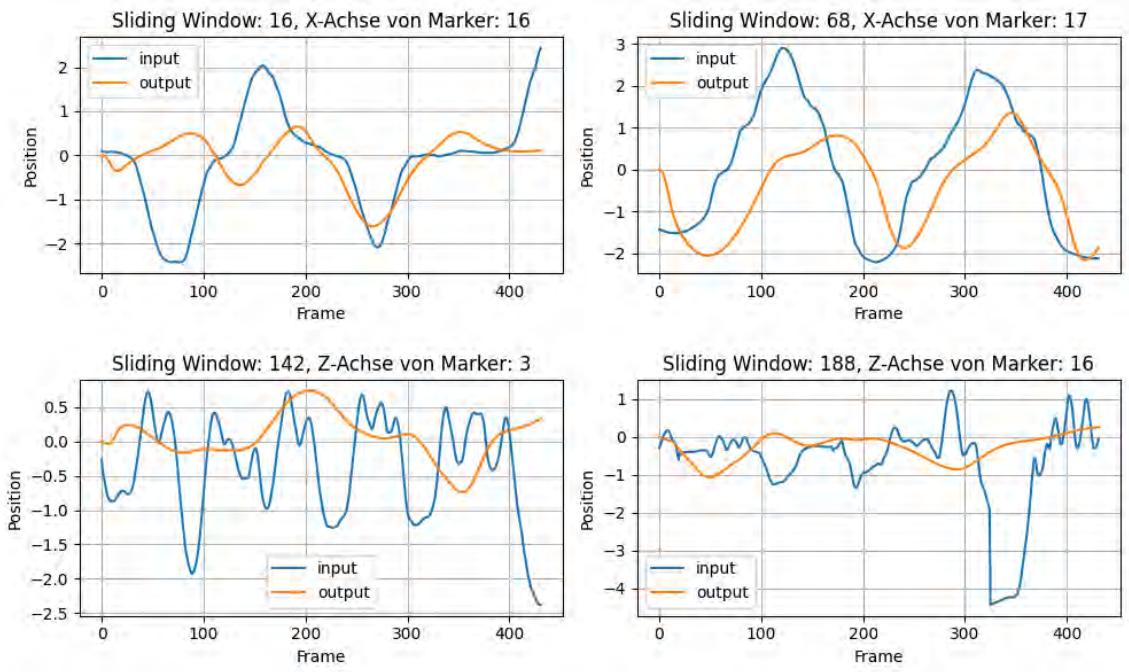
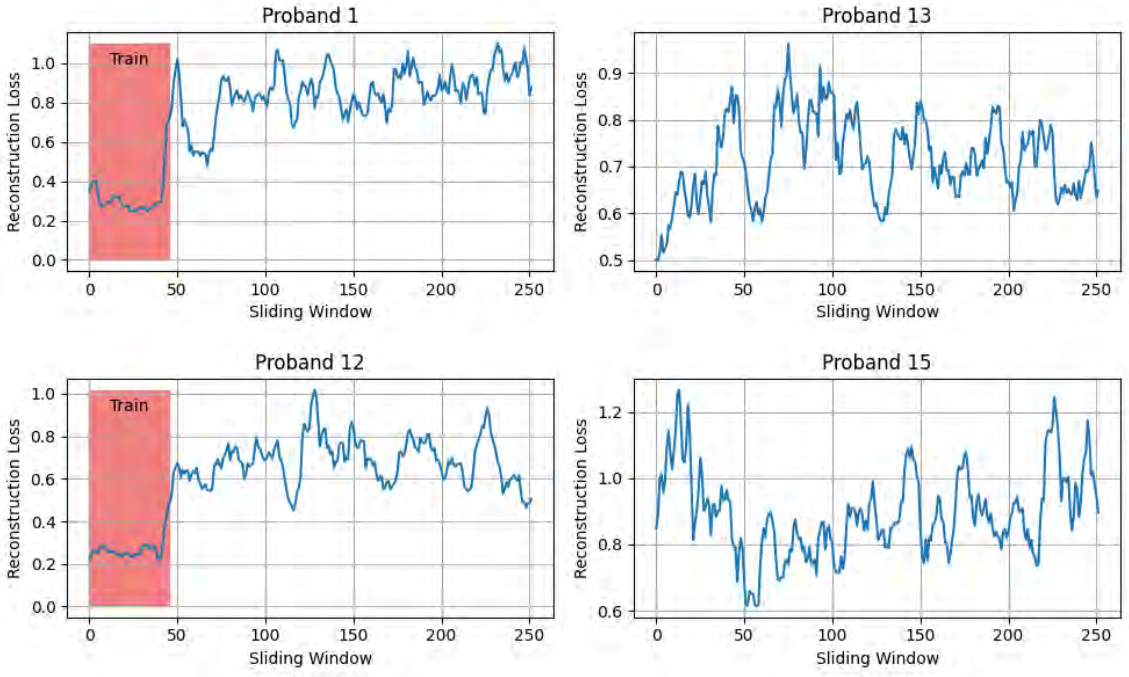


