HAW HAMBURG

Bachelorarbeit

Robin Müller

Erkennung menschlicher Aktivitäten auf Basis von multivariaten Zeitreihendaten mit Hilfe von Deep Learning Verfahren

Fakultät Technik und Informatik
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

Robin Müller

Erkennung menschlicher Aktivitäten auf Basis von multivariaten Zeitreihendaten mit Hilfe von Deep Learning Verfahren

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

Betreuender Prüfer: Prof. Dr. Kai von Luck
Zweitgutachter: Prof. Dr. Tim Tiedemann

Eingereicht am: 21.06.2019

Robin Müller

Thema der Arbeit

Erkennung menschlicher Aktivitäten auf Basis von multivariaten Zeitreihendaten mit Hilfe von Deep Learning Verfahren

Stichworte

Machine Learning, Deep Learning, Neuronales Netzwerk, Long Short-Term Memory, multivariat, Zeitreihendaten, Quantified Self

Kurzzusammenfassung

Quantified Self beschreibt das Aufzeichnen von personenbezogenen Daten, die für medizinische und sportliche Zwecke verwendet werden können. Der Bereich der Human Activity Recognition verwendet dabei verschiedene Sensoren, um Daten über menschliche Bewegungen aufzuzeichnen. Diese Daten können anschließend verwendet werden, um eine künstlichen Intelligenz zu trainieren, die in der Lage ist aus den Sensordaten die Bewegungen zu erkennen und korrekt und interpretieren. Diese Arbeit untersucht einen solchen Datensatz, bei dem einfache Bewegungsaktivitäten, wie Sitzen, Stehen oder Gehen kategorisiert werden. Dazu wurden der Beschleunigungssensor und das Gyroskop eines Smartphones, das an einem Gürtel getragen wurde, aufgezeichnet und in Zeitschritten einzelnen Bewegungsaktivitäten zugeordnet. Da es sich bei den Daten um Zeitreihendaten handelt, setzt diese Arbeit ein rekurrentes neuronales Netz, speziell ein Long Short-Term Memory (LSTM) ein. Mit Hilfe des LSTMs wird ein Modell auf dem Datensatz trainiert, das in der Lage ist den zeitlichen Kontext der Datenpunkte zu berücksichtigen. Es wird erwartet, dass diese zusätzliche betrachtete Dimension dem LSTM einen Vorteil gegenüber anderen Methoden bei der Erkennung der Bewegungsaktivitäten bietet. Um das Ergebnis dieser Arbeit zu bewerten, wird das entwickelte Modell mit den Ergebnissen anderer Arbeiten verglichen, die ebenso Modelle für den gleichen Datensatz trainierten. Die Experimente dieser Arbeit konnten eine Erkennungsrate von bis zu 98,46% erzielen. Damit zeigt das Modell einen erkennbaren Vorteil gegenüber anderen State-of-the-Art Modellen, die eine Genauigkeit von bis zu 97,70% erreichen. Die Experimente zeigen außerdem, dass die Betrachtung des zeitlichen Kontexts einen positiven Effekt hervorbringt. Damit kommt diese Arbeit zu dem Schluss, dass LSTMs ein vielversprechendes Verfahren für die Erkennung menschlicher Bewegungsaktivitäten auf dem verwendeten Datensatz sind.

Robin Müller

Title of Thesis

Human Activity Recognition based on multivariate time series data using Deep Learning techniques

Keywords

Machine Learning, Deep Learning, Neural Network, Long Short-Term Memory, multivariate, time series data, quantified self

Abstract

Quantified Self describes the recording of person-related data that can be used for medical and athletic purposes. The field of Human Activity Recognition uses various sensors to record data about human movements. This data can then be used to train an artificial intelligence that is able to recognize and correctly interpret the movements from the sensor data. This thesis examines a data set that categorizes simple movement activities such as sitting, standing or walking. For this purpose, the acceleration sensor and the gyroscope of a smartphone worn on a belt were recorded and assigned to individual movement activities in time steps. Since the data are time series data, this thesis uses a recurrent neural network, especially a Long Short-Term Memory (LSTM). The LSTM is used to train a model on the data set that is able to take into account the temporal context of the data points. It is expected that this additional dimension being considered will give the LSTM an advantage over other methods in detecting motion activities. In order to evaluate the result of this thesis, the developed model will be compared with the results of other studies, which also trained models for the same data set.
The experiments of this work were able to achieve a detection rate of up to 98.46%. Thus, the model shows a recognizable advantage over other state-of-the-art models that reach an accuracy of up to 97.70%. The experiments also show that the consideration of the temporal context produces a positive effect. Thus, this paper concludes that LSTMs are a promising method for the detection of human motion activities on the data set used.

Danksagung

Ich danke der Firma Acando und im Speziellen Tobias Kleyer für die Unterstützung bei der Themenfindung, sowie für die Möglichkeit, diese Arbeit im Rahmen meiner Werkstudentenanstellung erstellen zu können.

Außerdem danke ich den Mitgliedern der Machine Learning AG an der HAW Hamburg für den regen Wissensaustausch und die konstruktive Kritik. Zusätzlicher Dank gilt Tobias Eichler für die Unterstützung bei der Anfertigung eines Papers zum Thema dieser Arbeit.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Daten sind das wertvollste Gut des digitalen Zeitalters und werden vermehrt sogar als das neue Gold bezeichnet [43] [17]. Big Data hat sich in den letzten Jahren rasant entwickelt und wird mittlerweile von großen wie kleinen Unternehmen für den Informationsgewinn eingesetzt [6]. Der Wert der Daten liegt jedoch nicht in ihrer bloßen Ansammlung, sondern in den Informationen, die durch Analysen und Manipulationen gewonnen werden können [17]. Doch diese gewaltigen Datenmengen zu analysieren stellt seine eigenen Herausforderungen dar und ist kaum noch direkt durch den Menschen umsetzbar. Für diesen Anwendungsfall wurden automatisierte Verfahren populär, die auch bei riesigen Datenbeständen sinnvolle Erkenntnisse hervorbringen können. Der Bereich der künstlichen Intelligenz bzw. Machine Learning erlebt in den letzten Jahren eine Renaissance und wird nach wie vor für immer mehr Geschäftszweige und Forschungszwecke relevant [43]. Ein solcher Anwendungsfall ist das Quantified Self. Swan definiert ein Quantified Self als ein Individuum, dass sich in jeglicher Art der Selbst-Quantifizierung von biologischen, physikalischen, verhaltens- oder umweltbezogenen Informationen unterzieht [56]. Die aufgezeichneten Daten eines Quantified Selfs können anschließend unter anderem zur Ermittlung von gesundheitlichen oder sportlichen Problemstellungen eingesetzt werden.

Ein Teilbereich des Quantified Self wird durch die Human Activity Recognition abgedeckt. Human Activity Recognition konzentriert sich auf das Erkennen von menschlichen Bewegungen durch das Aufzeichnen durch verschiedenste Sensoren [3]. Die aufgezeichneten Daten können von externen Sensoren [44], wie Infrarot oder Ultraschall, oder von körpergetragenen Sensoren, die zum Beispiel die Beschleunigung oder räumliche Position aufnehmen, stammen [39] [31]. Für diese Arbeit wurde ein Datensatz aus der Human Activity Recognition gewählt, welcher durch den Beschleunigungssensor und das Gyroskop eines körpergetragenen Smartphones Bewegungsaktivitäten als Zeitreihendaten aufgezeichnet hat [3]. Seit der Veröffentlichung des Datensatzes wurden bereits verschiedene Methoden des Machine Learnings eingesetzt, um die Bewegungsaktivitäten möglichst

genau zu erkennen [27] [9]. In den letzten Jahren wurden dabei Deep Learning Modelle unter Einsatz von neuronalen Netzen populär. Jedoch wurde der Datensatz bis 2018 von allen Modellen bloß als unabhängige Datenpunkte untersucht und die zeitliche Abhängigkeit der Daten wurde nicht betrachtet. Erste Untersuchungen in 2018 setzten rekurrente neuronale Netze, wie das Long Short-Term Memory ein, um zusätzlich die zeitliche Dimension betrachten zu können [59]. Auch wenn diese Modelle nicht die Genauigkeit damaliger State-of-the-Art Modelle erreichen konnten, zeigten sie bereits die Möglichkeiten, die sich durch die Betrachtung des zeitlichen Kontexts der Daten ergeben.

1.1 Ziel der Arbeit

Diese Arbeit hat zum Ziel, den gewählten Human Activity Recognition Datensatz als Zeitreihendaten zu untersuchen. Das Ergebnis soll ein Deep Learning Modell sein, das in der Lage ist, die Bewegungsaktivitäten möglichst genau zu erkennen. Dazu wird das Long Short-Term Memory als Methode gewählt, da es den zeitlichen Kontext der Daten erfassen kann [24] und durch erste Untersuchungen bereits Potential für diese Problemstellung zeigte [59].

Die Unterscheidung von mehreren Bewegungsaktivitäten wird dabei als Klassifikationsproblem [35] aufgefasst, bei dem jede Bewegungsaktivität durch eine Aktivitätsklasse repräsentiert wird. Die Daten sind außerdem multivariat, da die Daten mehrerer Sensoren gleichzeitig untersucht werden, um die korrekte Bewegungsaktivität zu bestimmen. Um ein möglichst optimiertes Modell zu entwickeln, werden verschiedene Experimente durchgeführt. Diese haben zum Ziel, durch wiederholtes Training aus den möglichen Konfigurationen, die für das Modell vorgenommen werden können, eine möglichst optimale Konfiguration zu ermitteln. Das resultierende beste Modell wird anschließend analysiert, um dessen Eigenschaften einschätzen zu können. Im Speziellen soll untersucht werden, ob ein positiver Effekt durch die Betrachtung der Daten als Zeitreihendaten vorliegt. Die Güte des Modells ist schlussendlich abhängig von der Genauigkeit bei der Erkennung der Bewegungsaktivitäten. Das fertige Modell wird abschließend mit den Ergebnissen anderer Arbeiten auf dem gleichen Datensatz verglichen, um dessen Güte im Kontext einzuordnen.

1.2 Struktur der Arbeit

Die folgende Arbeit ist in mehrere Kapitel unterteilt, die die aufeinanderfolgenden Arbeitsschritte beinhalten. Im Analysekapitel wird zunächst die Problemstellung näher betrachtet. Dazu werden anschließend die wichtigsten Begriffe geklärt, die im Verlauf dieser Arbeit aufkommen. Es wird außerdem der Datensatz näher vorgestellt und charakterisiert sowie dargestellt, welche Ergebnisse bereits durch andere Arbeiten erzielt wurden. Abschließend wird die Zielsetzung abgegrenzt und zusammengefasst, welche Anforderungen sich aus der Analyse ergeben.
Im nächsten Kapitel wird der Aufbau eines Klassifikators beschrieben, nach dessen Vorgaben das Modell trainiert wird. Dazu wird die benötigte experimentelle Umgebung beschrieben und welche Technologien in der Umsetzung genutzt werden.
Im Evaluationskapitel werden der Aufbau, die Durchführung und die Ergebnisse der Experimente zusammengefasst. Außerdem werden die Eigenschaften der entwickelten Modelle genauer untersucht. Aus den Modellen wird anschließend die beste Version gewählt und mit den Modellen anderer Arbeiten verglichen. In einem abschließenden Fazit werden mögliche weitere Schritte diskutiert, die durchgeführt werden könnten, falls die Experimente fortgeführt werden.
Im letzten Kapitel werden alle Erkenntnisse der Arbeit zusammengefasst und ein Ausblick gegeben, welche Rolle diese für zukünftige Forschungen spielen können.

2 Analyse

In der Analyse werden die wichtigsten Komponenten, auf denen diese Arbeit aufbaut, vorgestellt und erläutert. Dazu wird zunächst die Problemstellung aufgeführt. Danach werden grundlegende Konzepte und Methoden, die für das Ziel der Arbeit benötigt werden, im Einzelnen beleuchtet. Anschließend wird der Datensatz präsentiert, auf dessen Grundlage die Experimente dieser Arbeit stattfinden. Es wird abgegrenzt, welche Ressourcen durch den Datensatz verfügbar sind, wo jedoch auch die Grenzen liegen. Darauf folgend wird die inhaltliche Zielsetzung der Arbeit definiert. Zuletzt werden die Anforderungen zusammengefasst, die sich aus der Analyse für die Arbeit ergeben haben.

2.1 Problemstellung

Der Bereich Human Activity Recognition (HAR) hat das Ziel, menschliche Bewegungen durch Daten aus verschiedenen Sensoren [3] aufzuzeichnen und zu erkennen. Es werden Machine Learning (ML) Methoden eingesetzt, um aus den aufgezeichneten Daten die korrekten Bewegungen zu identifizieren. Die Daten können durch externe Sensoren [44], wie zum Beispiel Ultraschall oder Infrarot, aufgezeichnet werden. Alternativ können ebenso körpergetragene Sensoren engesetzt werden. Dazu existieren verschiedene Optionen, die sich in Präzision, Umfang der aufgezeichneten Daten und Tragekomfort unterscheiden [39] [31]. Eine einfache Möglichkeit, Daten von körpergetragenen Sensoren zu erfassen, besteht bereits durch ein getragenes Smartphone.

In dieser Arbeit wird ein Datensatz untersucht, der unter Laborbedingungen eine Gruppe von Freiwilligen, mit an Gürteln getragenen Smartphones, ein Protokoll von Bewegungsaktivitäten hat ausüben lassen [3]. Während des Ausübens der vorgeschriebenen Aktivitäten wurden Zeitsignale des Beschleunigungssensors und des Gyroskops des Smartphones aufgezeichnet. Insgesamt unterscheidet der Datensatz sechs Bewegungsaktivitäten. Die

Aktivitäten spalten sich in drei dynamische und drei statische Aktivitäten. Die dynamischen Bewegungsaktivitäten sind Gehen, Treppen hinaufgehen und Treppen hinuntergehen. Die statischen Aktivitäten sind Sitzen, Stehen und Liegen.

Der Datensatz wurde bereits in verschiedenen, wissenschaftlichen Publikationen untersucht, in denen Modelle entwickelt wurden, um die Bewegungsaktivitäten möglichst genau zu identifizieren [2] [27] [9]. Es handelt sich bei dem Datensatz um ein multivariates Klassifikationsproblem auf Zeitreihendaten. Ein Großteil der Arbeiten, wie bei Anguita et. al. [2], Li et. al. [38] oder Ronao et. al [50] untersuchten den Datensatz im situationsbasierten Kontext und nicht mit Hilfe von Zeitreihen. Da es sich bei dem Datensatz jedoch im Ursprung um Zeitreihendaten handelt, ist es das Ziel dieser Arbeit eine Methode zu untersuchen, die diesen Kontext erfassen kann. Hierfür können allgemein rekurrente neuronale Netze (RNN) [10] eingesetzt werden. Diese Arbeit entschied sich für die Untersuchung eines spezifischen RNNs, das Long Short-Term Memory (LSTM) [24]. Das Ziel dieser Arbeit ist es, den Datensatz unter Einsatz eines LSTMs zu untersuchen, um anschließend evaluieren zu können, ob die Betrachtung der Daten als Zeitreihen eine positive Auswirkung auf die Erkennungsrate der Bewegungsaktivitäten hat.

2.2 Machine Learning

„[Machine Learning is the] field of study that gives computers the ability to learn without being explicitly programmed.“ Arthur Samuel, 1959

Machine Learning bezeichnet die Disziplin der automatisierten Entscheidungsprozesse von Algorithmen und Methoden für komplexe Probleme. Machine Learning wird allgemein in drei Disziplinen unterteilt. Unsupervised Learning [23], Supervised Learning [35] und Reinforcement Learning [30]. In dieser Arbeit wird lediglich das Supervised Learning betrachtet. Die klassische Programmierung legt eine Menge von Regeln zu einer gegebenen Eingabe fest, um eine gewünschte Ausgabe zu erhalten. Machine Learning im Bereich des Supervised Learning hingegen speist eine bekannte Eingabe (Features) mit der zugehörigen Ausgabe (Labels) in eine Methode, die durch ein anschließendes Training ein Modell entwickelt [35]. Dieses Modell legt durch das wiederholte Training eine Menge von Regeln fest, um mit den gegebenen Features die korrekten Labels vorherzusagen. Es existiert selten ein Modell, das für ein gegebenes Problem zu 100% der Fälle eine korrekte Vorhersage trifft. Machine Learning wird vor allem für Entscheidungsprozesse eingesetzt,

die zu komplex sind, um eine perfekte Vorhersagequalität zu erreichen. Das Ziel ist es stattdessen, die Fehlerrate zu minimieren (siehe Fehlerfälle in 2.3).

Ein Modell besitzt eine Menge von Eigenschaften, die angepasst werden können, um das Training für einen expliziten Fall zu optimieren. Diese Eigenschaften werden Hyperparameter genannt [11]. Das Finden der optimalen Hyperparameter, das sogenannte Hyperparameter Tuning, eines Modells ist ein wichtiger Schritt, um eine zufriedenstellende Leistung des Modells zu ermöglichen. Das Hyperparameter Tuning ist jedoch ein aufwendiger Prozess, der entweder von Hand oder automatisiert stattfinden kann [11]. Wird das Finden optimaler Hyperparameter automatisiert, so existieren verschiedene Vorgehen. Die einfachste Methode ist Grid Search [11]. Dabei wird aus einer Liste von verfügbaren Konfigurationen von Hyperparametern sequentiell jede Konfiguration für das Training ausprobiert, bis alle möglichen Konfigurationen versucht wurden. Diese Vorgehensweise ist jedoch vor allem für große Suchräume an Hyperparametern ineffizient, da die Reihenfolge der Konfigurationen eine Auswirkung auf die benötigte Zeit für das Finden der optimalen Lösung haben kann [11]. Alternativ kann Random Search [5] eingesetzt werden. Hierbei wird ebenso eine Auflistung aller möglichen Hyperparameterkonfigurationen untersucht, jedoch ist die Reihenfolge der Untersuchung zufällig. Wie Bergstra und Bengio [5] aufzeigten, ergibt sich durch Random Search ein merkbarer Vorteil in der Dauer, die es benötigt eine zufriedenstellende Konfiguration zu ermitteln. Es existieren weitere Methoden, wie Bayesian Hyperparameter Optimization [41] [29], die eigene Machine Learning Verfahren für das Hyperparameter Tuning einsetzen.

Einen Teilbereich des Machine Learning bilden neuronale Netze [21]. Die einfachste Form eines neuronalen Netzes stellt das Feed Forward Netz dar. Weitere Varianten von neuronalen Netzeen sind unter anderem das Convolutional Neural Network (CNN) [36] und das RNN. Das Einsatzgebiet eines speziellen neuronalen Netzes hängt stets von den genauen Umständen ab. CNNs werden klassischerweise in der Bilderkennung genutzt [37], wohingegen RNNs in der Sequenzverarbeitung eingesetzt werden [10].
Ein Teilbereich der neuronalen Netze wiederum, wird außerdem unter dem Begriff Deep Learning abgegrenzt [53]. Dabei beschreibt Deep Learning den Einsatz von neuronalen Netzen, die eine Vielzahl von Hidden Layern besitzen.