Abbildung 4.2.: Stichproben aus dem Validationset mit der jeweiligen Ausgabe aus dem LSTM-Autoencoder



(a) Probanden aus dem Trainingset (b) Probanden aus dem Validationset

Abbildung 4.3.: Auswertung des Rekonstruktionsfehlers über das ganze Training eines einzelnen Probanden mit dem LSTM-Autoencoder

4.2. CNN-Autoencoder

Der CNN-Autoencoder erreicht auf dem Trainingsset einen Rekonstruktionsfehler von 0.012 und auf dem Validationset 0.022, das zeigt bereits, dass der Autoencoder gut generalisiert. Die Analyse der Stichproben einzelner Sliding Windows auf dem Trainingsset (Abbildung 4.4) zeigt, dass nicht nur langsame, sondern auch schnelle Bewegungen gut rekonstruiert werden.

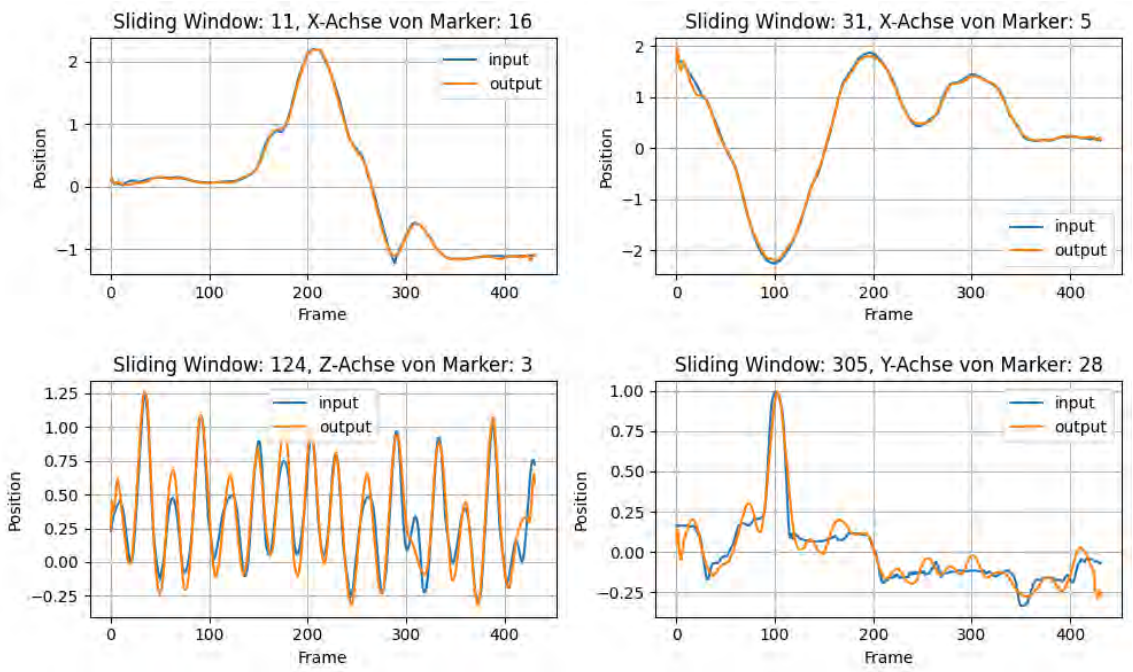


Abbildung 4.4.: Stichproben aus dem Trainingsset mit der jeweiligen Ausgabe aus dem CNN-Autoencoder

Auch auf dem Validationset können die Stichproben gut rekonstruiert werden (Abbildung 4.5), viel besser als es der LSTM-Autoencoder kann (Abbildung 4.2). Diese Fähigkeit, auch eine gute Leistung auf dem Validationset zu erbringen, zeigt, dass die Muster der Bewegungen gelernt wurden und nicht nur die Trainingsdaten. Somit wurde ein gutes Mass zwischen Varianz und Bias erreicht.