2.3 Klassifikation

Bei einem Klassifikationsproblem handelt es sich um das Differenzieren von Kategorien mit einer eindeutigen Zuordnung [35]. Dem gegenüber stehen Regressionsprobleme, bei denen ein quantifizierbarer Wert ermittelt werden soll [7]. Im Kontext dieser Arbeit bedeutet dies, dass das entwickelte Modell in der Lage sein muss, eine Eingabe von Features zu einem gegebenen Zeitschritt zu deuten und korrekt als eine der vorgegebenen Klassen zu interpretieren. Unter Feature wird in diesem Kontext ein einzelner Eingabewert bezeichnet. Die Gesamtheit aller Eingabewerte des Datensatzes wird Featurevektor genannt. Ein einzelner Datenpunkt mit seinen Features wird als Example bezeichnet. Beinhaltet der Featurevektor bloß einen einzigen Wert, so spricht man von einem univariaten Klassifikationsproblem. Mit einem Featurevektor von mehreren Werten handelt es sich um ein multivariates Klassifikationsproblem. Die Werte der Ausgabe, der Menge der zu ermittelnden Klassen, werden wiederum als Labels bezeichnet.

In der binären Klassifikation wird in verschiedene Fehlerfälle unterschieden, die je nach Kontext des Modells unterschiedlich gewichtet sein können [35]. False Positive beschreiben Fehlerfälle, bei denen ein falsches Ergebnis als richtig klassifiziert wurde. False Negative beschreibt ein richtiges Ergebnis, das als falsch klassifiziert wurde. Zum Beispiel in der Spamfilterung mag es wichtiger sein, allen existierenden Spam zu identifizieren, auch wenn dadurch Nachrichten, die kein Spam sind, fälschlicherweise als solcher klassifiziert werden [34]. Für diese Arbeit werden alle Fehlerfälle jedoch gleich schwer gewichtet, da es beim fehlerhaften Identifizieren einer Bewegungsaktivität keinen Unterschied macht, womit die Aktivität verwechselt wurde. Es ist nur die allgemeine Genauigkeit des Modells relevant, da die Ergebnisse anderer Arbeiten ebenso diese Metrik nutzen und so eine direkte Vergleichbarkeit der Ergebnisse möglich ist.

2.4 Zeitreihendaten

Um durch eine Machine Learning Methode, speziell unter Einsatz eines neuronalen Netzes, ein zufriedenstellendes Ergebnis zu erhalten, werden stets eine große Menge an Examples für den eingesetzten Datensatz benötigt [17]. Die häufigste Betrachtungsart der Daten ist dabei situationsbasiert. Einzelne Examples werden dabei als alleinstehende Situationen voneinander getrennt betrachtet. So steht jedes Example für sich und es besteht kein übergreifender Zusammenhang zwischen ihnen.

Alternativ können Daten in einer Sequenz angeordnet sein. Sequentiell betrachtete Daten werden auch als Zeitreihendaten bezeichnet [28]. Zeitreihendaten sind ein häufiges Vorkommen bei realweltlichen Aufzeichnung, wie Sensordaten oder Kameraaufnahmen. Bei dieser Art von Daten existieren zusätzlich zu den internen Informationen pro Examples, die durch ihre Features abgebildet werden, implizite Informationen durch die Reihenfolge der Daten. Im Falle dieser Arbeit handelt es sich bei den Examples um Zeitfenster, die aufeinanderfolgend die Aufzeichnungen von 2,5 Sekunden zusammenfassen.
Der Vorteil von Zeitreihendaten ist es, dass vergangene Zustände berücksichtigt werden können, wenn ein neuer Zustand untersucht wird. Dadurch können zum Beispiel Trends in den Daten identifiziert werden. Gleichzeitig ist jedoch nicht jedes Machine Learning Verfahren in der Lage, den zeitlichen Kontext von Daten zu berücksichtigen. Hierfür existieren RNNs, wie das LSTM, das in dieser Arbeit verwendet und im Folgenden näher erläutert wird.

2.5 Long Short-Term Memory

Ein Long Short-Term Memory (LSTM) ist eine spezielle Variante eines RNNs [24]. RNNs sind neuronale Netze, die in ihren Zustandsübergängen Informationen speichern und an folgende Zustände weiterreichen [19]. Somit entsteht ein Gedächtnis, das die Betrachtung eines zeitlichen Kontexts auf den Daten ermöglicht. LSTMs erweitern diese grundlegende Architektur eines RNNs um drei Gates, die in jeder Zelle des LSTM vorkommen [25]. Diese sind das Input Gate, das Forget Gate und das Output Gate. Jedes der Gates kontrolliert zu einem speziellen Zeitpunkt in der Zelle den Stand des Gedächtnis. Dabei können einzelne Informationen angepasst, abgeschwächt, oder absichtlich vergessen werden. Mit diesem kontrollierten Merken und Vergessen von vorangegangenen Zuständen des Netzes bietet das LSTM eine populäre Methode für Zeitreihenprobleme und wird deshalb auch in dieser Arbeit eingesetzt.

2.6 Vorstellung des Datensatzes

Der Datensatz, der in dieser Arbeit verwendet wird, wird in den zwei verfügbaren Versionen einzeln im Aufbau vorgestellt und näher beschrieben. Anschließend wird zusammengefasst, welche verwendbaren Ressourcen sich aus dem Datensatz für die Arbeit ergeben.

2.6.1 Human Activity Recognition Using Smartphones Data Set

Der Datensatz zur Erkennung von menschlichen Aktivitäten mit Hilfe von Smartphones wurde von Anguita et al. [3] vorgestellt und erstmals am 10.12.2012 veröffentlicht.

Das Ziel des Datensatzes ist es, menschliche Bewegungsaktivitäten mittels Klassifikation unter Einsatz von Sensoren eines getragenen Smartphones zu identifizieren. Der Datensatz wurde über eine Reihe von Experimenten gewonnen, in dem 30 Freiwillige in einem Alter von 19 bis 48 Jahren angewiesen wurden, ein Protokoll von Aktivitäten durchzuführen. Zur Aufnahme trugen die Freiwilligen Gürtel, an denen ein Samsung Galaxy S II montiert wurde. Das Aktivitätsprotokoll bestand aus den sechs Aktivitäten Stehen, Sitzen, Liegen, Gehen, Treppen hinuntergehen und Treppen hinaufgehen. Zwischen den Aktivitäten wurden fünfsekündige Ruhepausen eingehalten und die Freiwilligen wurden dazu angehalten, die Bewegungen trotz der Laborbedingungen frei durchzuführen, um den Datensatz natürlich zu gestalten.

Die Bewegungsaktivitäten wurden mit Hilfe des Beschleunigungssensors und des Gyroskops des Smartphones bei einer Messrate von 50Hz aufgezeichnet. Dadurch wurden dreiachsige lineare Beschleunigungs- und Winkelgeschwindigkeitssignale gewonnen, die anschließend mit verschiedenen Filtern zur Rauschunterdrückung vorverarbeitet wurden. Die vorverarbeiteten Zeitsignale wurden danach in Schiebefenstern mit einer festen Breite von 2,56 Sekunden und 50% Überlappung gesampelt, sowie durch eine schnelle Fourier-Transformation in den Frequenzbereich abgebildet (mehr Details in [3]).

Aus der Vorverarbeitung ergeben sich 17 Signale, die in Tabelle 2.1 aufgeführt werden. Dabei sind Zeitsignale mit einem „t“-Präfix, Frequenzsignale mit einem „f“-Präfix und dreiachsige Signale mit einem „XYZ“-Suffix notiert.

Auf Basis dieser Signale wurde eine Reihe von Variablen berechnet. In Tabelle 2.2 werden alle Funktionen aufgelistet, die zur Berechnung der Variablen genutzt wurden. Es ergeben sich aus den Berechnungen 561 Features für jedes Zeitfenster als Input des Datensatzes. Dabei wurden die Features nicht aufgabenorientiert erzeugt, sondern allgemein gehalten. Die den Featurevektoren zugeordneten Labels beschreiben jeweils eine der sechs Aktivitäten, wie in Tabelle 2.3 aufgelistet. Insgesamt weist der Datensatz 10.299 Features und Labels auf. Davon wurden zufällig 70% als Trainingsdaten und 30% als Testdaten designiert.

Zeitsignale	Frequenzsignale
tBodyAcc-XYZ	fBodyAcc-XYZ
tGravityAcc-XYZ	
tBodyAccJerk-XYZ	fBodyAccJerk-XYZ
tBodyGyro-XYZ	fBodyGyro-XYZ
tBodyGyroJerk-XYZ	
tBodyAccMag	fBodyAccMag
tGravityAccMag	
tBodyAccJerkMag	fBodyAccJerkMag
tBodyGyroMag	fBodyGyroMag
tBodyGyroJerkMag	fBodyGyroJerkMag

Tabelle 2.1: Ermittelte Zeit- und Frequenzsignale

Function	Description
mean()	Mean value
std()	Standard deviation
mad()	Median absolute deviation
max()	Largest value in array
min()	Smallest value in array
sma()	Signal magnitude area
energy()	Energy measure. Sum of the squares divided by the number of values.
iqr()	Interquartile range
entropy()	Signal entropy
arCoeff()	Autorregresion coefficients with Burg order equal to 4
correlation()	correlation coefficient between two signals
maxInds()	index of the frequency component with largest magnitude
meanFreq()	Weighted average of the frequency components to obtain a mean frequency
skewness()	skewness of the frequency domain signal
kurtosis()	kurtosis of the frequency domain signal
bandsEnergy()	Energy of a frequency interval within the 64 bins of the FFT of each window.
angle()	Angle between to vectors.

Tabelle 2.2: Funktionen zur Berechnung der Featurevektoren. Basierend auf [3]

2.6.2 Smartphone-Based Recognition of Human Activities and Postural Transitions Data Set

Reyes-Ortiz et al. [48] veröffentlichten aus der gleichen Forschungsgruppe, die bereits den „Human Activity Recognition Using Smartphones“ Datensatz vorstellte, den „Smartphone-

Aktivität	Label
WALKING	1
WALKING_UPSTAIRS	2
WALKING_DOWNSTAIRS	3
SITTING	4
STANDING	5
LAYING	6

Tabelle 2.3: Abbildung der Aktivitäten auf ganzzahlige Labels

Based Recognition of Human Activities and Postural Transitions“ Datensatz als aktualisierte und erweiterte Version. Dieser wurde am 29.07.2015 veröffentlicht.

Da keine Änderungen an den Features vorgenommen wurden, folgt der Datensatz dem gleichen Aufbau, der bereits in Abschnitt 2.6.1 beschrieben wurde. Der Datensatz wurde jedoch um sechs Bewegungsaktivitäten als weitere Labels erweitert, die den Übergang zwischen den statischen Aktivitäten aufgreifen. Außerdem wurden die rohen Sensordaten, die nicht Teil des ursprünglichen Datensatzes waren, der Veröffentlichung beigefügt. Dabei umfassen die Übergangsaktivitäten 8% der experimentellen Daten [48]. In Tabelle 2.4 werden die dadurch erweiterten Labels aufgelistet.

Aktivität	Label
WALKING	1
WALKING_UPSTAIRS	2
WALKING_DOWNSTAIRS	3
SITTING	4
STANDING	5
LAYING	6
STAND_TO_SIT	7
SIT_TO_STAND	8
SIT_TO_LIE	9
LIE_TO_SIT	10
STAND_TO_LIE	11
LIE_TO_STAND	12

Tabelle 2.4: Erweiterung der Labels um Übergangsaktivitäten

2.6.3 Verfügbare Ressourcen

Aus den vorgestellten Versionen des Datensatzes liegen unterschiedliche Daten für Experimente bereit. Diese umfassen die rohen Sensordaten der getragenen Smartphones (unterteilt in einzelne Dateien pro Teilnehmer, Experiment und Sensortyp), die daraus abgeleiteten 561 Features pro Zeitfenster von 2,56 Sekunden für 10.299 Zeitschritte (unterteilt in 70% Trainingsdaten und 30% Testdaten), Labels (mit und ohne Übergangszustände), die jedem Zeitschritt eine der 6, bzw. 12 Aktivitätsklassen als Labels zuordnen, sowie eine Zuordnung jedes Zeitschrittes zu einem individuellen Freiwilligen als alternatives Labeling.

Jedoch liegen keine weiteren Attribute vor, die den Kontext der Daten näher spezifizieren würden, wie geographische, oder kalendarische Zusatzinformationen. Beide Versionen des Datensatzes berechnen auf den erhobenen Rodaten lediglich eine Menge von allgemeinen, dekontextualisierten Features.

2.7 Bisherige Ergebnisse auf dem Datensatz

Es existieren bereits eine Vielzahl von Arbeiten, die Modelle auf Basis des vorgestellten Datensatzes trainiert und veröffentlicht haben. Um Vergleichswerte für die Ergebnisse dieser Arbeit zu schaffen, werden zunächst eine exemplarische Anzahl von bisherigen Arbeiten aufgelistet und erläutert. Durch die Grundlage an existierenden Methoden für den Datensatz werden anschließend die Ziele dieser Arbeit konkretisiert und abgegrenzt.

Die ersten Modelle auf dem Datensatz erschienen 2012 aus der Forschungsgruppe von Anguita et. al. [2], die den Datensatz hierfür erstmals aufzeichnete. Dabei verließ man sich auf die in Abschnitt 2.6.1 beschriebenen, statischen Features, sowie eine Support Vector Machine (SVM) als Methode zur Klassifizierung. SVMs wurden ursprünglich als binärer Klassifikator entwickelt, später jedoch auch für mehrklassige Modelle erweitert [26]. Diese ersten Modelle erreichten eine Genauigkeit von bis zu 89,00%.

2014 wurden die SVM Modelle nach wie vor eingesetzt, jedoch um weitere Methoden erweitert, um die Genauigkeit zu verbessern. Li et. al. [38] konstruierte Modelle, die in einem Fall um geschachtelte Autoencoder [58], in einem anderen Fall durch eine Principal Component Analysis (PCA) [1] erweitert wurden. Durch diese Erweiterungen konnte die

Genauigkeit im Fall der Autoencoder auf 92,16% und für die PCA auf 91,82% verbessert werden.

In 2016 wurden erstmals effektive neuronale Netze, speziell Convolutional Neural Networks (CNN) [22] entwickelt, die genauere Vorhersagen als die bisherigen Modelle erreichten. So entwickelten Ronao et. al [50] nach ersten Versuchen in 2015 [52] mit einem CNN eine Genauigkeit von 94,79%. Das trainierte Modell verließ sich hierbei nicht auf die von Anguita et. al. [3] vorverarbeiteten, statischen Features, sondern nutze lediglich die rohen Sensordaten als Input. Dabei ermittelte das Modell selbstständig dynamische Features aus den Rohdaten.

2018 stellten Ignatov und Andrey [27] eine verbesserte Version eines CNN Klassifikators für den Datensatz vor, der eine Genauigkeit von bis zu 95,31% erzielte. 2018 wurden außerdem erste Arbeiten veröffentlicht, die Modelle mit Recurrent Neural Networks (RNN), im Speziellen Long Short-Term Memories (LSTM) entwickelten. So stellten Yu et. al. [59] ein mehrschichtiges, paralleles LSTM Netzwerk vor, das im Vorbild der State-of-the-Art CNN Modelle mit dynamisch ermittelten Features auf den Rohdaten arbeitete. Dieses Modell erreichte eine Genauigkeit von 94,34% und konnte somit die Genauigkeit der damaligen CNN Modelle nicht erreichen.

2019 veröffentlichten Chelli und Pätzold [9] jedoch neue Modelle, die die Genauigkeit der bisherigen State-of-the-Art CNN Modelle überbieten konnten. Im Gegensatz zu den CNN Modellen, wurden jedoch erneut statische Features genutzt. Diese wurden von Chelli und Pätzold selbstständig aus den Rohdaten hergeleitet. Ein Quadratic SVM Modell erreichte so eine Genauigkeit von 96,10% und ein Ensemble Bagged Tree Modell [13] sogar 97,70%. Es wurde ebenso ein Feed Fordward Netz auf den Daten trainiert, erreichte jedoch nur eine Genauigkeit von 85,80%. Zum Vergleich wurden alle Modelle ebenso auf den ursprünglichen Features trainiert. Dabei erreichte der Ensemble Bagged Tree beispielsweise lediglich eine Genauigkeit von 94,10%. Die Ergebnisse von Chelli und Pätzold sind bloß eingeschränkt mit den Ergebnissen dieser Arbeit vergleichbar, als dass der Einsatz der neuen Features einen sichtbar großen Effekt auf die Leistung der Modelle hat. Eine zusammenfassende Auflistung der bisherigen Ergebnisse der beschriebenen Arbeiten ist in Tabelle 2.5 einsehbar.

Methode	**Genauigkeit** (%)
Ensemble Bagged Tree [9]	97,70
Quadratic SVM [9]	96,10
Convolutional Neural Network [27]	95,31
Convolutional Neural Network [50]	94,79
Multi-Layer Parallel LSTM Network [59]	94,34
Hierarchical Continuous Hidden Markov Model [51]	93,18
Stacked Autoencoders + SVM [38]	92,16
PCA + SVM [38]	91,82
Hidden Markov Models [49]	91,76
Convolutional Neural Network [52]	90,89
Handcrafted features + SVM [2]	89,00
Dynamic Time Warping [54]	89,00
Artificial Neural Network [9]	85,80

Tabelle 2.5: Klassifikatoren anderer Arbeiten. Basierend auf [59]

2.8 Zielsetzung

Im Zuge dieser Arbeit soll ermittelt werden, ob es einem LSTM Klassifikator möglich ist, die Genauigkeit von State-of-the-Art Modellen zu übertreffen und welche Auswirkung die zeitliche Abhängigkeit des Klassifikators auf dessen Ergebnisse hat. Dazu sollen kontrollierte Experimente durchgeführt werden, die im Vergleich zu den bisherigen publizierten Klassifikatoren in ihrer Güte bewertet werden. Dazu werden bereits implementierte Verfahren, wie das LSTM eingesetzt und im Zuge der Arbeit keine neuen Verfahren entwickelt. Außerdem wird der Datensatz in der veröffentlichten Form eingesetzt und es wird kein weiteres Feature Engineering vorgenommen. Der Fokus der Arbeit liegt auf der experimentellen Findung möglichst optimaler Hyperparameter zur Erkennung von Bewegungsaktivitäten unter Einsatz eines LSTMs.

2.9 Anforderungen der Analyse

Der einzusetzende Datensatz besteht aus Zeitreihendaten, die jedem Zeitpunkt des Inputs einen eindeutigen Output aus einer Menge von Aktivitätsklassen zuweist. Es wurden verwandte Arbeiten erläutert, die Modelle für den Datensatz entwickelt haben und welche Vorgehensweisen dabei eingesetzt wurden. Das Bestreben dieser Arbeit ist es, ein LSTM Netzwerk zu konstruieren, das anschließend mit den vorgestellten Arbeiten anderer verglichen wird, um die Güte des Modells im Kontext zu ermitteln. Es wurde sich für diese Netzwerkarchitektur entschieden, da LSTMs durch die Natur der Zeitreihendaten einen entscheidenden Vorteil gegenüber anderen Methoden besitzen sollten, die wiederum den zeitlichen Kontext nicht betrachten können.

3 Aufbau eines Klassifikators

In diesem Kapitel wird der Aufbau und die Konstruktion beschrieben, nach dem ein Klassifikator entsteht und trainiert wird. Dabei wird zunächst beschrieben, wie die experimentelle Umgebung aufgebaut wird. Hieraus ergeben sich zusammenfassend die Anforderungen an die Umgebung der Experimente, die erfüllt werden müssen, um einen Klassifikator erfolgreich trainieren und evaluieren zu können. Abschließend werden die Technologien aufgeführt, die für die Umsetzung gewählt wurden.

3.1 Konstruktion der experimentellen Umgebung

Die experimentelle Umgebung wird nach dem Knowledge Discovery in Databases (KDD) Prozess von Fayyad et. al. [16] aufgebaut. Das allgemeine Ziel des KDD Prozesses ist es, Wissen aus Daten zu gewinnen. Dafür beschreibt der KDD Prozess fünf Schritte. Diese sind Selection, Preprocessing, Transformation, Data Mining und Interpretation/Evaluation. Der Prozess verarbeitet in jedem Schritt die vorliegenden Daten und übergibt diese sequentiell an den nächsten Schritt. Das Ergebnis des letzten Schrittes, der Evaluation, soll zu einem Wissenszuwachs führen. Falls das Ergebnis der Evaluation nicht nutzbar ist, so kann der Prozess ab einem beliebigen Schritt wiederholt werden, bis ein valides Ergebnis entsteht. Eine visuelle Repräsentation des KDD Prozesses wird in Abbildung 3.1 dargestellt. Der KDD Prozess wurde ursprünglich für das Data Mining entwickelt. Jedoch lässt sich der Prozess in seinen Kernkonzepten ebenso im Machine Learning einsetzen, da die Struktur einen klaren Fokus auf die Daten besitzt. Im Folgenden werden die fünf Schritte des KDD Prozesses im Einzelnen aufgeführt. Die Ziele jedes Schrittes werden für Machine Learning erläutert und es wird aufgezeigt, wie sie sich auf den speziellen Anwendungsfall dieser Arbeit umsetzen lassen.

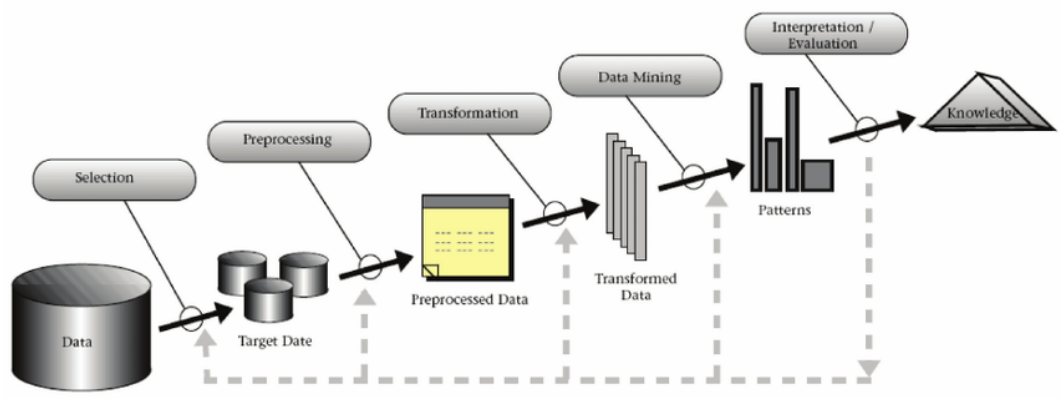

Abbildung 3.1: Knowledge Discovery in Databases Prozess nach Fayyad et. al. [16]

3.1.1 Selection

Im Selektionsschritt werden die Daten ermittelt, die durch den Klassifikator untersucht werden sollen. Es können verschiedene Datenbestände kombiniert oder nur Teile eines Datensatzes verwendet werden, je nachdem welcher Kontext für die gewählte Problemstellung relevant ist.

Die gewählten Daten dieser Arbeit wurden im Analysekapitel in Abschnitt 2.6 zuvor erläutert. Dabei beschränkt sich diese Arbeit auf die bereits vorverarbeiteten Features und zugehörigen Labels aus Aktivitätsklassen. Die folgenden Experimente werden nach der Version des Datensatzes unterteilt und getrennt voneinander für das Training eines Klassifikators eingesetzt. Zuerst wird der Datensatz mit den ursprünglichen sechs Labels untersucht. Nach Abschluss der Experimente mit dem ersten Datensatz, wird der Datensatz mit den alternativen 12 Labels betrachtet. Die verfügbaren Rohdaten oder weitere Labels, wie die Zuordnung von Freiwilligen, werden im Zuge der Experimente nicht weiter betrachtet.

3.1.2 Preprocessing

Im Vorverarbeitungsschritt werden die zuvor ausgewählten Daten angepasst, um irrelevante Daten herauszufiltern, bestehende anzupassen oder weitere hinzuzufügen. Wurde zum Beispiel eine Kombination aus zuvor unabhängigen Daten gewählt, wie die Kombination aus Wetter- und Straßenverkehrsdaten, so müssen diese korrekt zusammengeführt

werden. Bei realen Daten kann es zusätzlich zu Datenunreinheiten kommen, die vor der weiteren Nutzung der Daten bereinigt werden müssen. So können entweder unvollständige Daten vorliegen, oder aber auch eine fehlerhafte Aufzeichnung der Daten. In beiden Fällen müssen die Daten angepasst werden, um Fehler und ungewollte Interpretationen der Daten im Training des Klassifikators zu vermeiden.

Da der Datensatz, der in dieser Arbeit verwendet wird, unter Laborbedingungen erstellt und bereits in bereinigter Form veröffentlicht wurde, entfällt das Bereinigen von fehlerhaften oder fehlenden Daten. In dieser Arbeit wird ebenso kein zusätzliches Feature Engineering auf den Daten durchgeführt. Da die bereits vorverarbeiteten Daten von Anguita et. al. [3] eingesetzt werden, entfällt der Vorverarbeitungsschritt im Zuge dieser Arbeit.

3.1.3 Transformation

Im Transformationsschritt werden die vorverarbeiteten Daten in eine Form gebracht, die für den Klassifikator nutzbar ist. Es hängt vom eingesetzten Machine Learning Verfahren ab, in welcher Form Daten vorliegen müssen, um nutzbar zu sein. Es existieren jedoch Prinzipien, die für die meisten Machine Learning Verfahren gültig sind. Machine Learning Verfahren benötigen stets numerische Features und Labels, die wiederum in Listen oder Arrays zusammengeführt sind. Werden stattdessen, zum Beispiel Labels durch Zeichenketten repräsentiert, müssen diese auf einen numerischen Bereich abgebildet werden.

Für die Features eines Datensatzes existieren diverse Methoden, die eingesetzt werden können, um die Leistung des Verfahrens auf dem Datensatz zu optimieren. Die am häufigsten eingesetzte Methode ist das Normalisieren der Features, bei dem alle Features auf den Dezimalbereich von 0 bis 1 abgebildet werden. Der Datensatz von Anguita et. al. [3], der in dieser Arbeit verwendet wird, verfügt über normalisierte Features und eine numerische Abbildung der Aktivitätsklassen auf Labels von 1 bis 6, bzw. 1 bis 12.

Neben den genannten Transformationen, die von Anguita et. al. [3] bereits auf dem Datensatz durchgeführt wurden, müssen die Labels im Zuge dieser Arbeit zusätzlich einem One-Hot-Encoding [8] unterzogen werden. Das One-Hot-Encoding wird in der Klassifikation eingesetzt, um numerische Labels in eine binäre Form zu bringen. Dabei wird aus einem numerischen Label eine Liste gebildet. Die Länge der Liste entspricht der Anzahl an Kategorien, die dem Klassifikator bekannt sein sollen. Jedes Element der Liste repräsentiert eine der Kategorien. Eine Liste wird einer Kategorie zugeordnet, wenn das

korrekte Element den Wert 1 besitzt und die restlichen Elemente der Liste mit dem Wert 0 belegt sind. Im Falle dieser Arbeit muss jedes Label demnach auf eine Liste mit 6 Elementen abgebildet werden. Sollte ein Label die erste Kategorie repräsentieren, so nimmt die Liste folgende Form an: [1, 0, 0, 0, 0, 0]. Nachdem der Transformationsschritt durchgeführt wurde, ist der Datensatz für das Training des Klassifikators nutzbar.

3.1.4 Data Mining/Machine Learning

Für das Data Mining wird in diesem Schritt das eigentliche Mining der transformierten Daten durchgeführt. Für das Machine Learning wird stattdessen das Modell spezifiziert und auf den transformierten Daten trainiert. Ein Modell trainiert in einer vorher spezifizierten Anzahl von Epochen. In jeder Epoche wird der Datensatz vollständig für das Training untersucht.

Das Training über viele Epochen steht vor der Schwierigkeit, dass das trainierte Modell immer besser auf die spezifischen Trainingsdaten vorbereitet ist und Gefahr läuft, diese bloß auswendig zu lernen, anstatt sich auf generalisierbare Muster zu verlassen. Dieses Problem wird als Overfitting bezeichnet [14]. Ein Modell mit hohem Overfitting mag hervorragende Ergebnisse auf den Trainingsdaten liefern, wird es jedoch auf Daten getestet, die dem Modell nicht aus dem Training bekannt sind, so wird es wesentlich schlechtere Ergebnisse liefern. Underfitting stellt dabei das Gegenteil zum Overfitting dar, einer geringeren Genauigkeit auf den Trainingsdaten, als auf den Testdaten. Um Overfitting vorzubeugen, können verschiedene Maßnahmen ergriffen werden. Der Datensatz wird hierzu in zwei Teile aufgeteilt. Ein größerer Anteil der Daten wird regulär als Trainingsdatensatz eingesetzt, ein kleinerer Teil wird dem Modell im Trainingsprozess vorenthalten und als Testdatensatz genutzt. Das Modell kann so auf dem Testdatensatz überprüft werden. Diese Überprüfung kann entweder nach einer Zahl von Epochen wiederholt durchgeführt werden, oder bloß ein einziges Mal beim Abschluss des Trainings. Es ist anzustreben, die Erkennungsgenauigkeit des Modells für Trainings- und Testdatensatz in einem ähnlichen Rahmen zu halten. Ist eine klare Differenz in der Genauigkeit zwischen den Trainings- und Testdaten erkennbar, ist dieses over- oder underfittet und das trainierte Modell mag weniger nützlich für unbekannte Daten sein, als die Ergebnisse des Trainings vermuten lassen.

Nach dem beschriebenen Vorgehen wäre die Testgenauigkeit das Kriterium, um über die Güte eines trainierten Modells zu entscheiden. Hierbei existiert trotz der zusätzlichen

Maßnahme immer noch eine Voreingenommenheit, oder Bias, gegenüber der Wahl des besten Modellzustands. Um diesen Bias zu minimieren, kann der Trainingsdatensatz erneut geteilt werden, um einen Validierungsdatensatz zu erzeugen, der ebenso wie der Testdatensatz während dem Training vorenthalten wird [35]. So ergeben sich die drei Teildatensätze als Trainings-, Test- und Validierungsdaten. Ein Modell wird nach diesem Vorgehen in jeder Epoche auf den Trainingsdaten trainiert. Der Validierungsdatensatz wird nach jeder Epoche eingesetzt, um eine erste Einschätzung der Güte des Modells zu ermöglichen. Der Modellzustand mit der geschätzten, höchsten Güte wird nach Abschluss des Trainings final auf dem Testdatensatz geprüft. Die resultierende Testgenauigkeit des Modells liefert nun eine möglichst unvoreingenommene Einschätzung der allgemeinen Leistung eines Modells für den Datensatz mit Ausblick auf weitere, unbekannte Daten der gleichen Art.

Für diese Arbeit wird der Klassifikator jeweils für eine konkrete Hyperparameterkonfiguration trainiert. Das trainierende Modell wird nach jeder Epoche mit Hilfe der Validierungsdaten in seiner geschätzten Güte überprüft. Für einen einzelnen Klassifikator wird der Modellzustand mit der höchsten geschätzten Güte persistiert, um nach Abschluss des Trainings auf den Testdaten abschließend getestet zu werden. Der Ablauf und die Ergebnisse des Trainings müssen festgehalten werden, um anschließend evaluiert werden zu können. Dazu zählen alle Hyperparameterkonfigurationen, die für ein einzelnes Modell vorgenommen wurden, der Trainingsverlauf mit der Trainings- und Validierungsgenauigkeit über den Verlauf der Epochen und die abschließende Testgenauigkeit des Modells. Hierdurch soll sichergestellt werden, dass der erzielte Modellzustand replizierbar und sinnvoll interpretierbar ist.

3.1.5 Interpretation/Evaluation

Dem KDD Prozess nach werden im Interpretationsschritt die erkannten Muster, die durch das Data Mining entstanden sind, interpretiert und evaluiert. Durch die Interpretation der Ergebnisse sollen nützliche Informationen gewonnen werden. Sollte dies nicht erfolgreich gelungen sein, so kann der KDD Prozess, wie beschrieben, wiederholt werden.

Im Zuge dieser Arbeit sollen die entstandenen Klassifikationsmodelle betrachtet und eingeschätzt werden. Hierbei unterteilen sich die Experimente in zwei Gruppen, die zuerst getrennt voneinander untersucht werden. Diese unterteilen sich in die zwei Versionen des Datensatzes. Zuerst werden die Experimente pro Gruppe in ihrer Gesamtheit betrachtet

und untersucht, welche Auswirkungen die untersuchten Hyperparameter im Allgemeinen hatten. Darauf folgend wird eine Auswahl der entwickelten Modelle im Einzelnen evaluiert. Der Trainingsverlauf wird betrachtet, um zu überprüfen, ob Overfitting oder andere erkennbare Phänomene vorliegen. Außerdem wird die Konfusionsmatrix des Modells auf den Testdaten betrachtet. Eine Konfusionsmatrix stellt die vom Modell vorhergesagten und die tatsächlichen Labels des Datensatzes gegenüber [45]. Hierdurch können die häufigsten und seltensten Verwechselungen von Kategorien aufgezeigt werden, die einen Einblick in die Vorgehensweise des konkreten Modells bieten. Das Ziel der Betrachtung einzelner, ausgewählter Modelle ist es, eine Einschätzung geben zu können, wo die Stärken und Schwächen des Modells in der Erkennung der Kategorien liegen. Hierdurch kann detailliert die mögliche Leistung auf neuen Datenbeständen der gleichen Art eingeschätzt werden. Anschließend werden die einzeln betrachteten Modelle einerseits pro verwendetem Datensatz und andererseits datensatzübergreifend gegenübergestellt. So sollen übergreifende Verhaltensweisen erkennbar werden, die eine allgemeine Einschätzung ermöglichen, wie sich LSTM Modelle für den gewählten Datensatz verhalten. Außerdem wird ersichtlich, welche Maßnahmen am effektivsten wären, um in einer Fortführung der Experimente die Ergebnisse weiter zu verbessern.

Zusätzlich zu dem Vergleich der entstandenen Klassifikationsmodelle untereinander, können ebenso die in Abschnitt 2.7 vorgestellten Klassifikatoren hinzugezogen werden. Hierdurch können die Ergebnisse dieser Arbeit und die Relevanz von LSTMs im Allgemeinen für den Datensatz im Kontext eingeordnet und verglichen werden, wo Gemeinsamkeiten oder Unterschiede zwischen den Ergebnissen liegen.

3.2 Zusammenfassung der Anforderungen an die experimentelle Umgebung

Die experimentelle Umgebung erfolgt nach dem KDD Prozess. Für den KDD Prozess müssen fünf Schritte umgesetzt werden. Datenselektion, Vorverarbeitung der Daten, Transformation der Daten, Training des Machine Learning Verfahrens, sowie die Interpretation und Evaluation der Ergebnisse. Die ersten drei Schritte sind bereits zum größten Teil durch den gewählten Datensatz umgesetzt, weshalb die eigene Umsetzung entfällt. Es bleibt lediglich die Transformation des Datensatzes in eine Form, die vom gewählten Machine Learning Verfahren genutzt werden kann. Dabei muss einerseits das

Format der Features und Labels beachtet werden, andererseits die Aufteilung des gesamten Datensatzes in Trainings-, Validierungs- und Testdaten.

Der Fokus dieser Arbeit liegt auf dem Machine Learning und Evaluationsschritt. Für den Machine Learning Schritt muss der zu trainierende Klassifikator die transformierten Daten zusammen mit einer Konfiguration von Hyperparametern entgegennehmen. Damit soll anschließend ein Modell definiert und auf den Trainingsdaten trainiert werden. Nach jeder Trainingsepoche wird das Modell mit den Validierungsdaten vorläufig eingeschätzt. Sobald der Trainingsprozess abgeschlossen ist, soll das entstandene Modell auf den Testdaten überprüft werden. Das resultierende Ergebnis muss persistiert werden, um anschließend evaluiert werden zu können.

Der gesamte Prozess soll für verschiedene Hyperparameter beliebig oft wiederholbar sein. Dadurch muss vorher ein Suchraum von möglichen Hyperparameterkonfigurationen definierbar sein, der über die verschiedenen Trainings untersucht wird. Außerdem soll der Prozess für beide Versionen des Datensatzes unabhängig voneinander anwendbar sein.

3.3 Eingesetzte Technologien

Für die programmatische Umsetzung der experimentellen Umgebung wurde die Programmiersprache Python gewählt [57] [46]. Für Python liegen eine Vielzahl von Machine Learning Bibliotheken vor, mit denen die Sprache allgemein den Standard für die Umsetzung von Machine Learning Anwendungen darstellt. Diese Arbeit nutzt für die benötigten Machine Learning Methoden, wie die Erstellung eines LSTM Netzwerks, die Bibliothek TensorFlow [18]. Zusätzlich wird die Bibliothek Keras [32] als High Level API zu TensorFlow ergänzend genutzt, um die Nutzung der Funktionen zu vereinfachen.

Für die benötigten Datentransformationen vor dem Training wird unterstützend die Bibliothek numpy eingesetzt, die diverse mathematische Funktionen und Listenoperationen zur Verfügung stellt. Zur programmatischen Erstellung von Tabellen und Graphen für die Evaluation wird die Bibliothek matplotlib genutzt.

Zusätzlich soll das Training der Modelle in einer virtuellen Umgebung stattfinden. Hierfür wurde Docker [15] als Technologie gewählt. Hiermit können virtuelle Container definiert werden, die die Quellcodedateien und den Datensatz beinhalten. Die Ergebnisse des Trainings können außerhalb des Containers durch die Angabe eines Volumes persistiert werden.

4 Evaluation

Nachdem in Kapitel 3 beschrieben wurde, nach welcher Vorgehensweise die Experimente durchgeführt werden sollen, evaluiert dieses Kapitel die entstandenen Ergebnisse. Dazu wird zuerst die Ausführung der Experimente zusammengefasst. Danach werden die besten Modelle pro Netzwerktiefe für sich und untereinander evaluiert. Zum Schluss werden die erlangten Ergebnisse den Ergebnissen anderer Arbeiten gegenübergestellt und im Kontext eingeordnet.

4.1 Ablauf der Experimente

Die Experimente werden in drei Schritten zusammengefasst. Im ersten Schritt wird beschrieben, welche maßgeblichen Entscheidungen in der Umsetzung der experimentellen Umgebung getroffen und wie die Trainingszyklen aufgebaut wurden. Im zweiten Schritt wird in größerem Detail beschrieben, nach welchem Prozess der Suchraum an Hyperparameterkonfigurationen festgelegt wurde. Im dritten Schritt werden zuletzt die Ergebnisse in ihrem Umfang pro Experiment, den erzielten Testgenauigkeiten und der Dauer der Trainings präsentiert.

4.1.1 Durchführung

Die experimentelle Umgebung wurde vollständig innerhalb eines Python Projektes angelegt, um eine Virtualisierung durch einen Docker Container möglichst einfach zu gestalten. Zur Persistierung von und zum Zugriff auf Daten wurde sich für eine Verwaltung mittels eines lokalen Dateisystems entschieden, auf den ein Docker Container durch ein Volume ohne Probleme zugreifen kann. Dieses wurde außerdem dadurch begründet, dass der Datensatz selbst in Form von Textdateien zur Verfügung steht und mit einer Größe von 201 MB über beide Versionen keine Probleme beim Einlesen oder Zwischenspeichern im Hauptspeicher verursacht.