Die Auswertung des Rekonstruktionsfehlers auf den ganzen Trainingssequenzen einzelner Probanden ist in der Abbildung 4.6 abgebildet. In der Abbildung 4.6a sind zwei Probanden aus dem Testset ausgewertet und in der Abbildung 4.6b zwei aus dem Validationset. Auf der Auswertung vom Proband 1 ist ein leichter Aufwärtstrend im Verlauf der Zeit sichtbar, dies zeigt sich jedoch nicht in den Auswertungen der anderen Probanden, somit kann auch hier keine Müdigkeit durch die Auswertung des Rekonstruktionsfehler festgestellt werden.

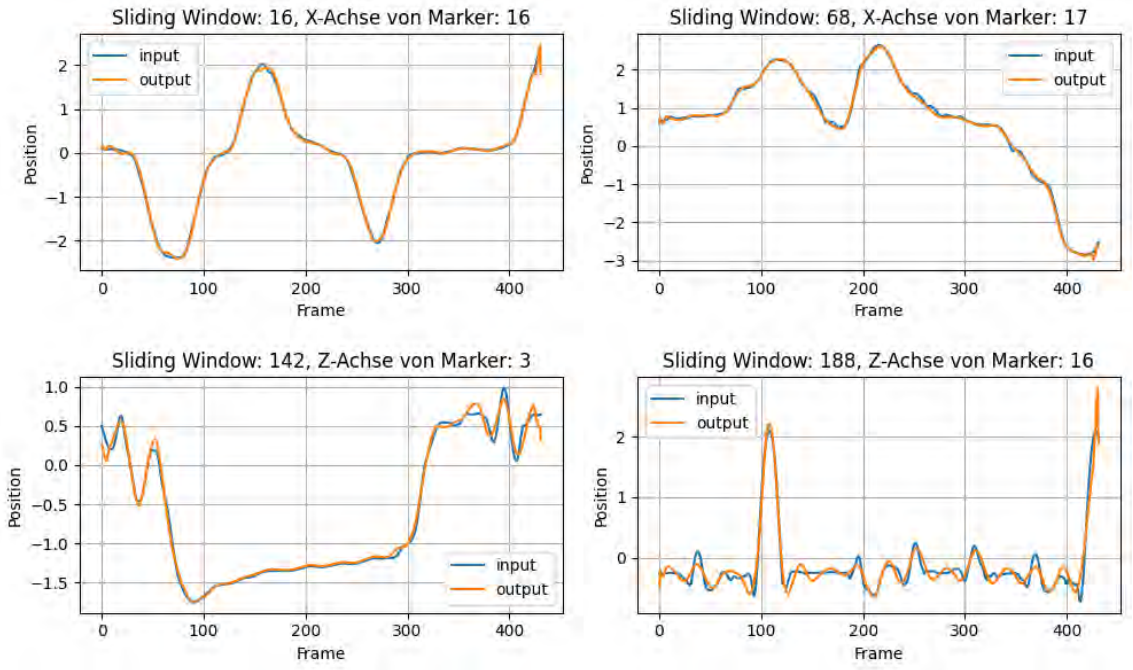
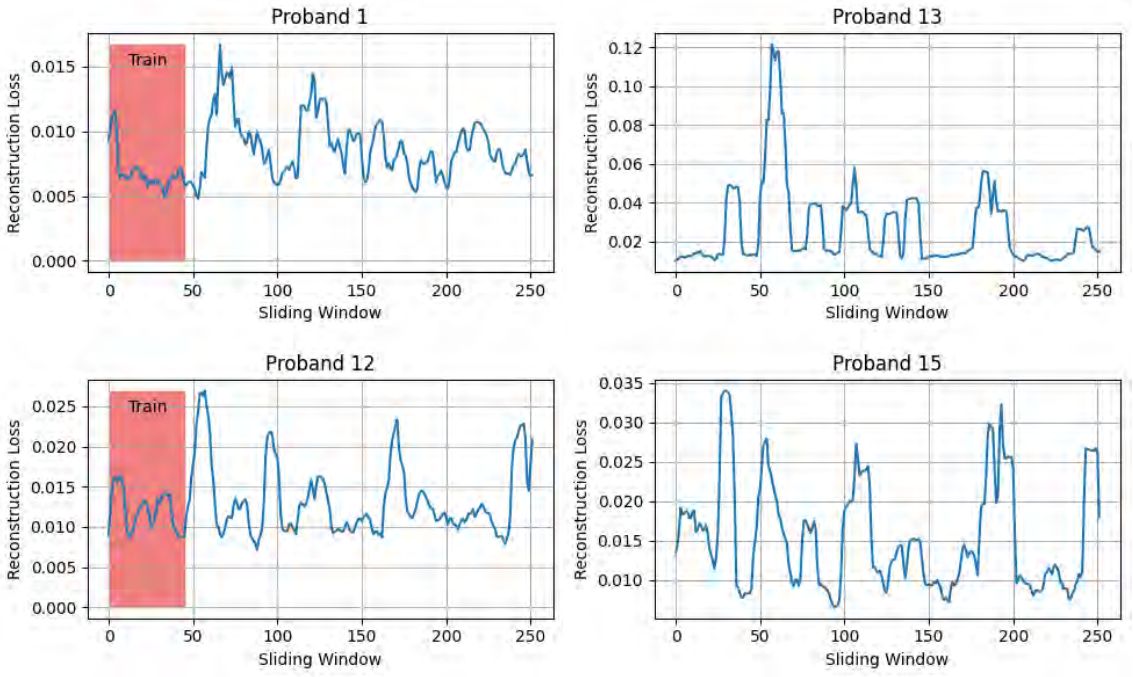