Für das Einlesen des Datensatzes mussten zuvor die vorhandenen Textdateien der Features angepasst werden. Dies wurde dadurch notwendig, dass zur optischen Darstellung der Daten unterschiedlich viele Leerzeichen zwischen den Features eingefügt wurden. Es wurde händisch eine gleichmäßige Form mit einzelnen Leerzeichen erzeugt. Eine alternative Option wäre es gewesen, die korrekte Form beim Einlesen der Dateien programmatisch umzusetzen. So wäre der Transformationsschritt jedoch bei jedem Einlesen der Dateien erneut erforderlich gewesen.

Nachdem die Features und Labels eingelesen wurden, wurden sie direkt in ein korrektes Format transformiert. In dieser Arbeit wurden Modelle durch ein mehrschichtiges LSTM Netzwerk aus der Layers Bibliothek von Keras umgesetzt. Dazu müssen die Features und Labels des Datensatzes in Form eines dreidimensionalen Arrays an ein Modell übergeben werden. Die drei Dimensionen umfassen in Reihenfolge die Anzahl an Samples, die Anzahl an Zeitschritten pro Sample und die Größe des Featurevektors pro Zeitschritt. Ein Sample bezeichnet dabei eine beliebige Anzahl an Zeitschritten, die einem Modell im Training sequentiell übergeben werden, ohne den Zustand des Modells zurückzusetzen. Die Größe der Samples, oder Sample Size, entscheidet somit über die Länge des Gedächtnisses des LSTMs, bevor dieses für das nächste Sample zurückgesetzt wird. Die Sample Size wurde als Hyperparameter für das Einlesen der Daten festgelegt, um über mehrere Trainings dynamisch festlegbar zu sein. Die Labels wurden zusätzlich zur oben beschriebenen Transformation durch ein One-Hot-Encoding [8], welches in Abschnitt 3.1.3 beschrieben wurde, transformiert. Es wurde außerdem ein Mechanismus entwickelt, der eine Überlappung von 50% für die Zeitschritte der Samples erzeugt, um die Menge der trainierbaren Daten zu erhöhen, ohne die Daten zu verändern. Dieser Mechanismus wurde jedoch schlussendlich nicht genutzt, da keine Verbesserung des Trainingsergebnisses festgestellt werden konnte, sondern bloß ein erhöhter zeitlicher Aufwand pro Epoche.

Die Definition eines LSTM Netzwerks wurde dynamisch umgesetzt, um unter anderem eine beliebige Anzahl von Hidden Layern zu erlauben. Ein LSTM Layer beinhaltet dabei die Menge an Neuronen und die Eingabeform, die der Form der transformierten Features entspricht. Hinter jedem LSTM Layer wurde außerdem ein Dropout Layer hinzugefügt, um Overfitting zu vermeiden [55]. Hinter den LSTM Layern wurde ein Dense Layer eingesetzt, um das Ergebnis des Modells auf die korrekte Form der Labels mit 6 bzw. 12 Gewichtungen zu bringen. Dabei handelt es sich um ein einfaches neuronales Netz, das als Output Layer des Modells dient. Das Modell wurde anschließend mit Categorical Cross-Entropy [40] als Loss Funktion, dem Adam Optimizer [33] und der Genauigkeit des Modells als Metrik kompiliert. Für den Adam Optimizer ist eine dynamische Lear-

ning Rate und Decay Rate konfigurierbar. Die Decay Rate stellt dabei eine zusätzliche Funktion für den Adam Optimizer bereit, bei der die Learning Rate im Verlauf der Epochen verkleinert wird. Für die Ausführung des Modells wurde eine variable Batch Size hinzugefügt, die die Anzahl an trainierten Zeitschritten festlegt, nach der die Gewichtungen des Modells aktualisiert werden. Die Anzahl an Hidden Layern, die Menge von Neuronen pro Hidden Layer, die Learning Rate und Decay Rate des Adam Optimizers, die Rate des Dropouts und die Batch Size wurden als Hyperparameter aufgenommen.

Für den Validierungsdatensatz wurden 10% der Trainingsdaten gewählt, die dem Modell während dem Training vorenthalten wurden. Die Samples beider Datensätze wurden anschließend zufällig angeordnet. Für ein Modell, das dem oben beschriebenen Aufbau folgt, wurden außerdem einige zusätzliche Funktionen angefügt, die das Training optimieren und aufzeichnen. Zunächst wurde ein Early Stopping Mechanismus [47] hinzugefügt, der die Entwicklung der Genauigkeit eines Modells überwacht. Sollte ein Modell sich innerhalb einer festgelegten Anzahl an Epochen nicht weiter verbessern, so wurde das Training vorzeitig beendet. Durch Experimentieren wurde die Toleranzschwelle auf 50 Epochen festgelegt. Ein Modell, dass zusätzlich nach den ersten 50 Epochen unter 92,0% Validierungsgenauigkeit verbleibt, wurde ebenso verfrüht beendet. Diese zusätzliche Bedingung wurde ergänzt, da erste Versuche zeigten, dass Modelle unter dem Schwellenwert auch bei einem langen, weiteren Training nicht in der Lage waren, herausragende Ergebnisse zu erreichen. Als nächstes wurde ein Mechanismus eingesetzt, der ein trainierendes Modell in seiner höchsten, geschätzten Güte durch die Validierungsgenauigkeit, als binäre Datei persistierte. Dieser Modellzustand wurde nach Abschluss des Trainings eingesetzt, um die Genauigkeit auf den Testdaten zu ermitteln. Außerdem wurde der gesamte Verlauf des Trainings in einer .csv Datei gespeichert, um im Nachhinein ausgewertet werden zu können. Analog wurden alle relevanten Informationen, wie die Länge des Trainings, die erzielte Test- und Validierungsgenauigkeit und die Hyperparameter des Modells in einer .json Datei zu jedem Modell gespeichert.

Um die Kontrolle über bereits trainierte Modelle und das Auswerten der Ergebnisse automatisieren zu können, wurde eine gemeinsame Referenzdatei angelegt. In dieser wurden die Bezeichner der Modelle und die wichtigsten Informationen, wie die erzielte Genauigkeit und die untersuchte Hyperparameterkonfiguration tabellarisch aufgezeichnet. Über den Modellbezeichner konnten die zugehörigen Dateien des Modells ausfindig gemacht werden. Außerdem konnte so sichergestellt werden, dass eine bereits untersuchte Hyperparameterkonfiguration kein zweites Mal untersucht wurde.

Um das Trainieren von Modellen mit unterschiedlichen Hyperparametern zu automatisieren, wurde vor dem Start des ersten Trainings eine Datei erzeugt, die alle möglichen Kombinationen an Hyperparametern in einem Suchraum zusammenfasst. Es existieren verschiedene Vorgehensweisen, um aus dem Suchraum eine Hyperparameterkonfiguration auszuwählen, mit der ein Modell trainiert werden soll. Die einfachste Vorgehensweise wäre Grid Search. Hierbei würde für das erste Modell die erste Hyperparameterkonfiguration des Suchraums ausgewählt werden, für das zweite Modell die zweite Hyperparameterkonfiguration und entsprechend für alle weiteren Modelle. Bei großen Suchräumen, die zu viele Hyperparameterkonfiguration besitzen, so dass nur ein Teil von ihnen untersucht werden kann, hat Grid Search den Nachteil, dass nicht die volle Breite an Möglichkeiten untersucht wird, da dies von der Sortierung des Suchraums abhängig ist [5]. Aus diesem Grund wurde in dieser Arbeit stattdessen nach dem Random Search Verfahren entschieden, welche Hyperparameterkonfiguration für ein Modell gewählt wurde. Dabei wurde für jedes neu zu trainierende Modell eine zufällige Hyperparameterkonfiguration bestimmt, um eine umfassendere Untersuchung zu erlauben. Auch wenn weitere Verfahren neben Random Search existieren, die noch effizienter im Finden von optimalen Hyperparametern sind, sind diese ebenso mit erheblich höherem Umsetzungsaufwand verbunden, weshalb sie in dieser Arbeit nicht näher betrachtet wurden.

Mit Hilfe des Suchraums der zu untersuchenden Hyperparameterkonfigurationen und der Referenzdatei der bereits abgeschlossenen Trainings wurde eine Schleife definiert, die sequentiell neue Hyperparameterkonfigurationen auswählt, aus ihnen LSTM Modelle definiert, diese trainiert und anschließend testet. Um eine Parallelisierung von Trainings zu ermöglichen, wurden diese in eigenen Threads gestartet. Der Zugriff auf die gemeinsame Referenzdatei stellte hierbei kein Problem dar, da jedes Modell bloß eine einzelne Zeile in einem einzigen Schreibaufruf hinzufügte. Im Verlauf des Trainings zeigte sich jedoch, dass die gewählte Bibliothek Keras in Kombination mit einem statusbehafteten Modell, wie dem LSTM, den Kontext, der für das Training erstellt wurde, nach jedem trainierten Modell neu starten musste. Wurde unter diesen Umständen versucht, ein zweites Modell im gleichen Kontext zu erzeugen, entstanden Fehler. Diese kamen dadurch vor, dass festgelegte, interne Parameter für vorherige Modelle nicht überschrieben werden konnten. Um dieses Problem zu umgehen, wurde aus der Schleife heraus für jedes einzelne Training der Kontext neu gestartet. Außerdem zeigte sich bei dem Einsatz einer GPU statt einer CPU, dass die Parallelisierung von Threads nicht möglich war. Dies war dadurch bedingt, dass zu jedem Zeitpunkt nur ein einzelner erzeugter Kontext Zugriff auf die GPU hatte. Versuchte ein zweiter Kontext, während der Laufzeit des ersten Kontextes,

auf die GPU zuzugreifen, so kam es zum Konflikt und der erste Kontext wurde beendet. Durch diese Einschränkungen konnten Trainings bloß sequentiell stattfinden.

Es wurde entschieden, ein einzelnes Experiment als die Findung möglichst optimaler Hyperparameter für ein Netzwerk mit einer festen Tiefe von Hidden Layern zu einem der beiden Versionen des Datensatzes zu definieren. Dazu wurde jedes Experiment mit einer gleichmäßigen Menge von Hyperparameterkonfigurationen untersucht, um eine ausgewogene Basis für weitere Vergleiche zu erschaffen. Außerdem wurde, für eine ausgewogene Vergleichsbasis, vor der Durchführung der eigentlichen Experimente eine Reihe von händisch ausgewählten Hyperparameterkonfigurationen untersucht, die anschließend für den finalen Suchraum eingegrenzt wurden. So sollte sichergestellt werden, dass das Hyperparameter Tuning möglichst effizient abläuft und alle Experimente auf der gleichen Basis erfolgen. Im nächsten Abschnitt wird die Selektion der Hyperparameterkonfigurationen im Detail beschrieben.

4.1.2 Ermittlung des Suchraums der Hyperparameter

Wie in Abschnitt 4.1.1 beschrieben wurde, wurde zunächst eine Menge von Hyperparametern ausgewählt, die beim Training eines Modells einstellbar waren, um die Leistung zu optimieren. Als erstes war die Tiefe des Netzwerks mit der Anzahl an Hidden Layern einstellbar. Dieser Hyperparameter wurde zusätzlich zur Unterteilung der einzelnen Experimente eingesetzt. Als nächstes legte der Neurons Hyperparameter fest, wie viele Neuronen pro Hidden Layer eingestellt werden sollen. Dies musste pro Hidden Layer unabhängig voneinander möglich sein. Außerdem war die Batch Size konfigurierbar, um die Geschwindigkeit des Trainings zu optimieren. Die Sample Size war dynamisch einstellbar, um die maximale Gedächtnisgröße des Modells während des Trainings zu kontrollieren. Hiermit konnte überprüft werden, welche Auswirkungen, ob positiv oder negativ, der zeitliche Kontext für die Modelle hatte. Das Dropout wurde als Maßnahme gegen Overfitting eingesetzt. Die Learning Rate war ein prominenter Hyperparameter, der bei neuronalen Netze im Allgemeinen meist die größte Auswirkung auf die Leistung eines Modells hat [60]. Das Decay wurde zusätzlich als Option des gewählten Optimizers Adam eingesetzt. Eine Auflistung der gewählten Hyperparameter ist in Tabelle 4.1 zu sehen.

Hyperparameter
Layer
Neurons
Batch Size
Sample Size
Dropout
Learning Rate
Decay

Tabelle 4.1: Liste der Hyperparameter

Um alle Experimente auf der gleichen Basis durchzuführen, wurden die zu untersuchenden Hyperparameterkonfigurationen im Vorfeld selektiert. Umfangreiche Experimente von Greff et. al. [20] zeigten, dass die Hyperparameter eines LSTMs so gut wie unabhängig zueinander sind. Dadurch ist es möglich die Konfigurationen einzelner Hyperparameter individuell anzupassen, ohne dass gravierende unvorhergesehene Effekte in der Leistung eintreten. Deshalb wurden diverse Konfigurationen für alle Hyperparameter untersucht und die besten anschließend dem Suchraum der Experimente hinzugefügt. Im Folgenden werden die untersuchten Hyperparameterkonfigurationen pro Hyperparameter betrachtet und diskutiert, welche Hyperparameter nach diesen ersten Untersuchungen eine potentiell große oder kleine Auswirkung auf die Leistung der Modelle haben. Nichtsdestotrotz ist diese Betrachtung nur zur Unterstützung in der Wahl der Hyperparameterkonfigurationen gedacht und nicht als primäres Kriterium. Da in den Untersuchungen die Zuordnung von Hyperparameterkonfigurationen und ihrer erzielten Genauigkeit nicht isoliert pro Hyperparameter in der gesamten Konfiguration betrachtet werden kann, besteht stets die Möglichkeit, dass zufällige Konstellationen das Ergebnis für einzelne Konfigurationen beeinflussen. Um der gegenseitigen Beeinflussung der Hyperparameter in ihren Ergebnissen entgegenzugehen, wurde der Durchschnitt über die Ergebnisse einzelner, gleicher Hyperparameterkonfigurationen gebildet. Aus zeitlichen Gründen wurden außerdem nur Modelle mit einem Hidden Layer betrachtet. Zusätzlich wurden die Modelle nur auf der ersten Version des Datensatzes untersucht. Da der zweite Datensatz lediglich eine Erweiterung des ersten Datensatzes ist, konnte angenommen werden, dass die Erkenntnisse der Untersuchungen für beide Versionen gleichwertig anwendbar sind. Nachdem die Hyperparameter im einzelnen betrachtet wurden, wird der entstandene Suchraum zusammengefasst, der sich aus den Untersuchungen ergeben hat.

Wie in Tabelle 4.2 einsehbar, wurden 13 verschiedene Konfigurationen für die Neuronen untersucht. Die linke Spalte (**Konfig.**) zeigt die jeweils untersuchte Konfiguration. Die mittlere Spalte (#) zeigt die Menge an Trainings, die mit der Konfiguration untersucht wurden. Die rechte Spalte (∅ (%)) zeigt das durchschnittliche Ergebnis für eine Konfigurationen über alle zugehörigen Trainings.
Durch diese Betrachtungsweise soll eine möglichst klare Interpretation der Ergebnisse ermöglicht werden. Im ersten Schritt wurden die Zweierpotenzen zwischen 64 und 1024 betrachtet. Es wurde bereits in den ersten, wenigen Untersuchungen deutlich, dass die Komplexität des Datensatzes verhältnismäßig gering ist, da selbst bei nur einem Hidden Layer wenige Hundert Neuronen bereits die besten Ergebnisse erzielen. 1024 Neuronen wurden lediglich mit einem Modell überprüft, da dieses bereits äußerst langsam

Neurons		
Konfig.	**#**	**∅ (%)**
200	130	95,62
300	123	95,53
150	124	95,41
250	123	95,11
100	134	94,86
350	68	94,31
256	155	93,49
192	162	93,35
384	127	93,30
128	48	91,91
64	16	90,90
512	14	86,90
1024	1	32,44

Tabelle 4.2: Erste Untersuchung der Neuronen

im Trainingsvorgang war und mit einem massiven Abstand mit 32,44% das schlechteste Ergebnis aller getesteten Modelle hervorbrachte. Ein Modell mit 1024 Neuronen besitzt augenscheinlich weit mehr Parameter, als es für den Datensatz nötig war, was zu einer starken Unsicherheit des Modells führt. Im zweiten Schritt wurden feingranularere Intervalle in der vielversprechendsten Reichweite von Neuronen hinzugefügt. Schlussendlich zeigte sich, dass Modelle mit einer Anzahl von zwischen 100 und 300 Neuronen, die besten Ergebnisse hervorbrachten. Außerdem wurde gezeigt, dass weniger als 100 Neuronen und wesentlich mehr Neuronen als 300 zu teils starken Leistungseinbußen führen. Für den Suchraum der Experimente wurden Konfigurationen zwischen 100 und 300 in Schritten von 50 Neuronen gewählt.

Im Vergleich zu der Anzahl an Neuronen zeigte die Batch Size wesentlich weniger Auswirkungen auf die Leistung des Modells. Wie in Tabelle 4.3 gezeigt wird, variieren die durchschnittlichen Leistungen bloß bei einer Batch Size von 50 oder mehr erkennbar. Jedoch wirkte sich eine kleinere Batch Size positiv auf die Trainingsgeschwindigkeit aus. Aus diesem Grund wurde als Kompromiss aus den untersuchten Leistungen und der Trainingsgeschwindigkeit eine Batch Size von 20 für die Experimente gewählt.

Für die Sample Size ist in Tabelle 4.4 zu erkennen, dass generell größere Gedächtnisse bessere Leistungen erzielen, als kleinere Gedächtnisse. Durch dieses Verhältnis ergibt sich bereits ein erster Einblick in die Auswirkungen des zeitlichen Kontext und dass

Batch Size		
Konfig.	**#**	**Ø (%)**
30	321	95,75
10	108	95,54
20	95	95,48
25	141	94,64
40	86	94,02
50	250	93,59
75	186	91,59
100	38	90,02

Tabelle 4.3: Erste Untersuchung der Batch Size

Sample Size		
Konfig.	**#**	**Ø (%)**
200	66	95,82
150	197	95,45
110	61	95,39
130	64	94,96
120	73	94,58
140	69	94,47
70	24	94,46
90	153	94,15
100	334	94,12
75	20	93,48
80	145	92,25
50	12	90,33
25	7	75,27

Tabelle 4.4: Erste Untersuchung der Sample Size

dieser scheinbar einen positiven Einfluss auf die Leistung des Modells besitzt. Es werden jedoch mehr Untersuchungen benötigt, um diese Erkenntnis zu konkretisieren. Dies ist daran zu erkennen, dass kein striktes Verhältnis zwischen einer höheren Sample Size und einem besseren Ergebnis vorliegt. Gleichzeitig bedeutet eine hohe Sample Size ebenso eine geringe Anzahl an Samples. Da der Datensatz jedoch allgemein nur wenige Daten besitzt, ist dieses Verhältnis wichtig, um das Training nicht ungewollt zu verschlechtern. Um die Größe der Samples und der Anzahl an Samples möglichst ausgewogen zu halten, wurde für den Suchraum eine Sample Size zwischen 50 und 125 in Schritten von 25 Zeitschritten gewählt.

In Tabelle 4.5 ist zu erkennen, dass zwischen den untersuchten Konfigurationen für das Dropout bloß ein Leistungsunterschied von 1% liegt. Dies ist jedoch nicht verwunderlich, da das Dropout primär zur Reduzierung von Overfitting eingesetzt wird und nicht zur strikten Leistungsverbesserung. Auch wenn die durchschnittlich besten Ergebnisse bei einem Dropout von 0.1 erzielt wurden, wird diese Konfiguration nicht für den Suchraum ausgewählt. Der Grund ist, dass die Leistung zwischen den Epochen eines solchen Modells starke Einbußen erleidet, die zunächst neu erlernt werden müssen, da zu radikal gelernte Gewichtungen entfernt werden. Dies erhöht einerseits die Gesamtmenge an Epochen, die ein Modell zum Trainieren benötigt und andererseits variieren die Ergebnisse sehr stark und sind damit nicht verlässlich. Schlussendlich wurden 0.01, 0.001 und 0.0001 als mögliche Konfigurationen für das Dropout im Suchraum gewählt.

Dropout		
Konfig.	**#**	**∅ (%)**
0.1	77	94,49
0.001	362	94,43
1e-05	265	94,42
0.0	67	94,41
0.0001	259	94,12
0.01	195	93,49

Tabelle 4.5: Erste Untersuchung des Dropouts

Learning Rate		
Konfig.	**#**	**∅ (%)**
0.0025	125	96,00
0.001	337	95,76
0.005	257	95,61
0.01	177	95,28
0.0005	86	94,29
0.025	62	94,25
0.0001	15	88,05
0.05	162	87,89
0.1	4	48,49

Tabelle 4.6: Erste Untersuchung der Learning Rate

Wie aus allgemeinen Erkenntnissen zu neuronalen Netzen zu vermuten war, zeigt sich, dass die Learning Rate einen großen Einfluss auf die Leistung der Modelle besitzt. Über alle Hyperparameter hat die Learning Rate mit 96% die höchste, durchschnittliche Leistung erzielt. Dies kann in Tabelle 4.6 eingesehen werden. Eine Learning Rate von 0.05 oder mehr hat starke negative Auswirkungen auf das Modell. Gleiches ist für eine Learning Rate von 0.0001 erkennbar, auch wenn in diesem Fall wesentlich weniger durchgeführ-

te Untersuchungen vorliegen, was das Ergebnis durch Zufall verfälschen mag. Am Ende wurden 0.005, 0.0025, 0.001 und 0.0005 als Konfigurationen der Learning Rate für den Suchraum gewählt.

Decay		
Konfig.	**#**	**∅ (%)**
0.00025	140	95,60
0.0025	126	95,54
0.0005	54	94,43
0.005	56	94,33
0.0	138	94,19
0.0001	343	93,99
0.001	324	93,91
0.01	40	92,35
0.1	4	62,42

Tabelle 4.7: Erste Untersuchung des Decays

Für das Decay ist in Tabelle 4.7 zu erkennen, dass ein kleineres Decay durchaus einen positiven Effekt besitzt. Ein zu hohes Decay zeigt jedoch auch einen sehr negativen Einfluss. Gleichzeitig ist das durchschnittliche Ergebnis mit keinem Decay (0.0) nur etwas über 1% schlechter, als das beste Ergebnis von 0.00025. Die Ergebnisse folgen ähnlich zur Sample Size keinem klaren Verhältnis zum Ergebnis. Aus diesem Grund werden nur kleinere Werte für den Suchraum übernommen. Es wurden 0.0, 0.005, 0.0025, 0.0005 und 0.00025 als Konfigurationen für den Suchraum gewählt.

Hyperparameter	**Konfigurationen**
Neuronen	100, 150, 200, 250, 300
Sample Size	50, 75, 100, 125
Dropout	0.01, 0.001, 0.0001
Learning Rate	0.005, 0.0025, 0.001, 0.0005
Decay	0.0, 0.005, 0.0025, 0.0005, 0.00025

Tabelle 4.8: Gewählte Hyperparameterkonfigurationen

Nachdem nun für jeden einzelnen Hyperparameter mögliche Konfigurationen untersucht und unter ihnen die vielversprechendsten ausgewählt wurden, ergibt sich in Summe der finale Suchraum, der für alle folgenden Experimente eingesetzt wird. Die gewählten Hyperparameterkonfigurationen werden in Tabelle 4.8 zusammengefasst. Hieraus ergibt sich

Netzwerktiefe	Größe
1	1500
2	7500
3	37.500
4	187.500
5	937.500

Tabelle 4.9: Größe des Suchraums von Hyperparameterkonfigurationen

pro Tiefe des Netzwerks eine spezielle Anzahl an möglichen Zusammenstellungen. Selbst mit einer starken Reduzierung der zu untersuchenden Konfigurationen, ergibt sich für tiefere Netzwerke eine große Menge an möglichen Zusammenstellungen. Wie in Tabelle 4.9 einzusehen ist, wächst der Suchraum von 1500 möglichen Hyperparameterkonfigurationen für ein Netzwerk mit einem Hidden Layer bereits auf 7500 für zwei Hidden Layer. Allgemein steigert sich der zu untersuchende Suchraum pro weiterem Hidden Layer um einen Faktor von 5 zum nächst kleineren Netzwerk. Da ein tieferes Netzwerk ebenso einen längeren zeitlichen Aufwand pro trainierendem Modell besitzt, wird es schnell in absehbarer Zeit unmöglich, einen Suchraum vollständig zu untersuchen. Hierzu ist erkennbar, warum ein effizientes Auswahlverfahren wie Random Search, das unter 4.1.1 beschrieben wurde, ein wichtiger Faktor für das Hyperparameter Tuning ist.

4.1.3 Ergebnisse

Nachdem in Abschnitt 4.1.2 zunächst der Suchraum für die Experimente aus möglichen Hyperparameterkonfigurationen ermittelt wurde, werden in diesem Abschnitt die Ergebnisse der durchgeführten Experimente zusammengefasst. Insgesamt wurden im Zuge dieser Arbeit 10.270 Modelle über alle Experimente trainiert. In Tabelle 4.10 wird aufgezeigt, wie sich die Gesamtzahl an trainierten Modellen in die einzelnen Experimente aufteilt. Es wurden Modelle mit einer Netzwerktiefe von bis zu fünf untersucht. Dies wurde während dem Training festgelegt, nachdem ersichtlich wurde, bei welcher Tiefe die entwickelten Modelle die besten Testgenauigkeiten erzielen. Die 1225 trainierten Modelle für den ersten Datensatz mit einem Hidden Layer, die gesondert aufgefasst werden, sind dabei die Untersuchungen für den Suchraum. Für die Experimente beider Datensatzversionen mit einem Hidden Layer wurde der Suchraum vollständig untersucht, alle weiteren Experimente jedoch nur anteilig. Aus diesem Grund wurde die Anzahl der untersuchten

Hyperparameterkonfigurationen für die erste Datensatzversion möglichst angeglichen.

Datensatzversion 1 (2.6.1)		**Datensatzversion 2 (2.6.2)**	
Netzwerktiefe	**Anzahl Modelle**	**Netzwerktiefe**	**Anzahl Modelle**
1	1225 + 1500	1	1500
2	1185	2	837
3	1012	3	568
4	1034	4	207
5	1001	5	201
	1225 + 5732		3313

Tabelle 4.10: Aufteilung der trainierten Modelle pro Experiment

Bei der zweiten Datensatzversion zeigte sich jedoch, dass die Trainings oft eine größere Anzahl an Epochen benötigten, als die der ersten Version. Gleichzeitig erreichten die besten erzielten Testgenauigkeiten im direkten Vergleich geringere Ergebnisse. In Tabelle 4.11 wird dieses Verhältnis aufgezeigt. Aus diesem Grund wurden die Experimente der zweiten Datensatzversion kürzer gehalten und beendet, sobald ihre Leistung, über die jeweils besten Modelle gemessen, stagnierte. Alle Experimente wurden unter Einsatz einer GTX 1060 mit 6 GB VRAM und 32 GB RAM durchgeführt.

Um das allgemeine Resultat der durchgeführten Experimente zu diskutieren, zeigt Tabelle 4.11 eine Übersicht der einzelnen Experimente. Jedem Experiment wird die erzielte Testgenauigkeit im Minimum, Mittel und Maximum und analog die Anzahl der Trainingsepochen zugeordnet. Für die erste Datensatzversion werden bei einer Netzwerktiefe von zwei die besten Ergebnisse erzielt, mit einem ähnlichen Leistungsabfall zu einem und drei Hidden Layern. Die erzielte Genauigkeit sinkt bei jeder Vergrößerung der Netzwerktiefe. Dies lässt sich darauf zurückführen, dass der Datensatz im Vergleich sehr klein ist und nur eine geringe Netzwerkkomplexität nötig ist, um die erkannten Muster vernünftig abzubilden. Für die zweite Datensatzversion ist bereits die Netzwerktiefe von einem Hidden Layer in der Lage die besten Ergebnisse zu erzielen, wobei dies auch dadurch verursacht sein kann, dass für dieses Experiment mit Abstand die meisten Modelle untersucht wurden. Gleichzeitig ist jedoch der Verlust der Genauigkeit über die Vergrößerung der Netzwerktiefe gleichmäßig. Dies lässt darauf schließen, dass sich die Rangfolge der Netzwerktiefen auch mit weiteren Trainings für tiefere Modelle nicht groß verändern würde. Die weitere Unterteilung der Labels in der zweiten Version reduziert die Komplexität der

Datensatzversion 1 (2.6.1)						
	Genauigkeit (%)			Trainierte Epochen		
Netzwerktiefe	Min.	Median	Max.	Min.	Median	Max.
1	90,06	95,48	98,18	63	111,5	267
2	89,20	95,68	98,46	50	115	263
3	82,51	95,20	98,15	50	120	310
4	71,93	94,34	98,05	50	119	284
5	18,51	87,51	97,40	50	50	283
Datensatzversion 2 (2.6.2)						
	Genauigkeit (%)			Trainierte Epochen		
Netzwerktiefe	Min.	Median	Max.	Min.	Median	Max.
1	90,96	96,31	98,12	71	119	331
2	90,17	96,22	97,93	69	122	319
3	84,16	95,74	97,67	50	133	381
4	59,96	94,60	97,06	50	139	310
5	41,11	91,07	96,51	50	50	299

Tabelle 4.11: Ergebnisse pro Netzwerktiefe

Daten scheinbar genug, um noch kleinere Modelle zu favorisieren. Trotzdem sind die erreichten Genauigkeiten im Maximum geringer. Auch dies lässt sich mit hoher Sicherheit auf die Aufteilung der Labels zurückführen. Die weitere Reduzierung von Examples pro Label, bei einem bereits im Ursprung sehr kleinen Datensatz, führt zu einer stärkeren Unsicherheit der Modelle im Erkennen der Labels. Diese Beobachtung wird ebenso über den erhöhten Aufwand des Trainings, anhand der Menge an Epochen unterstützt.

4.2 Vergleich der Modelle

Nachdem die Experimente in Abschnitt 4.1 im Allgemeinen untersucht wurden und bereits erste Erkenntnisse über die Leistung und Eigenschaften der Modelle gesammelt werden konnten, werden nun die besten Modelle der einzelnen Experimente betrachtet. Die Modelle werden zunächst für sich vorgestellt und anschließend untereinander in ihren Stärken und Schwächen verglichen. Aus diesen Ergebnissen leiten sich die weitere Erkenntnisse über den Einsatz von LSTMs auf dem gewählten Datensatz ab.

Datensatzversion 1 (2.6.1)							
		Hyperparameter					
Tiefe	**%**	**N**	**B**	**S**	**DO**	**LR**	**DC**
1	98,18	200	20	75	0.001	0.001	0.0025
2	98,46	150, 300	20	75	0.001	0.001	0.00025
3	98,15	250, 150, 200	20	75	0.0001	0.001	0.0025
4	98,05	250, 100, 200, 300	20	75	0.01	0.001	0.0005
5	97,40	100, 200, 250, 250, 250	20	75	0.001	0.001	0.00025

Datensatzversion 2 (2.6.2)							
		Hyperparameter					
Tiefe	**%**	**N**	**B**	**S**	**DR**	**LR**	**DE**
1	98,12	250	20	100	0.001	0.001	0.0005
2	97,93	150, 250	20	100	0.0001	0.001	0.0
3	97,67	100, 100, 150	20	100	0.01	0.005	0.0005
4	97,06	150, 100, 250, 150	20	100	0.0001	0.0025	0.0005
5	96,51	200, 200, 150, 300, 200	20	125	0.01	0.001	0.005

Tabelle 4.12: Die besten Modelle und ihre Hyperparameter nach Netzwerktiefe

Tabelle 4.12 zeigt zusammenfassend die besten Modelle für jedes Experiment mit ihrer erzielten Testgenauigkeit und ihren Hyperparametern. Dazu wurden die Namen der Hyperparameter wie folgt abgekürzt: N = Anzahl Neuronen nach aufsteigendem Layer, B = Batch Size, S = Sample Size, DO = Dropout, LR = Learning Rate, DC = Decay.

Es sind pro Datensatzversion für die meisten Hyperparameter klare Tendenzen zu speziellen Hyperparametern zu erkennen. Die Schwankungen einzelner Modelle in der Auswahl der besten Hyperparameter für die zweite Version lassen sich dabei anteilig auf die geringere Anzahl an trainierten Modellen zurückführen.
Die Sample Size für die erste Version ist uniform bei 75 für alle Experimente. Wäre die Konfiguration am unteren oder oberen Ende, hätte die Möglichkeit bestanden, dass eine unaufgeführte Konfiguration noch bessere Ergebnisse erzielt hätte. Da die Konfiguration aber im unteren Mittelfeld der Möglichkeiten liegt, ist es sicher, dass es sich hierbei unter den gegebenen Möglichkeiten um die optimale Konfiguration handelt. Diese Erkenntnis zeigt gleichzeitig, dass die Modelle durchaus einen Nutzen aus der zeitlichen Abhängigkeit der Daten ziehen. Für die zweite Datensatzversion gelten die gleichen Erkenntnisse, jedoch mit 100 als dem oberen Mittelfeld der möglichen Sample Sizes. Diese Veränderung zwischen den Versionen lässt ebenso darauf schließen, dass die feinere Unterteilung der Labels ein größeres Verständnis der zeitlichen Übergange der Aktivitäten erfordert, wie bei Übergangsaktivitäten intuitiv zu vermuten wäre.

Die Learning Rate ist ebenso ein Hyperparameter mit einer klaren Tendenz pro Datensatzversion. Aus den verfügbaren Möglichkeiten scheint eine tendenziell kleinere Learning Rate vorteilhaft. Außerdem ist zu beobachten, dass über beide Versionen des Datensatzes mit 0.001 die gleiche Learning Rate als optimal festgestellt wurde. Dies entspricht ebenso der Intuition, da die Learning Rate einen abstrakteren Bezug auf die eingesetzten Daten besitzt, als noch die Sample Size.
Das Dropout und das Decay haben ebenso Tendenzen zu gewissen Konfigurationen, jedoch in geringerem Ausmaß als bei den vorherigen Hyperparametern. Hier mag es daran liegen, dass die Hyperparameter weniger entscheidend für die allgemeine Leistung der Modelle sind. Die Anzahl an Neuronen ist wiederum wesentlich weniger uniform und bietet wenig Einblick über bevorzugte Konfigurationen für den Datensatz. Würden weitere Experimente durchgeführt werden, die auf den Erkenntnissen der bisherigen aufbauen, so wäre dies der wichtigste Hyperparameter, der weiterer Untersuchung bedarf.

Nachdem nun die besten Modelle pro Experiment für ihre Hyperparameter verglichen wurden, werden sie in im Einzelnen im Detail beleuchtet. Dazu werden für jedes Modell zwei Abbildungen betrachtet. Zuerst der Trainingsverlauf mit der Entwicklung der Trainings- und Validierungsgenauigkeit über die Epochen. Außerdem die Konfusionsmatrix mit der Gegenüberstellung der vorhergesagten und tatsächlichen Labels für den Testdatensatz. Durch den Trainingsverlauf kann die Entwicklung des Modells genauer betrachtet werden, ob Over- oder Underfitting vorliegt und wie sicher das Modell im Allgemeinen seine Vorhersagen trifft. Die Konfusionsmatrix beleuchtet diesen letzten Punkt in mehr Detail für den Testdatensatz. So können die Stärken und Schwächen des Modells für die Vorhersagen konkretisiert werden.

4.2.1 Die besten Modelle für die erste Datensatzversion

Das beste Modell für eine Netzwerktiefe von einem Hidden Layer auf der ersten Version des Datensatzes erreichte eine Testgenauigkeit von 98,18% in der 78. Epoche. Insgesamt trainierte das Modell 129 Epochen. Abbildung 4.1 zeigt den zugehörigen Trainingsverlauf in der linken Abbildung und die Konfusionsmatrix in der rechten Abbildung.

Am Trainingsverlauf ist zu erkennen, dass sich die Trainings- und Testgenauigkeit stets auf einem ähnlichen Niveau bewegten. Das Modell erreichte mit der 40. Epoche eine Stagnation der Genauigkeit, von der aus nur noch wenig Zuwachs im weiteren Training

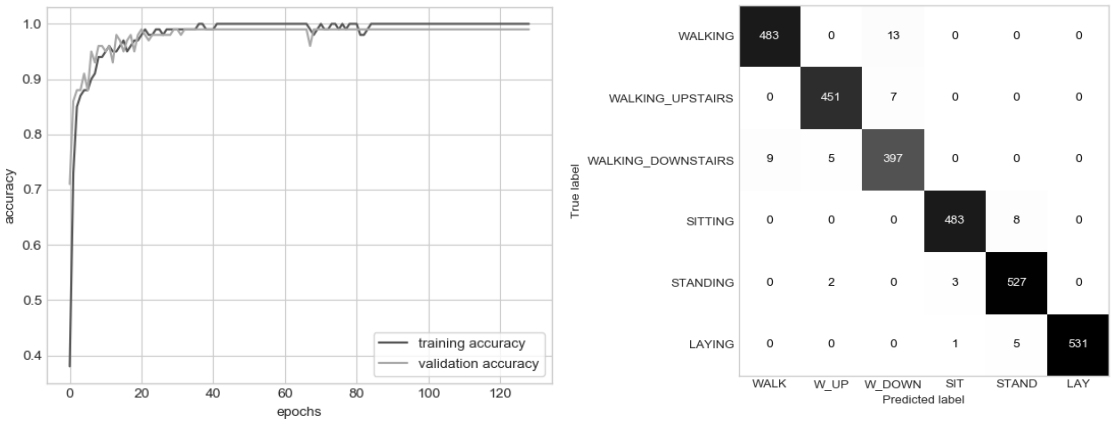

Abbildung 4.1: Das beste Modell mit einem Hidden Layer für Datensatzversion 1 mit einer Testgenauigkeit von 98,18%

stattfand. Bloß ein kurzzeitiger Einbruch der Trainingsgenauigkeit zwischen der 65. und 85. Epoche ist erkennbar. Die Validierungsgenauigkeit verschlechterte sich in dieser Phase nur kurz zu Beginn, jedoch verbesserte sie sich in den nächsten Epochen sofort wieder. Abgesehen davon war das Training im späteren Verlauf sehr stabil und die Differenz zwischen den beiden Genauigkeiten nur minimal. Das Modell scheint somit ein gutes, allgemeines Bild von den Daten erhalten zu haben und nur in einem geringen, akzeptablen Maß auf den Trainingsdaten overfittet zu sein.
Die Konfusionsmatrix zeigt, dass das Modell zwischen den drei dynamischen Aktivitäten WALKING, WALKING_UPSTAIRS und WALKING_DOWNSTAIRS die größte Verwirrung besitzt. Allgemein hat das Modell für diese Aktivitäten eine Tendenz, bei Unsicherheit bei den anderen beiden Aktivitäten die WALKING_DOWNSTAIRS Aktivität zu wählen. Gleichzeitig ist es jedoch auch am unsichersten in der Vorhersage der WALKING_DOWNSTAIRS Aktivität und verwechselt diese mit den anderen beiden dynamischen Aktivitäten. Dies lässt darauf schließen, dass die Abgrenzung der WALKING_DOWNSTAIRS Aktivität zu den anderen beiden dynamischen Aktivitäten für das Modell sehr gering ausfällt.
Mit wenigen Ausnahmen trennt das Modell jedoch sauber zwischen den drei dynamischen Aktivitäten und den drei statischen Aktivitäten SITTING, STANDING und LAYING. Bei den statischen Aktivitäten werden, bei falschen Vorhersagen des Modells, bloß SITTING und STANDING gewählt. Es wird nie eine Aktivität fälschlicherweise als LAYING vorhergesagt.
Allgemein ist das Modell sehr gut in der Lage, die Bewegungsaktivitäten vorherzusagen.