Abbildung 4.5.: Stichproben aus dem Validationset mit der jeweiligen Ausgabe aus dem CNN-Autoencoder



(a) Probanden aus dem Trainingset (b) Probanden aus dem Validationset

Abbildung 4.6.: Auswertung des Rekonstruktionsfehlers über das ganze Training eines einzelnen Probanden mit dem CNN-Autoencoder

5. Diskussion

In dieser Arbeit ist es nicht gelungen, die Ermüdung der Athleten anhand der Bewegungsdaten zu erkennen. Dennoch bildet sie eine gute Grundlage für weitere Versuche mit demselben Ziel. Ebenso lassen sich daraus zusätzliche Erkenntnisse in Bezug auf maschinelles Lernen mit Bewegungsdaten gewinnen. Da die Arbeit mit dem RNN und dem CNN in zwei Teile gegliedert ist, gibt es für beide Architekturen ein eigenes detailliertes Fazit.

5.1. Diskussion der LSTM-Architektur

Während des Trainings ist ein Rückgang des Rekonstruktionsfehlers zu beobachten, was bedeutet, dass sich das LSTM-Netz an die Daten anpasst. Allerdings lässt sich dies nur schwer skalieren, weil bei einem Anstieg der Datenmenge auch das Netz vergrössert werden muss. Dies zeigt sich daran, dass eine Architektur, die an einem Probanden funktioniert, bei mehreren Probanden nur einen konstanten Wert ausgibt und versucht, diesen zu optimieren. Das in dieser Arbeit verwendete Modell ist mit über 38 Millionen Parametern bereits sehr komplex, was auch das Training verlangsamt. Trotzdem hatte das Modell Probleme, die Daten gut zu erlernen. Die Schwierigkeiten lassen sich wohl auf die beschränkte Menge der Trainingsdaten von 12 Probanden und gleichzeitig die grosse Variabilität in den Daten zurückführen.

Eventuell war es keine gute Entscheidung den Encoder und Decoder gleich stark aufzubauen, da dadurch dem Autoencoder die Möglichkeit zur Verfügung steht, die Daten eins zu eins zu codieren und wieder zu decodieren. Dabei würde er nicht das eigentliche Bewegungsmuster lernen.

Um den Rekonstruktionsfehler weiter zu reduzieren, müsste das Modell vergrössert werden und die Anzahl der Parameter würde weiter zunehmen. Ein solch grosses Netzwerk wäre für diese Aufgabenstellung schlicht nicht mehr zielführend und auch nicht mehr nachvollziehbar. Aus diesem Grund wurde dieser Ansatz mit einem RNN nicht weiter verfolgt, auch aus dem Grund, dass in den Validierungsdaten keine Tendenz des Rekonstruktionsfehlers erkennbar ist.

5.2. Diskussion der CNN-Architektur

Das CNN-Netz scheint für diese Aufgabenstellung aus folgenden Gründen wesentlich besser geeignet zu sein. Zum einen hat das CNN-Netz mit rund 8'000 Parametern deutlich weniger als das LSTM-Netz. Es ist grundsätzlich ratsam, ein Netz zu verwenden, das für den jeweiligen Einsatzzweck so wenige Parameter wie möglich hat. Zweitens passt sich das Netz mit einem Rekonstruktionsfehler von 0,022 im Validierungsset, wie in Tabelle 5.1 ersichtlich, deutlich besser an die Trainingsdaten an. Das Modell lernt die Daten sehr gut zu rekonstruieren, wie in den Abbildungen 4.4 und 4.5 zu erkennen ist. Der gewünschte Effekt der Müdigkeitserkennung kann jedoch nicht erzielt werden.

Architektur	Trainingsset	Validierungsset	Testset
LSTM	0.249	0.795	0.677
CNN	0.013	0.022	0.020

Tabelle 5.1.: Rekonstruktionsfehler beider Architekturen im Vergleich

Zusammenfassend lässt sich sagen, dass sich das Modell mit der CNN-Architektur, unabhängig von unserer Aufgabe, für diese Art von Daten und Zielsetzungen als gute Methode erweist. Aufgrund des zeitlichen Rahmens dieser Projektarbeit waren wir nicht in der Lage, weitere Experimente durchzuführen. Deshalb ist das Resultat dieser Arbeit als inkonklusiv zu betrachten.

6. Ausblick

Wie in der Diskussion erwähnt, wurde das Ziel der Müdigkeiterkennung in den Bewegungsdaten des ExerCube nicht erreicht. Basierend auf den hier präsentierten Modellen und Verfahren können noch zahlreiche weitere Experimente durchgeführt und die daraus resultierenden Auswirkungen beobachtet werden. Die möglichen Experimente wurden in verschiedene Kategorien unterteilt, die als Kapitel aufgeführt sind. Im letzten Abschnitt werden ausserdem zwei vollkommen neue Ansätze vorgestellt, die während der Fertigstellung der Arbeit entstanden sind.

6.1. Window Size und Overlap

Da unsere Arbeit an diejenige von Balaskas und Siozios [3] anlehnt, liegt es auf der Hand, dieselben Experimente wie sie durchzuführen. Zum einen haben sie ein Experiment durchgeführt, bei dem sie die Grösse des Sliding Window verändert haben. Dies wäre ebenfalls sinnvoll, da in unserem Sliding Window von 9 Sekunden mehrere Übungen ausgeführt werden. Der Hintergedanke ist, dass bei einer kleineren Grösse des Sliding Window mit einer einzigen Übung das Modell in der Lage sein könnte, sich mehr auf die Ausführung dieser Übung zu konzentrieren. Zum anderen führten sie ein Experiment durch, bei dem sie die Überlappung des Sliding Window zwischen 0 Sekunden und 7 Sekunden variierten. Dies könnte interessant sein, da in ihrem Experiment signifikante Unterschiede zwischen den Überlappungsdauern festgestellt wurden.

6.2. Freezing Layers

Zu diesem Zeitpunkt wird das Modell auf allen Probanden trainiert, unabhängig davon, auf welchen Probanden es angewendet wird. Eine Möglichkeit besteht darin, das Modell auf einen bestimmten Athleten zu optimieren. Zuerst würde das Modell auf allen Probanden trainiert werden, allerdings nur so weit, bis es sich die allgemeinen Merkmale der Bewegungsdaten erlernt hat, die für alle Sportler gleich sind. Sobald es um spezifische Merkmale des Testprobandes geht, würden die unteren Layer eingefroren. Danach werden nur noch die oberen Schichten des Modells auf einzelnen Probanden trainiert. Der Grundgedanke dahinter ist, dass die unteren Layer des Modells die allgemeinen Eigenschaften der Übungen erfassen, während die probandenspezifischen Eigenschaften auf den oberen Layer erfasst werden.

6.3. CNN-Architektur

Es besteht die Möglichkeit, weitere Parameter der CNN-Architektur heuristisch zu evaluieren. Das Modell passt bereits sehr gut zu den Daten und es geht nicht darum, die Daten noch besser abbilden zu können. Vielmehr geht es darum, mit welchen Parametern ein Unterschied zwischen den Müdigkeitsstufen festgestellt werden kann. So können zum Beispiel verschiedene Aktivierungsfunktionen ausprobiert werden. Es besteht auch die Möglichkeit, den Encoder mit weniger Layer und den Decoder stattdessen mit mehr Layern auszustatten und zu sehen, ob dies einen Unterschied macht.

6.3.1. Separate Kanäle

Aktuell werden alle Marker und ihre Achsen in einer einzelnen Matrix dem CNN-Autoencoder übergeben. Dabei könnte es von Vorteil sein, die drei Achsen X, Y und Z als separate Kanäle anzusehen und wie bei einem Farbbild in der Form von drei Matrizen pro Sample zu gruppieren. Durch das könnte der Autoencoder eventuell die Zusammenhänge der Achsen besser erlernen.