Die größte Schwäche des Modells ist eindeutig die hohe Unsicherheit bei der Abgrenzung der WALKING_DOWNSTAIRS Aktivität.

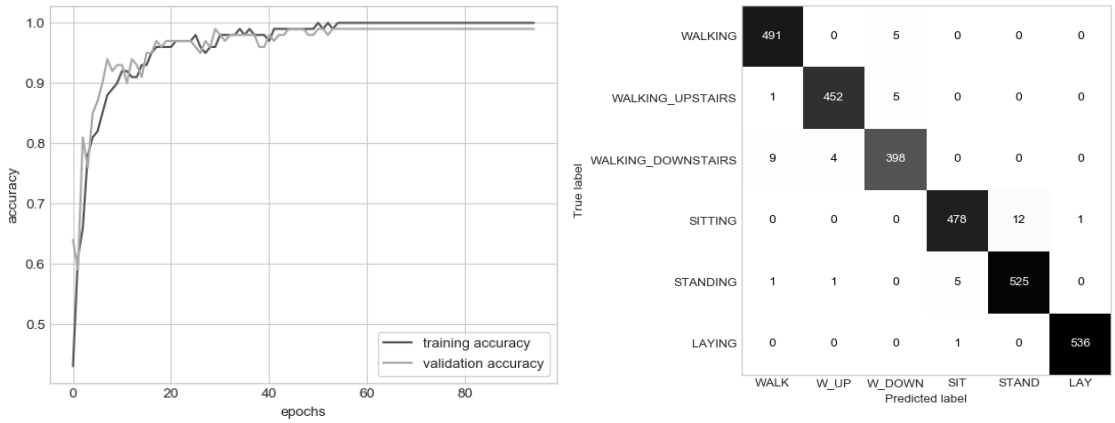

Abbildung 4.2: Das beste Modell mit zwei Hidden Layern für Datensatzversion 1 mit einer Testgenauigkeit von 98,46%

Das beste Modell für eine Netzwerktiefe von zwei Hidden Layern auf der ersten Version des Datensatzes erreichte eine Testgenauigkeit von 98,46% in der 44. Epoche und ist damit das beste Modell über alle Experimente. Insgesamt trainierte das Modell 95 Epochen. Abbildung 4.2 zeigt den zugehörigen Trainingsverlauf in der linken Abbildung und die Konfusionsmatrix in der rechten Abbildung.

Im Vergleich zum ersten Modell benötigt das beste Modell für zwei Hidden Layer ca. 15 Epochen länger, bis der Trainingseffekt zur 65. Epoche überwiegend stagniert. Nach diesem Punkt erleidet das Modell im weiteren Training jedoch keine weiteren Einbrüche und hält sich stabil. Es ist eine ähnliche Differenz mit minimalem Overfitting zum ersten Modell erkennbar.
Auch in der Konfusionsmatrix sind die gleichen Tendenzen in der Verwechselung von Bewegungsaktivitäten erkennbar. Die verhältnismäßig starke Unsicherheit bezüglich der WALKING_DOWNSTAIRS Aktivität bleibt bestehen, verringert aber die Anzahl an Verwechselungen bei denen das Modell fälschlicherweise WALKING_DOWNSTAIRS vorhersagte. Aus der größeren Unsicherheit bei den statischen Bewegungsaktivitäten ergab sich eine ähnliche Beobachtung, wie bei den dynamischen Aktivitäten, mit einer fokussierten Unsicherheit der SITTING Aktivität. Da es sich bei diesem Modell um die beste Version für alle Experimente handelt, kann die Unsicherheit bezüglich WAL-

KING_DOWNSTAIRS und SITTING als die grundsätzliche Schwäche der LSTM Netzwerke auf dem gegebenen Datensatz angenommen werden. Dies kann auch darauf zurückgeführt werden, dass für diese beiden Aktivitätsklassen die jeweils geringste Anzahl an Examples zur Verfügung steht, wenn zwischen den dynamischen und statischen Bewegungsaktivitäten unterschieden wird. Dies gilt sowohl für die Trainings- als auch für die Testdaten.

Das beste Modell für eine Netzwerktiefe von drei Hidden Layern auf der ersten Version des Datensatzes erreichte eine Testgenauigkeit von 98,15% in der 69. Epoche. Insgesamt trainierte das Modell 120 Epochen. Abbildung 4.3 zeigt den zugehörigen Trainingsverlauf in der linken Abbildung und die Konfusionsmatrix in der rechten Abbildung.

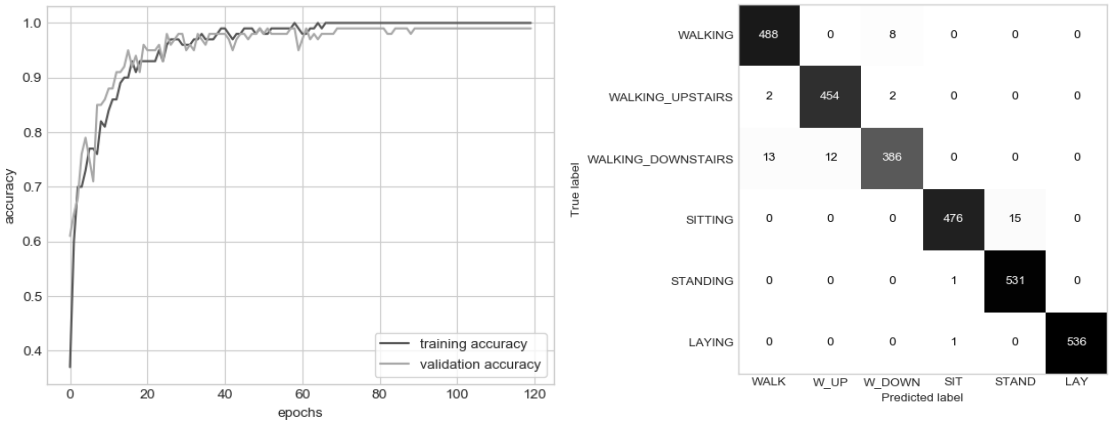

Abbildung 4.3: Das beste Modell mit drei Hidden Layern für Datensatzversion 1 mit einer Testgenauigkeit von 98,15%

Es ist erneut zu erkennen, dass auch dieses Modell im Vergleich zu den anderen beiden Modellen eine höhere Anzahl an Epochen benötigt, bevor der Trainingseffekt stagniert. In diesem Fall tritt die Stagnation zur 70. Epoche auf. Über diese ersten drei Beobachtungen ist bereits zu vermuten, dass ein direktes Verhältnis zwischen der benötigten Trainingslänge vor einer Stagnation und der Netzwerktiefe besteht. Die restlichen beiden Modelle werden ein klareres Bild zu dieser Vermutung geben. Ab der 80. Epoche ist ein kürzerer Einbruch der Validierungsgenauigkeit erkennbar. Abgesehen davon ist auch dieses Modell äußerst stabil und weist nur ein minimales Overfitting auf.
Die Konfusionsmatrix zeigt erneut die gleichen Verwechselungstendenzen, wie das zweite Modell. Im Gegensatz zum ersten Modell, ist die Verwechselung der statischen Bewegungsaktivitäten, trotz einer sehr ähnlichen erreichten Testgenauigkeit wesentlich klarer

auf SITTING konzentriert. Dies lässt annehmen, dass die Komplexität eines Modells mit drei Hidden Layern dem Datensatz besser angeglichen ist, als eines mit bloß einem Hidden Layer. Da der Suchraum für eine Netzwerktiefe von drei Hidden Layern außerdem nicht vollständig untersucht wurde, ist es vorstellbar, dass bei weiterem Hyperparameter Tuning ein Modell entsteht, das die Genauigkeit des ersten Modells übersteigt. Gleichzeitig ist jedoch auch eine spezielle Schwäche des Modells erkennbar. Wohingegen die vorherigen Modelle ausgeglichener bei der Verwechselung auf beiden Achsen waren, hat dieses Modell eine stärkere Neigung zur Verwechselung von korrekten WALKING_DOWNSTAIRS und SITTING Instanzen mit anderen Bewegungsaktivitäten. Im Vergleich mit dem zweiten Modell werden allgemein die seltenen Fehlerfälle ein wenig abgeschwächt, die häufigen Fehlerfälle jedoch auch erkennbar verstärkt. Dadurch ist das dritte Modell mit 17 Fehlerfällen für die statischen Aktivitäten im Vergleich zu den 19 Fehlerfällen des zweiten Modells leicht besser. Das Modell ist mit 37 Fehlerfällen für die dynamischen Aktivitäten jedoch auch wesentlich schlechter als das zweite Modell mit 24 Fehlerfällen.

Das beste Modell für eine Netzwerktiefe von vier Hidden Layern auf der ersten Version des Datensatzes erreichte eine Testgenauigkeit von 98,05% in der 53. Epoche. Insgesamt trainierte das Modell 104 Epochen. Abbildung 4.4 zeigt den zugehörigen Trainingsverlauf in der linken Abbildung und die Konfusionsmatrix in der rechten Abbildung.

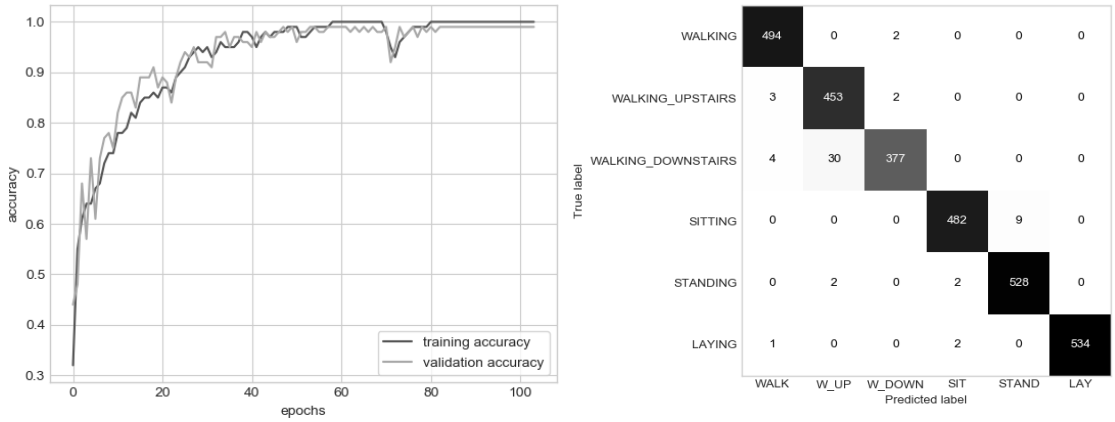

Abbildung 4.4: Das beste Modell mit vier Hidden Layern für Datensatzversion 1 mit einer Testgenauigkeit von 98,05%

Das vierte Modell stagniert bereits nach ca. 60 Epochen, wodurch es eine geringere Zeit benötigt, als das dritte Modell. Jedoch bleibt die Validierungsgenauigkeit geringer, was auf ein temporär stärkeres Overfitting zurückgeführt werden kann. Erst nach einem Einbruch von knapp 8% für beide Genauigkeiten erholt sich das Modell und hält eine stabile Leistung. Demnach beginnt die wirkliche Stagnation des Trainingsverlaufes ohne maßgebliche Einbrüche erst bei der 80. Epoche.
Das vierte Modell führt die Tendenz, die zuvor beim dritten Modell beobachtet wurde, weiter, dass größere Fehler verstärkt und kleinere Fehler abgeschwächt werden. Dies führt in diesem Fall dazu, dass ein enormer Fokus der Fehlerfälle auf die Verwechselung von WALKING_DOWNSTAIRS Instanzen mit der WALKING_UPSTAIRS Aktivität erkennbar ist. Bei den statischen Aktivitäten gleicht sich die Anzahl an Fehlerfällen im Vergleich zum dritten Modell mehr aus. Gleichzeitig besitzt das Modell aber auch die höchste Verwechselung zwischen dynamischen und statischen Aktivitäten aller Modelle für die erste Datensatzversion mit drei Verwechselungen.

Das beste Modell für eine Netzwerktiefe von fünf Hidden Layern auf der ersten Version des Datensatzes erreichte eine Testgenauigkeit von 97,40% in der 221. Epoche. Insgesamt trainierte das Modell 272 Epochen. Abbildung 4.5 zeigt den zugehörigen Trainingsverlauf in der linken Abbildung und die Konfusionsmatrix in der rechten Abbildung.

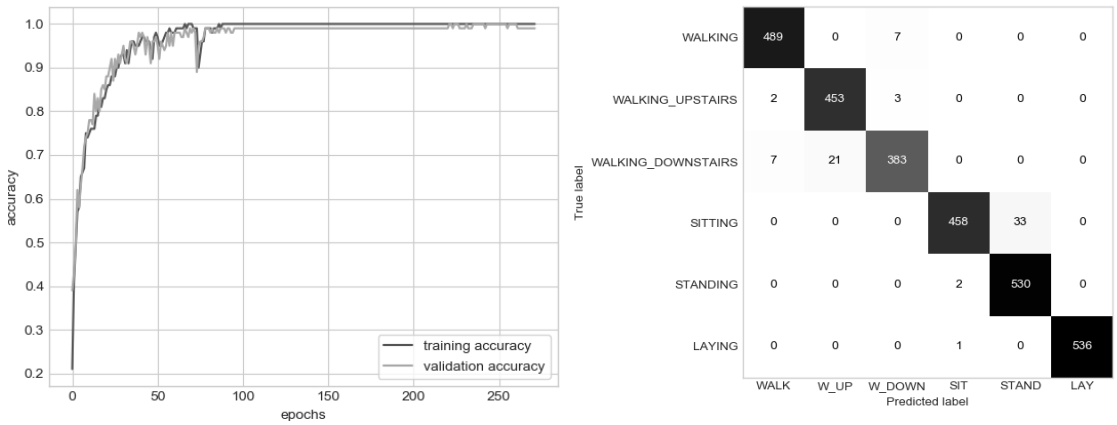

Abbildung 4.5: Das beste Modell mit fünf Hidden Layern für Datensatzversion 1 mit einer Testgenauigkeit von 97,40%

Das Modell mit fünf Hidden Layern zeigt mit Abstand das bisher geringste Overfitting über alle betrachteten Modelle. Bereits in den frühen Epochen bewegen sich beide Genauigkeiten auf einem äußerst gleichen Niveau. Gleichzeitig benötigt das Modell aber auch knapp 100 Epochen vor der Stagnation des Trainings. Von da aus bewegen sich beide Genauigkeiten mit einer ähnlichen Differenz wie die vorherigen Modelle. Ab der 220. Epoche kann beobachtet werden, dass sich die Validierungsgenauigkeit über ca. 40 Epochen mehrmals stark an die Trainingsgenauigkeit angleicht. Damit ist das fünfte Modell, trotz allgemein niedrigster Testgenauigkeit, das stabilste der Modelle. Das Modell trainierte mehr als doppelt so viele Epochen, wie das dritte oder vierte Modell und benötigte damit mit Abstand die längste Zeit für das Training.
Die Verwechselung der dynamischen Aktivitäten in der Konfusionsmatrix zeigt erneut die gleiche Tendenz wie das vierte Modell, wenn auch ein wenig gleichmäßiger. Das Modell ist gleichzeitig aber auch das einzige, das eine höhere Fehleranzahl unter den statischen Aktivitäten als unter den dynamischen besitzt. Dabei besitzt das Modell eine weitere starke Tendenz in der Verwechselung von SITTING Instanzen mit der STAND Aktivität, welche sich analog zur WALKING_DOWNSTAIRS Aktivität verhält. Damit besitzt das fünfte Modell die stärkste Tendenz zu einzelnen Fehlerfällen unter den Modellen.

4.2.2 Die besten Modelle für die erste Datensatzversion

Das beste Modell für eine Netzwerktiefe von einem Hidden Layer auf der zweiten Version des Datensatzes erreichte eine Testgenauigkeit von 98,12% in der 58. Epoche. Insgesamt trainierte das Modell 109 Epochen. Abbildung 4.6 zeigt den zugehörigen Trainingsverlauf in der linken Abbildung und die Konfusionsmatrix in der rechten Abbildung.

Das Training folgt einem ähnlichen Verlauf, wie beim ersten Modell der ersten Datensatzversion. Jedoch ist bei der Validierungsgenauigkeit eine größere Unsicherheit erkennbar, die erst für die letzten 15 endet. Damit hat das Modell, auch bei einer sehr ähnlichen Testgenauigkeit, ein stärkeres Problem mit Overfitting. Diese Beobachtung lässt sich wahrscheinlich auf die weitere Unterteilung der Examples auf doppelt so viele Labels zurückführen. Gleichzeitig ist dies das beste Modell für die zweite Datensatzversion.
Auch in der Konfusionsmatrix sind die gleichen Tendenzen, wie beim ersten Modell des ersten Datensatzes erkennbar. Die dynamischen und statischen Aktivitäten weisen erneut eine vergleichsweise hohe Unsicherheit bei der Erkennung der WALKING_DOWNSTAIRS

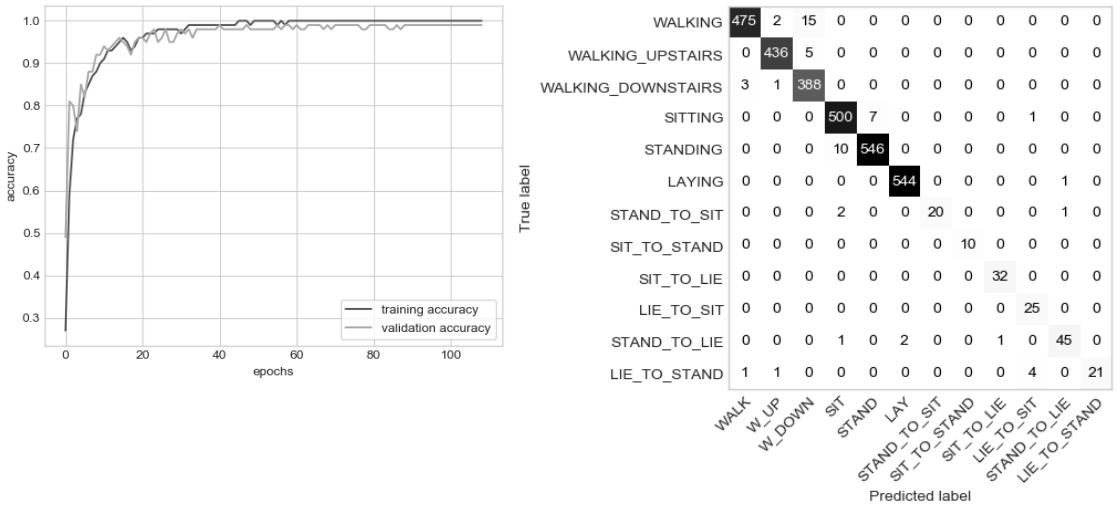

Abbildung 4.6: Das beste Modell mit einem Hidden Layer für Datensatzversion 2 mit einer Testgenauigkeit von 98,12%

und SITTING Aktivitäten auf. Die Übergangsaktivitäten, die die zweite Version des Datensatzes hinzufügte, sind jedoch wesentlich weniger lokal für ihre Fehlerfälle. Die Übergangsaktivitäten werden dabei auch mit den anderen beiden Gruppen verwechselt. Diese Fehlerfälle beschränken sich auch nicht auf die beiden Aktivitäten, die vor und nach der Übergangsaktivität stattfinden. Die Ursache ist höchstwahrscheinlich der ungleiche Anteil an Übergangsaktivitäten, zu den anderen beiden Kategorien.