6.4. Supervised Learning

Ein ganz anderer Ansatz besteht darin, die Aufgabe zu einem Supervised Learning Problem zu machen. Der Grund dafür ist, dass das Modell nur auf den ersten fünf Minuten trainiert wird und der Grossteil der Daten eines Probanden, nicht verwendet werden können. Eine Möglichkeit bestünde darin, als Output des Modells den Zeitpunkt der Übung vorherzusagen. Beim Training würde wieder ein Sliding Window als Eingabe verwendet werden und als Output würde das Modell den Zeitpunkt der Übung in Sekunden erhalten. Daraus sollte das Modell das Mapping der Übung auf den jeweiligen Zeitpunkt lernen. Dies beruht auf der Annahme, dass die Müdigkeit im Verlauf des Trainings stetig ansteigt. Die Bewertung des Modells kann dann mit den Methoden des Supervised Learning vollzogen werden.

Ein zweiter möglicher Ansatz im Supervised Learning wäre, zuerst ein Modell für die Detektion der Übungen zu erstellen und danach auf diesen Übungen wieder den hier behandelten Autoencoder Ansatz anzuwenden. Durch das Separieren der Übungen wird ein Grossteil der Varianz wegfallen, der aktuell in den Daten vorhanden ist und das Lernen mit der beschränkten Menge an Daten erschwert.

7. Verzeichnisse

Literaturverzeichnis

[1] Kota Aoki u. a. „Physical Fatigue Detection From Gait Cycles via a Multi-Task Recurrent Neural Network“. In: *IEEE Access* 9 (2021). Conference Name: IEEE Access, S. 127565–127575. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2021.3110841.

[2] Divya Arora, Mehak Garg und Megha Gupta. „Diving deep in Deep Convolutional Neural Network“. In: *2020 2nd International Conference on Advances in Computing, Communication Control and Networking (ICACCCN)*. 2020 2nd International Conference on Advances in Computing, Communication Control and Networking (ICACCCN). Dez. 2020, S. 749–751. DOI: 10.1109/ICACCCN51052.2020.9362907.

[3] Konstantinos Balaskas und Kostas Siozios. „Fatigue Detection Using Deep Long Short-Term Memory Autoencoders“. In: *2021 10th International Conference on Modern Circuits and Systems Technologies (MOCAST)*. 2021 10th International Conference on Modern Circuits and Systems Technologies (MOCAST). Juli 2021, S. 1–4. DOI: 10.1109/MOCAST52088.2021.9493378.

[4] Johannes Burdack u. a. „Systematic Comparison of the Influence of Different Data Preprocessing Methods on the Performance of Gait Classifications Using Machine Learning“. In: *Frontiers in Bioengineering and Biotechnology* 8 (2020). ISSN: 2296-4185. URL: https://www.frontiersin.org/articles/10.3389/fbioe.2020.00260 (besucht am 18. 10. 2022).

[5] Zhaomin Chen u. a. „Autoencoder-based network anomaly detection“. In: *2018 Wireless Telecommunications Symposium (WTS)*. 2018 Wireless Telecommunications Symposium (WTS). Apr. 2018, S. 1–5. DOI: 10.1109/WTS.2018.8363930.

[6] Francois Chollet. *Building Autoencoders in Keras*. The Keras Blog. 14. Mai 2016. URL: https://blog.keras.io/building-autoencoders-in-keras.html (besucht am 16. 12. 2022).

[7] Xiaole Guan u. a. „Sports fatigue detection based on deep learning“. In: *2021 14th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*. 2021 14th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI). Okt. 2021, S. 1–6. DOI: 10.1109/CISP-BMEI53629.2021.9624395.

[8] Sheldon J. Hawley, Andrew Hamilton-Wright und Steven L. Fischer. „Detecting subject-specific fatigue-related changes in lifting kinematics using a machine learning approach“. In: *Ergonomics* 0.0 (4. Apr. 2022). Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/00140139.2022.2061052, S. 1–12. ISSN: 0014-0139. DOI: 10.1080/00140139.2022.2061052. URL: https://doi.org/10.1080/00140139.2022.2061052 (besucht am 18. 10. 2022).