Das beste Modell für eine Netzwerktiefe von zwei Hidden Layern auf der zweiten Version des Datensatzes erreichte eine Testgenauigkeit von 97,93% in der 134. Epoche. Insgesamt trainierte das Modell 185 Epochen. Abbildung 4.7 zeigt den zugehörigen Trainingsverlauf in der linken Abbildung und die Konfusionsmatrix in der rechten Abbildung.

Die verhältnismäßig höhere Unsicherheit des ersten Modells der zweiten Version des Datensatzes, kann auch bei diesem Modell beobachtet werden. Jedoch sind die Einbrüche der Genauigkeit im Verlauf des Trainings um einiges intensiver und auch für beide Genauigkeiten klar erkennbar. Die Einbrüche der Validierungsgenauigkeit sind trotzdem stärker als die der Trainingsgenauigkeit. Eine stabile Erkennungsrate gelingt dem Modell erst zur 140. Epoche, wohingegen das zweite Modell der ersten Version dies bereits nach nicht einmal 60 Epochen gelang. Auch zeigt dieses Modell ein wesentlich längeres Training auf, als das erste Modell der zweiten Version. Gleichzeitig ist die Trainingslänge

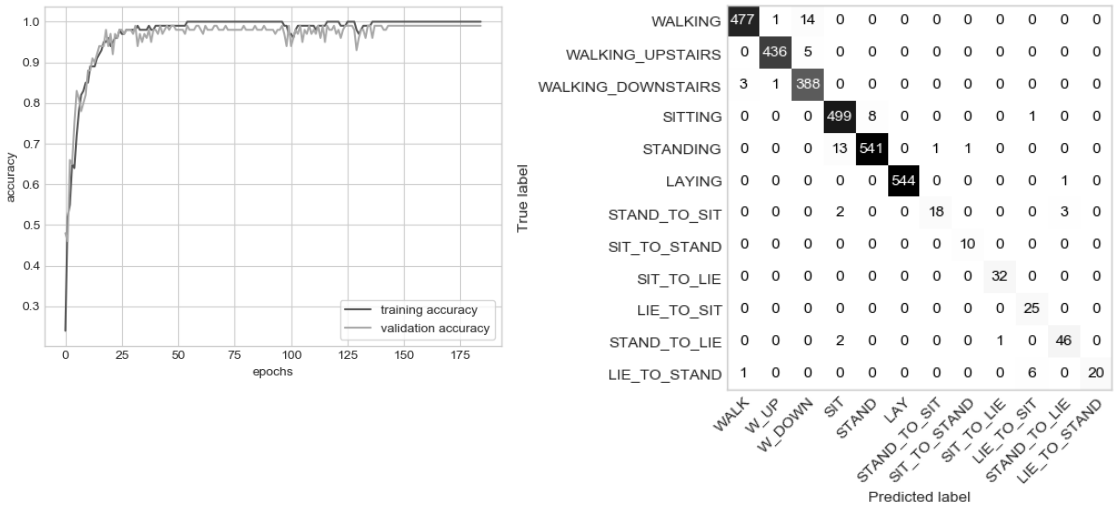

Abbildung 4.7: Das beste Modell mit zwei Hidden Layern für Datensatzversion 2 mit einer Testgenauigkeit von 97,93%

mit 185 Epochen auch deutlich länger, als die des zweiten Modells der ersten Version mit 95. Wie bereits in Tabelle 4.11 einsehbar war, ist die maximale Trainingsdauer der Modelle der zweiten Datensatzversion um einiges höher, als die der ersten Version. Auch hier ist die Ursache sicherlich die breitere Trennung der Labels und damit der geringere Lerneffekt für einzelne Aktivitätsklassen pro Epoche.
Die Konfusionsmatrix zeigt eine ähnliche Verteilung der Häufigkeit der Fehlerfälle im Vergleich zum ersten Modell der zweiten Version. Es ist aber eine leichte Verbesserung in der Erkennung von dynamischen Aktivitäten und eine leichte Verschlechterung bei den statischen zu sehen.

Das beste Modell für eine Netzwerktiefe von drei Hidden Layern auf der zweiten Version des Datensatzes erreichte eine Testgenauigkeit von 97,67% in der 109. Epoche. Insgesamt trainierte das Modell 160 Epochen. Abbildung 4.8 zeigt den zugehörigen Trainingsverlauf in der linken Abbildung und die Konfusionsmatrix in der rechten Abbildung.

Auch wenn das dritte Modell der zweiten Datensatzversion weniger extreme Einbrüche in der Trainingsgenauigkeit zeigt als das zweite Modell, ist die Differenz zur Validierungsgenauigkeit um einiges höher. Dieses vergleichsweise hohe Overfitting bleibt ebenso bis zum Ende des Trainings bestehen, mit bloß einer kürzeren stabilen Phase zwischen der 120. und 140. Epoche. Die Trainingsdauer ist ebenso erneut um einiges höher, als die des

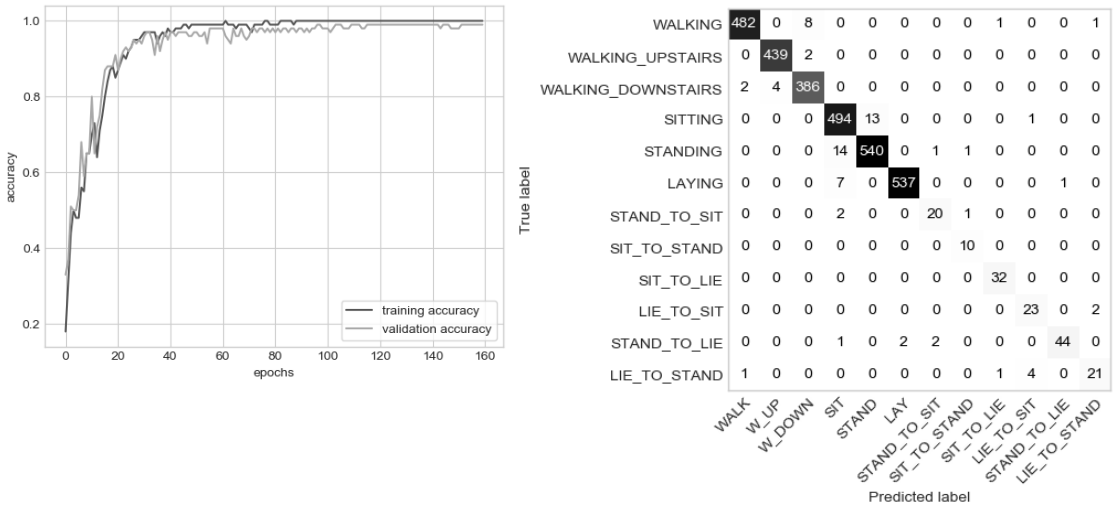

Abbildung 4.8: Das beste Modell mit drei Hidden Layern für Datensatzversion 2 mit einer Testgenauigkeit von 97,67%

dritten Modells der ersten Version.
Bei der Konfusionsmatrix ist ebenso erneut zu erkennen, dass sich die Erkennungsrate der dynamischen Aktivitäten weiter verbessert hat, bei den statischen aber auch weiter verschlechtert. Die Genauigkeit für die Übergangsaktivitäten bleibt weiterhin verhältnismäßig unverändert.

Das beste Modell für eine Netzwerktiefe von vier Hidden Layern auf der zweiten Version des Datensatzes erreichte eine Testgenauigkeit von 97,06% in der 139. Epoche. Insgesamt trainierte das Modell 190 Epochen. Abbildung 4.9 zeigt den zugehörigen Trainingsverlauf in der linken Abbildung und die Konfusionsmatrix in der rechten Abbildung.

Der Trainingsverlauf des vierten Modells führt die beobachtete Tendenz fort. Dabei ist bei diesem Modell so gut wie keine stabile Phase erkennbar und das Overfitting ist vor allem zwischen der 60. und 130. Epoche besonders hoch. Auch die Trainingsdauer bleibt um einiges höher, als beim Gegenstück der ersten Datensatzversion.
Die Konfusionsmatrix zeigt im Gegensatz zum dritten Modell der zweiten Version eine Verschlechterung der dynamischen Bewegungsaktivitäten und eine geringe Verbesserung bei den statischen. Auch bei den Übergangsaktivitäten ist erstmals eine leichte Verschlechterung erkennbar. Insgesamt verschlechtert sich das Modell mit diesen Veränderungen jedoch weiter.

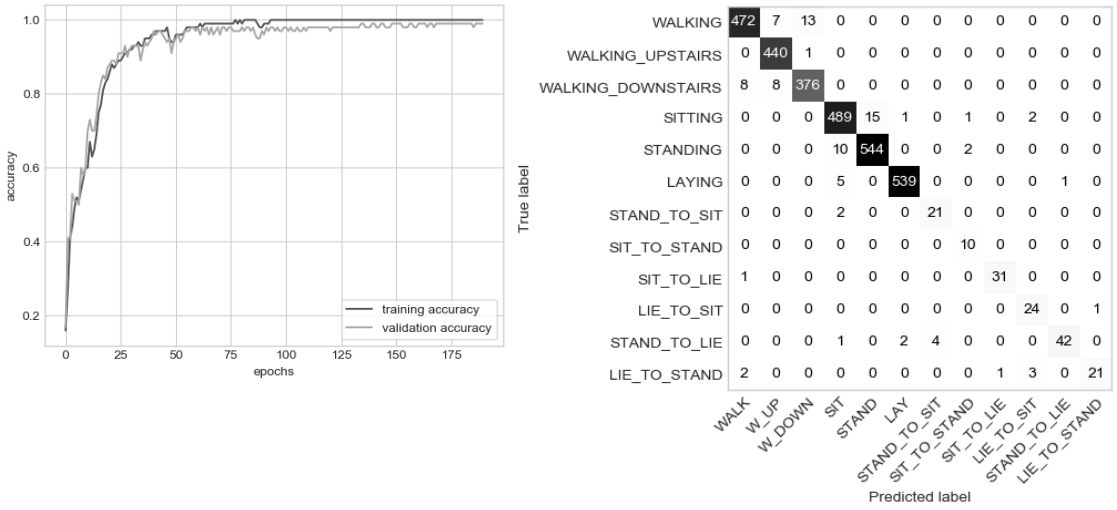

Abbildung 4.9: Das beste Modell mit vier Hidden Layern für Datensatzversion 2 mit einer Testgenauigkeit von 97,06%

Das beste Modell für eine Netzwerktiefe von fünf Hidden Layern auf der zweiten Version des Datensatzes erreichte eine Testgenauigkeit von 96,51% in der 168. Epoche. Insgesamt trainierte das Modell 219 Epochen. Abbildung 4.10 zeigt den zugehörigen Trainingsverlauf in der linken Abbildung und die Konfusionsmatrix in der rechten Abbildung.

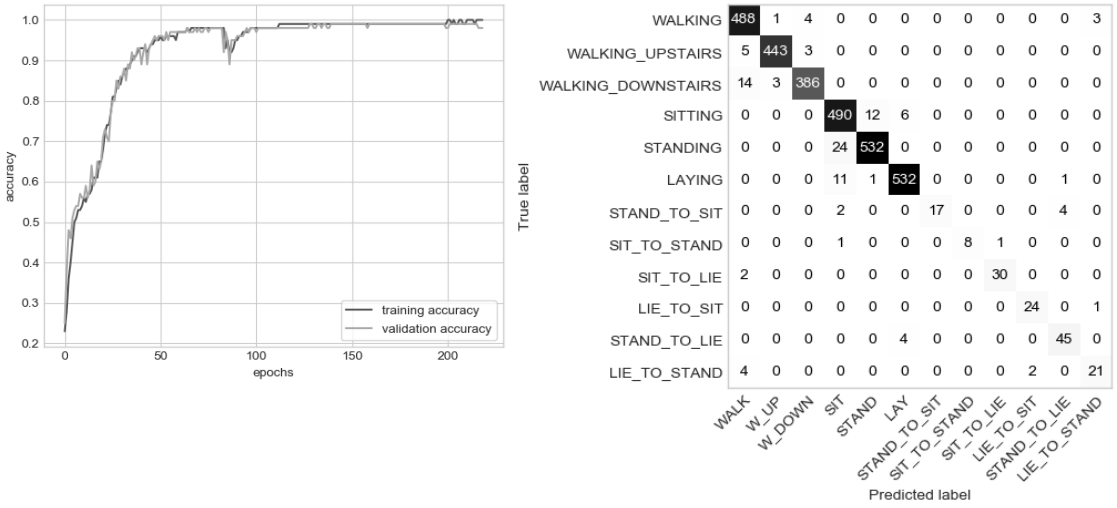

Abbildung 4.10: Das beste Modell mit fünf Hidden Layern für Datensatzversion 2 mit einer Testgenauigkeit von 96,51%

Der Trainingsverlauf zeigt, dass das Modell das Verhalten des fünften Modells der ersten Datensatzversion beibehält und gleichzeitig die allgemeinen Beobachtungen für die Modelle der zweiten Version annimmt. Dadurch ergibt sich ein äußerst geringes Overfitting im gesamten Training. Gleichzeitig aber auch eine hohe Schwankung beider Genauigkeiten. Wohingegen die Trainingsgenauigkeit bei allen bisherigen Modellen sehr nah an 100% gelang, bewegt sich dieses Modell bis zur 200. Epoche stets erkennbar darunter. Erst nach der 200. Epoche und auch nur durch ein verstärktes Overfitting, erreicht die Trainingsgenauigkeit die 100%.
Auch wenn das vierte Modell der zweiten Version entgegen der bisherigen Tendenzen ging, nimmt das fünfte Modell diese erneut an. Das Modell ist leicht besser im Erkennen der dynamischen Aktivitäten, als bei den statischen. Auch wenn die Leistung allgemein geringer ist, als bei den vorherigen Modellen.

4.2.3 Zusammenfassung der besten Modelle

Das beste Modell für die erste Version des Datensatzes besitzt eine Tiefe von zwei Hidden Layern mit einer erreichten Testgenauigkeit von 98,46%. Alle Netzwerktiefen drumherum fallen langsam in ihrer Leistung ab. Ein tieferes Netzwerk benötigt gleichzeitig einen erhöhten Trainingsaufwand, was in der Anzahl an Epochen beobachtet werden konnte, die nötig sind, bis das Training stagniert. Je tiefer das Netzwerk, desto geringer fällt jedoch auch das Overfitting der Trainingsdaten aus. Für die zweite Datensatzversion ist das Modell mit einem Hidden Layer mit einer Testgenauigkeit von 98,12% das beste Modell. Es ist ein ähnlicher Leistungsabfall über die Vergrößerung der Netzwerktiefe erkennbar. Die Modelle der zweiten Datensatzversion weisen ein durchschnittlich höheres Overfitting auf, als die der ersten Version. Auch benötigen sie eine höhere Anzahl an Epochen für ihr Training.

Alle Modelle sind sehr gut in der Lage die dynamischen und statischen Bewegungsaktivitäten zu differenzieren. Die Modelle der zweiten Version zeigen im Vergleich eine höhere Unsicherheit bei der Abgrenzung der Übergangsaktivitäten von den anderen beiden Kategorien. Die Modelle waren meist ein wenig besser im Erkennen der dynamischen Bewegungsaktivitäten, als bei den statischen. Gleichzeitig besitzen alle Modelle die gleiche Tendenz, sich besonders unsicher bezüglich der WALKING_DOWNSTAIRS und SITTING Aktivitäten zu sein. Dies impliziert je nach Modell in unterschiedlicher Ausprägung sowohl das falsche Identifizieren der Aktivität, als auch das Verwechseln von anderen Aktivitäten der gleichen Kategorie mit ihr.

Der Vergleich der Modelle zwischen den beiden Versionen des Datensatzes zeigt, dass die Unterteilung in 12 statt wie zuvor 6 Labels einen eindeutig negativen Effekt auf das Training der Modelle hat. Der Trainingsverlauf ist instabiler und benötigt weitaus mehr Epochen, bevor das Training sich stabilisiert.

4.3 Auswertung

Nachdem die Ergebnisse der Experimente dieser Arbeit zuvor vorgestellt und charakterisiert wurden, werden sie in diesem Abschnitt abschließend ausgewertet. Das Ergebnis der Experimente ist das beste Modell, das aus allen Experimenten entwickelt wurde.
Das beste Modell, das im Zuge dieser Arbeit trainiert wurde, besitzt eine Tiefe von zwei Hidden Layern und wurde auf der ursprünglichen, ersten Version des Datensatzes ohne Übergangsaktivitäten trainiert. Dabei erreichte das Modell eine Testgenauigkeit von 98,46%. In Tabelle 4.12 wurde der Aufbau des Netzwerks bereits genauer beschrieben.

Das Modell ist um über 4% besser im Erkennen von Bewegungsaktivitäten, als das LSTM Netzwerk von Yu et. al. [59], das statt den statischen Features des Datensatzes die Rohdaten direkt als Eingabe verwendete. Die besten Ergebnisse unter allen neuronalen Netzen erreichten Ignatov und Andrey [27] mit einem CNN mit einer Genauigkeit von 95,31%, welches ebenso die Rohdaten als Features nutzte. Somit erreicht das LSTM Modell dieser Arbeit ebenso eine über 3% bessere Genauigkeit auf dem Datensatz, als das bisher beste neuronale Netz. Selbst im Vergleich zu State-of-the-Art Methoden des Machine Learnings ohne neuronale Netze ist ein klarer Vorsprung erkennbar. Hierbei stellte bisher der Ensamble Bagged Tree von Chelli und Pätzold [9] mit 97,70% das beste Modell über alle Arbeiten dar. Hierbei zeigt das beste LSTM Modell dieser Arbeit eine Verbesserung von 0,76%. Der Ensamble Bagged Tree wurde mit Hilfe von speziell engineerten Features trainiert. Diese wurden in den Experimenten von Chelli und Pätzold mit den ursprünglichen Features verglichen und zeigten einen eindeutigen Leistungsanstieg für die entwickelten Modelle.

4.4 Fazit

Die Experimente dieser Arbeit zeigen, dass LSTMs eine konkurrenzfähige Methode des Machine Learnings zur Erkennung von Bewegungsaktivitäten für den eingesetzten Datensatz sind. Im Speziellen die zusätzliche Betrachtung der zeitlichen Dimension durch Zeitreihendaten, zeigte sich als positiver Einfluss auf die Leistung des Modells.