[9] Geovanni Hernandez u. a. „Machine Learning Techniques for Motion Analysis of Fatigue from Manual Material Handling Operations Using 3D Motion Capture Data“. In: *2020 10th Annual Computing and Communication Workshop and Conference (CCWC)*. 2020 10th Annual Computing and Communication Workshop and Conference (CCWC). Jan. 2020, S. 0300–0305. DOI: 10.1109/CCWC47524.2020.9031222.

[10] Sepp Hochreiter und Jürgen Schmidhuber. „Long Short-Term Memory“. In: *Neural Computation* 9.8 (Nov. 1997). Conference Name: Neural Computation, S. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735.

[11] Ozan İrsoy und Claire Cardie. „Opinion Mining with Deep Recurrent Neural Networks“. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. EMNLP 2014. Doha, Qatar: Association for Computational Linguistics, Okt. 2014, S. 720–728. DOI: 10.3115/v1/D14-1080. URL: https://aclanthology.org/D14-1080 (besucht am 17. 12. 2022).

[12] Anastasia V. Keller u. a. „Unsupervised Machine Learning on Motion Capture Data Uncovers Movement Strategies in Low Back Pain“. In: *Frontiers in Bioengineering and Biotechnology* 10 (2022), S. 868684. ISSN: 2296-4185. DOI: 10.3389/fbioe.2022.868684.

[13] Alex Krizhevsky, Ilya Sutskever und Geoffrey E. Hinton. „ImageNet classification with deep convolutional neural networks“. In: *Communications of the ACM* 60.6 (24. Mai 2017), S. 84–90. ISSN: 0001-0782. DOI: 10.1145/3065386. URL: https://doi.org/10.1145/3065386 (besucht am 16. 12. 2022).

[14] Erik Marchi u. a. „Non-linear prediction with LSTM recurrent neural networks for acoustic novelty detection“. In: *2015 International Joint Conference on Neural Networks (IJCNN)*. 2015 International Joint Conference on Neural Networks (IJCNN). ISSN: 2161-4407. Juli 2015, S. 1–7. DOI: 10.1109/IJCNN.2015.7280757.

[15] Benjamin Michaud und Mickaël Begon. „‘ezc3d‘: An easy C3D file I/O cross-platform solution for C++, Python and MATLAB“. In: *Journal of Open Source Software* 6.58 (21. Feb. 2021), S. 2911. ISSN: 2475-9066. DOI: 10.21105/joss.02911. URL: https://joss.theoj.org/papers/10.21105/joss.02911 (besucht am 20. 12. 2022).

[16] *Neural Networks, Types, and Functional Programming – colah's blog*. URL: https://colah.github.io/posts/2015-09-NN-Types-FP/ (besucht am 05. 12. 2022).

[17] Razvan Pascanu u. a. *How to Construct Deep Recurrent Neural Networks*. 24. Apr. 2014. arXiv: 1312.6026[cs,stat]. URL: http://arxiv.org/abs/1312.6026 (besucht am 17. 12. 2022).

[18] Angkoon Phinyomark u. a. „Analysis of Big Data in Gait Biomechanics: Current Trends and Future Directions“. In: *Journal of Medical and Biological Engineering* 38.2 (1. Apr. 2018), S. 244–260. ISSN: 2199-4757. DOI: 10.1007/s40846-017-0297-2. URL: https://doi.org/10.1007/s40846-017-0297-2 (besucht am 18. 10. 2022).

[19] Irfan Pratama u. a. „A review of missing values handling methods on time-series data“. In: *2016 International Conference on Information Technology Systems and Innovation (ICITSI)*. 2016 International Conference on Information Technology Systems and Innovation (ICITSI). Okt. 2016, S. 1–6. DOI: 10.1109/ICITSI.2016.7858189.

[20] G. Ramos u. a. „Fatigue Evaluation through Machine Learning and a Global Fatigue Descriptor“. In: *Journal of Healthcare Engineering* 2020 (6. Jan. 2020), S. 6484129. ISSN: 2040-2295. DOI: 10.1155/2020/6484129. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6969995/ (besucht am 18. 10. 2022).

[21] Timo Roth. *Rehabilitation nach Sportverletzung: spielerische Übungen im ExerCube*. 1. Juni 2022. URL: https://www.medica.de/de/reha-sportverletzung-ExerCube (besucht am 26. 10. 2022).

[22] Ilya Sutskever, Oriol Vinyals und Quoc V. Le. *Sequence to Sequence Learning with Neural Networks*. 14. Dez. 2014. arXiv: 1409.3215[cs]. URL: http://arxiv.org/abs/1409.3215 (besucht am 05. 12. 2022).

[23] Great Learning Team. *Introduction to Autoencoders? What are Autoencoders Applications and Types?* Great Learning Blog: Free Resources what Matters to shape your Career! 8. Mai 2020. URL: https://www.mygreatlearning.com/blog/autoencoder/ (besucht am 10. 12. 2022).

[24] *Understanding LSTM Networks – colah's blog*. URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/ (besucht am 06. 12. 2022).

[25] Vicon Motion Systems. *Vantage | Cutting Edge Flagship Camera by Vicon*. Vicon. URL: https://www.vicon.com/hardware/cameras/vantage/ (besucht am 21. 12. 2022).

[26] *What are Recurrent Neural Networks?* 7. Apr. 2021. URL: https://www.ibm.com/cloud/learn/recurrent-neural-networks (besucht am 05. 12. 2022).

[27] Huang Yi u. a. „A study on Deep Neural Networks framework“. In: *2016 IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*. 2016 IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC). Okt. 2016, S. 1519–1522. DOI: 10.1109/IMCEC.2016.7867471.

[28] ZHAW InIT/CAI. *InIT/CAI Lab Resources — InIT/CAI Lab Docu 2.1 documentation*. InIT/CAI Lab Resources. 14. Sep. 2022. URL: https://info.cloudlab.zhaw.ch/index.html (besucht am 09. 11. 2022).

[29] Victor Zhou. *Machine Learning for Beginners: An Introduction to Neural Networks*. Medium. 20. Dez. 2019. URL: https://towardsdatascience.com/machine-learning-for-beginners-an-introduction-to-neural-networks-d49f22d238f9 (besucht am 04. 12. 2022).

[30] Yong-Qiong Zhu, Ye-Ming Cai und Fan Zhang. „Motion Capture Data Denoising Based on LST-Net Autoencoder“. In: *Journal of Internet Technology* 23.1 (1. Jan. 2022). Number: 1, S. 11–20. ISSN: 2079-4029. URL: https://jit.ndhu.edu.tw/article/view/2637 (besucht am 18. 10. 2022).

Abbildungsverzeichnis

Tabellenverzeichnis

A. Anhang

A.1. Aufgabenstellung

A.1.1. Titel

Machine Learning-Based Analysis of Data from the ZHAW Movement Analysis Laboratory for Fatigue Detection during Sports Exercises

A.1.2. Beschreibung

In the movement analysis laboratory at ZHAW School of Health Sciences, a variety of data is collected in clinical studies. Examples include multi-dimensional movement data, force or data on muscle activity. Traditionally, data are analysed in a simplified manner which limits the interpretability of the data. Machine Learning principles have the potential to allow a deeper understanding of underlying principles that could revolutionize the conclusions that are possible based on data analysis.

In this thesis, kinematic data from 25 minute workouts will be utilised to identify patterns indicative of fatigue during exercise. The ultimate goal is to detect fatigue early during training. The results will be applied during the development of a new approach for sports rehabilitation. During rehabilitation, it is of importance to challenge but to not put the athlete at risk of re-injury, for example due to fatigue.

Researchers from the movement laboratory will support this thesis and provide support for all questions related to movement analysis.

Work packages: - Review of related work - Familiarisation with kinematic data - Iteratively develop an appropriate machine learning solution - Write a scientific report with a focus on motivation, argumentation, methods, evaluation & results (the PA thesis)

A.1.3. Voraussetzungen

This thesis can be conducted in German or English. The students should have strong interest and potentially even basic knowledge in machine learning and constructing (deep) learning models in Python. Strong interest in AI and hands-on work to build something that lasts is required, as well as the ability to independent conceptual work, very good general programming skills and a pragmatic approach to development and experimentation. Top grades in the previous course of study are typically indicative of these skills.

The Computer Vision, Perception and Cognition Group at the newly-founded ZHAW Centre for Artificial Intelligence conducts research and teaching in pattern recognition using deep learning methodology. In our thesis proposals, we want to address high-achieving students with a curiosity for gaining (first) experience with scientific research applied to the problem at hand. The supervisors are very enthusiastic about the topics and have plenty of experience as well as ideas (and offering of support) for the project startup. We are looking for equally enthusiastic students, with the hope of building a solution that will be used as a starter for long-term collaboration between the CAI and the movement analysis laboratory.

A.2. Quellcode

Der Quellcode ist auf dem ZHAW GitHub unter der folgenden URL verfügbar:
`https://github.zhaw.ch/PA-Movement-Analysis/Fatigue_Detection_With_Autoencoder`