Gleichzeitig besitzt das entwickelte Modell klare Einschränkungen in seiner Benutzbarkeit und eindeutige Fehlertendenzen. Eine solche Einschränkung ist es, dass das Modell auf aufwendig vorverarbeiteten Features trainiert wurde. Hierdurch ist eine Übertragung in einen Echtzeitkontext nur insofern bedingt möglich, als dass eine Vorverarbeitung der rohen Signale in Echtzeit notwendig wäre. Hierbei müssten cloud-basierte Ansätze verfolgt werden, da eine direkte Berechnung auf einem Smartphone sehr ressourcenintensiv wäre.
Alternativ könnte die überlegene Erkennungsrate des LSTMs mit anderen Methoden des Machine Learnings kombiniert werden, um deren Stärken mit aufzugreifen. So wäre ein Vorverarbeitungsschritt in der Modellarchitektur möglich, bei dem ein CNN Layer eingesetzt wird, um dynamische Features zu erzeugen, die anschließend an das LSTM weitergereicht werden, um die Aktivitätsklassen zu erkennen.

Außerdem kamen im Verlauf dieser Arbeit mehrere Aspekte zum Vorschein, die bei einer Weiterführung der Experimente betrachtet werden können, um die Fehlertendenzen zu minimieren und damit die Leistung der entwickelten Modelle weiter zu verbessern. Die erste Methode zur weiteren Verbesserung wäre es dabei, den Suchraum an möglichen Hyperparameterkonfigurationen weiter auszuschöpfen. Bis auf die Modelle mit einem einzelnen Hidden Layer, wurde kein Suchraum bisher vollständig untersucht. Selbst von den nächstgrößeren Modellen mit zwei Hidden Layern wurden im Falle der ersten Datensatzversion nur 1185 von 7500 möglichen Hyperparameterkonfigurationen untersucht. Der Suchraum für diese Fortführung könnte außerdem durch die Verringerung von möglichen Konfigurationen pro Hyperparameter weiter optimiert werden, da die bisherigen Experimente bereits weitreichende Erkenntnisse über die Relevanz und Güte der Konfigurationen bieten.
Für eine weitere Optimierung des Hyperparameter Tunings kann ein anderes Entscheidungsverfahren als Random Search gewählt werden. So kann zum Beispiel ein Verfahren wie Bayesian Optimization genutzt werden, bei dem die Genauigkeit von bereits trainierten Modellen mit ihren Hyperparameterkonfigurationen als Kriterium genutzt wird,

um neue Konfigurationen auszuwählen. Es existieren sogar weitere Methoden, die das Hyperparameter Tuning über eigene neuronale Netze umsetzen.

Für das Training von einem neuronalen Netz ist der Datensatz zudem äußerst klein. Dies wurde von Anguita et. al. ursprünglich damit begründet, dass der Datensatz mit Support Vector Machines als Lösungsmethode aufgezeichnet wurde und diese besser auf kleinen Datensätzen arbeiten, als auf größeren [2]. Für eine Verbesserung von Modellen, die sich auf neuronale Netze im Allgemeinen verlassen, wäre jedoch eine Vergrößerung des Datenbestands die wichtigste Maßnahme. Die Evaluation der besten Modelle nach ihrer Netzwerktiefe zeigte außerdem auf, dass die Aktivitätsklassen, bei denen sich die Modelle am unsichersten bei der Erkennung waren, gleichzeitig auch die am seltensten im Datensatz vorkommenden Aktivitäten sind. Demnach kann eine ausgewogenere Verteilung der Examples pro Label einen weiteren positiven Effekt auf die Erkennungsrate eines LSTMs besitzen.

Eine Verbesserung des Featurevektors könnte ebenso einen positiven Einfluss besitzen, wie Chelli und Pätzold [9] in ihren Experimenten bereits für andere Methoden zeigten. Hierbei ist anzunehmen, dass auch LSTMs mit den neuen Features von Chelli und Pätzold verbessert werden würden. Gleichzeitig setzt dies jedoch auch weiteres, verstärktes Expertenwissen über die Domäne voraus, das den Stand des ursprünglichen Datensatzes übersteigt. Die Betrachtung von weiteren Datensätzen aus dem Bereich der Human Activity Recognition oder gar angrenzenden Klassifikationsproblemen mit ähnlichen realweltlichen Anwendungsfällen könnte außerdem das allgemeine Potential von LSTMs für diese Art von Problemstellung untersuchen.

5 Schluss

Im letzten Kapitel werden die gewonnenen Erkenntnisse der Arbeit zusammengefasst und ein Ausblick gegeben, welche Forschungsansätze künftig auf dieser Arbeit aufbauen könnten.

5.1 Zusammenfassung der Arbeit

Diese Arbeit hatte zum Ziel, einen Klassifikator mit Hilfe eines LSTMs zu entwickeln und zu optimieren. Dieser sollte in der Lage sein, die Aktivitätsklassen des vorgestellten Datensatzes in seinen beiden Versionen als Zeitreihendaten möglichst genau zu erkennen. In Kapitel 2 wurden dazu zuerst zugrundeliegende Begriffe und Vorgehensweisen erläutert. Aus der anschließenden Vorstellung des Datensatzes ergab sich, dass der Datensatz keine weiteren Features neben den aufgezeichneten Sensoren besitzt, die die Daten kontextualisieren würden. Danach wurden die bisherigen Ergebnisse anderer Arbeiten auf dem Datensatz beschrieben. Hierbei zeigten sich unterschiedliche Herangehensweisen an die Daten, bei denen entweder vorverarbeitete, statische Features [2] [9] oder dynamische Features, die das Modell selbst aus den Rohdaten ermittelte [27] [59], eingesetzt.

Kapitel 3 beschrieb auf Grundlage der Analyse den geplanten Aufbau der experimentellen Umgebung. Dabei wurde der KDD Prozess von Fayyad et. al. [16] als Grundlage gewählt, um die Schritte zur Entwicklung eines Klassifikators zu beschreiben. Aus den einzelnen Schritten Selection, Preprocessing, Transformation, Data Mining/Machine Learning und Interpretation/Evaluation ergaben sich die Anforderungen an die experimentelle Umgebung. Letztlich wurden die Technologien beschrieben, die gewählt wurden, um die Umgebung zu implementieren.

In Kapitel 4 wurde zu Beginn die Umsetzung der experimentellen Umgebung beschrieben und welche Änderungen vom ursprünglichen Vorhaben vorgenommen werden mussten. Es wurde eine übergeordnete Schleife definiert, die sequentiell zufällige Hyperparameterkonfigurationen auswählt und aus diesen das Netzwerk definiert. Zusammen mit den transformierten Daten wurde anschließend ein Modell trainiert und nach Abschluss des Trainings getestet. Dieser Vorgang wurde anschließend für die nächste Konfiguration wiederholt. Zur korrekten Weiterführung der Experimente wurden alle relevanten Informationen über abgeschlossene Trainings und den allgemeinen Fortschritt persistiert. Es wurde außerdem die Herleitung des finalen Suchraums an Hyperparameterkonfigurationen beschrieben, bei dem die Leistungen erster Modelle zur Unterstützung der Auswahl genutzt wurden.
Danach wurden die Ergebnisse der abgeschlossenen Experimente zusammengefasst. Insgesamt wurden dabei 10.270 Modelle trainiert und getestet. Aus diesen wurden die besten Modelle pro Experiment ausgewählt, im Einzelnen näher untersucht und verglichen. Die Modelle zeigten alle die Fähigkeit, die unterschiedlichen Kategorien an Bewegungsaktivitäten (dynamisch, statisch) sehr gut zu differenzieren. Außerdem zeigte sich eine allgemeine Fehlertendenz pro Kategorie. Bei den dynamischen Aktivitäten war WALKING_DOWNSTAIRS stets am unsichersten, sowohl bei der korrekten Vorhersage, als auch bei der Häufigkeit der Verwechslung mit anderen dynamischen Aktivitäten. Analog zeigte sich unter den statischen Aktivitäten eine verstärkte Unsicherheit bei der SITTING Aktivität. Allgemein waren die Modelle ein wenig besser im Erkennen der dynamischen Aktivitäten, als bei den statischen.
Das beste Modell, das im Zuge der Experimente entwickelt werden konnte, besitzt eine Testgenauigkeit von 98,46%. Damit übertrifft das Modell alle Ergebnisse, die andere Arbeiten bisher auf dem Datensatz hervorbrachten. Es konnte somit gezeigt werden, dass LSTMs eine vielversprechende Methode für die gewählte Problemstellung sind. Zuletzt wurden in einem Fazit mögliche Weiterführungen der Experimente diskutiert.

5.2 Ausblick

Im Verlauf dieser Arbeit konnte gezeigt werden, dass für den gewählten Datensatz LSTMs eine aussichtsvolle Methode sind, um menschliche Bewegungen aus Zeitreihendaten zu erkennen. Das Modell besitzt ein sehr gutes Verständnis darüber, wie sich verschiedene Aktivitätsgruppen in ihren Mustern unterscheiden. Dieses bemerkenswerte konzeptionelle Verständnis der Bewegungsaktivitäten lässt vermuten, dass LSTMs auch bei anderen Problemstellungen aus der Human Activity Recognition oder dem Quantified Self ähnlich gute Ergebnisse erzielen würden.

Im Allgemeinen finden Deep Learning Methoden in letzter Zeit bereits diverse neue Anwendungsfälle. So existieren im Bereich des Quantified Selfs zum Beispiel Gesundheits-Apps, die Vitalwerte untersuchen und bereits eigene Diagnosen stellen [4]. Krankenkassen stellen bereits Prämien aus, wenn Versicherte einer Datenübermittelung von einem getragenen Fitnessarmband zustimmen [12]. Gleichzeitig kommen Bedenken bezüglich der Privatsphäre beim Quantified Self auf [42]. Auch wenn Fragen bezüglich der Verwendung von personenbezogenen Daten auch in Zukunft ein wichtiger Bestandteil der allgemeinen Entwicklung sein werden, nimmt diese Arbeit keine weitere Stellung dazu.

Der Einsatz von Machine Learning und Deep Learning Verfahren ist bereits seit Jahren nicht mehr auf Forschungszwecke beschränkt. Kleinere wie größere Organisationen jeglicher Art können wertvolle Informationen aus ihren Daten gewinnen [17]. Viele Organisation besitzen diese Daten bereits, es fehlt jedoch das Wissen um den Umgang mit ihnen. Außerdem ist die Qualitätsabsicherung von Datenaufzeichnungen in vielen Fällen noch sehr gering. Nur wenn Daten mit einer hohen Qualität vorliegen, kann aus ihnen ihr wirklicher Wert gewonnen werden [17].
Die Unterstützung von Entscheidungsprozessen durch Machine Learning Modelle ist bereits heute ein entscheidender Faktor und wird auch in Zukunft eine zunehmend tragende Rolle spielen.

Literaturverzeichnis

[1] Abdi, Hervé ; Williams, Lynne J.: Principal component analysis. In: *Wiley interdisciplinary reviews: computational statistics* 2 (2010), Nr. 4, S. 433–459

[2] Anguita, Davide ; Ghio, Alessandro ; Oneto, Luca ; Parra, Xavier ; Reyes-Ortiz, Jorge L.: Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine. In: *International workshop on ambient assisted living* Springer (Veranst.), 2012, S. 216–223

[3] Anguita, Davide ; Ghio, Alessandro ; Oneto, Luca ; Parra, Xavier ; Reyes-Ortiz, Jorge L.: A public domain dataset for human activity recognition using smartphones. In: *ESANN*, 2013

[4] Beneker, Christian: Die Selbstvermessung des eigenen Ichs. In: *Ärzte Zeitung* (2016), September. – URL `https://www.aerztezeitung.de/praxis_wirtschaft/medizintechnik/article/919796/gesundheits-apps-selbstvermessung-des-eigenen-ichs.html`. – (zuletzt aufgerufen am 21.06.2019)

[5] Bergstra, James ; Bengio, Yoshua: Random search for hyper-parameter optimization. In: *Journal of Machine Learning Research* 13 (2012), Nr. Feb, S. 281–305

[6] Brown, Terry: 5 Big Data Predictions for 2019. In: *ITChronicles* (2018), Dezember. – URL `https://www.itchronicles.com/big-data/5-big-data-predictions-for-2019/`. – (zuletzt aufgerufen am 21.06.2019)

[7] Brownlee, Jason: *Difference Between Classification and Regression in Machine Learning*. Mai 2019. – URL `https://machinelearningmastery.com/classification-versus-regression-in-machine-learning/`. – (zuletzt aufgerufen am 21.06.2019)

[8] Brownlee, Jason: *How to One Hot Encode Sequence Data in Python.* Juni 2019. – URL https://machinelearningmastery.com/how-to-one-hot-encode-sequence-data-in-python/. – (zuletzt aufgerufen am 21.06.2019)

[9] Chelli, Ali ; Pätzold, Matthias: A Machine Learning Approach for Fall Detection and Daily Living Activity Recognition. In: *IEEE Access* (2019)

[10] Chung, Junyoung ; Gulcehre, Caglar ; Cho, KyungHyun ; Bengio, Yoshua: Empirical evaluation of gated recurrent neural networks on sequence modeling. In: *arXiv preprint arXiv:1412.3555* (2014)

[11] Claesen, Marc ; Moor, Bart D.: *Hyperparameter Search in Machine Learning.* 2015

[12] Deutscher Ärzteverlag GmbH, Redaktion Deutsches Ärzteblatt: Fitness-Tracker: Der Datenhunger wächst. In: *Deutsches Ärzteblatt* (2016), Februar. – URL https://www.aerzteblatt.de/archiv/174975/Fitness-Tracker-Der-Datenhunger-waechst. – (zuletzt aufgerufen am 21.06.2019)

[13] Dietterich, Thomas G.: An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting and randomization. In: *Machine learning* 32 (1998), S. 1–22

[14] Dietterich, Tom: Overfitting and Undercomputing in Machine Learning. In: *ACM Comput. Surv.* 27 (1995), September, Nr. 3, S. 326–327. – URL http://doi.acm.org/10.1145/212094.212114. – ISSN 0360-0300

[15] Docker Inc.: *Docker Documentation.* – URL https://docs.docker.com/. – (zuletzt aufgerufen am 21.06.2019)

[16] Fayyad, Usama ; Piatetsky-Shapiro, Gregory ; Smyth, Padhraic: From data mining to knowledge discovery in databases. In: *AI magazine* 17 (1996), Nr. 3, S. 37

[17] Goddard, William: Data – The New Gold Rush for Businesses. In: *ITChronicles* (2019), April. – URL https://www.itchronicles.com/technology/data-the-new-gold-rush-for-businesses/. – (zuletzt aufgerufen am 21.06.2019)

[18] Google Developers: *API Documentation TensorFlow Core 1.13.* – URL https://www.tensorflow.org/api_docs. – (zuletzt aufgerufen am 21.06.2019)

[19] Graves, Alex: *Generating Sequences With Recurrent Neural Networks.* 2013

[20] Greff, Klaus ; Srivastava, Rupesh K. ; Koutnik, Jan ; Steunebrink, Bas R. ; Schmidhuber, Jurgen: LSTM: A Search Space Odyssey. In: *IEEE Transactions on Neural Networks and Learning Systems* 28 (2017), Oktober, Nr. 10, S. 2222–2232. – URL http://dx.doi.org/10.1109/TNNLS.2016.2582924. – ISSN 2162-2388

[21] Hagan, Martin T. ; Demuth, Howard B. ; Beale, Mark H. ; De Jesús, Orlando: *Neural network design*. Bd. 20. Pws Pub. Boston, 1996

[22] Hermans, Michiel ; Schrauwen, Benjamin: Training and Analysing Deep Recurrent Neural Networks. In: Burges, C. J. C. (Hrsg.) ; Bottou, L. (Hrsg.) ; Welling, M. (Hrsg.) ; Ghahramani, Z. (Hrsg.) ; Weinberger, K. Q. (Hrsg.): *Advances in Neural Information Processing Systems 26*. Curran Associates, Inc., 2013, S. 190–198. – URL http://papers.nips.cc/paper/5166-training-and-analysing-deep-recurrent-neural-networks.pdf

[23] Hinton, Geoffrey E. ; Sejnowski, Terrence J. ; Poggio, Tomaso A.: *Unsupervised learning: foundations of neural computation*. MIT press, 1999

[24] Hochreiter, Sepp ; Schmidhuber, Jürgen: Long short-term memory. In: *Neural computation* 9 (1997), Nr. 8, S. 1735–1780

[25] Hochreiter, Sepp ; Schmidhuber, Jürgen: LSTM can solve hard long time lag problems. In: *Advances in neural information processing systems*, 1997, S. 473–479

[26] Hsu, Chih-Wei ; Lin, Chih-Jen: A comparison of methods for multiclass support vector machines. In: *IEEE transactions on Neural Networks* 13 (2002), Nr. 2, S. 415–425

[27] Ignatov, Andrey: Real-time human activity recognition from accelerometer data using Convolutional Neural Networks. In: *Applied Soft Computing* 62 (2018), S. 915–922

[28] Ismail Fawaz, Hassan ; Forestier, Germain ; Weber, Jonathan ; Idoumghar, Lhassane ; Muller, Pierre-Alain: Deep learning for time series classification: a review. In: *Data Mining and Knowledge Discovery* 33 (2019), März, Nr. 4, S. 917–963. – URL http://dx.doi.org/10.1007/s10618-019-00619-1. – ISSN 1573-756X

[29] Joy, Tinu T. ; Rana, Santu ; Gupta, Sunil ; Venkatesh, Svetha: *Fast Hyperparameter Tuning using Bayesian Optimization with Directional Derivatives*. 2019

[30] Kaelbling, Leslie P. ; Littman, Michael L. ; Moore, Andrew W.: Reinforcement learning: A survey. In: *Journal of artificial intelligence research* 4 (1996), S. 237–285

[31] Karantonis, Dean M. ; Narayanan, Michael R. ; Mathie, Merryn ; Lovell, Nigel H. ; Celler, Branko G.: Implementation of a real-time human movement classifier using a triaxial accelerometer for ambulatory monitoring. In: *IEEE transactions on information technology in biomedicine* 10 (2006), Nr. 1, S. 156–167

[32] Keras Team: *Keras Documentation.* – URL `https://keras.io/`. – (zuletzt aufgerufen am 21.06.2019)

[33] Kingma, Diederik ; Ba, Jimmy: Adam: A Method for Stochastic Optimization. In: *International Conference on Learning Representations* (2014), 12

[34] Kolari, Pranam ; Java, Akshay ; Finin, Tim ; Oates, Tim ; Joshi, Anupam u. a.: Detecting spam blogs: A machine learning approach. In: *Proceedings of the national conference on artificial intelligence* Bd. 21 Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999 (Veranst.), 2006, S. 1351

[35] Kotsiantis, Sotiris B. ; Zaharakis, I ; Pintelas, P: Supervised machine learning: A review of classification techniques. In: *Emerging artificial intelligence applications in computer engineering* 160 (2007), S. 3–24

[36] LeCun, Yann ; Bengio, Yoshua u. a.: Convolutional networks for images, speech, and time series. In: *The handbook of brain theory and neural networks* 3361 (1995), Nr. 10, S. 1995

[37] LeCun, Yann ; Kavukcuoglu, Koray ; Farabet, Clément: Convolutional networks and applications in vision. In: *Proceedings of 2010 IEEE International Symposium on Circuits and Systems* IEEE (Veranst.), 2010, S. 253–256

[38] Li, Yongmou ; Shi, Dianxi ; Ding, Bo ; Liu, Dongbo: Unsupervised feature learning for human activity recognition using smartphone sensors. In: *Mining Intelligence and Knowledge Exploration.* Springer, 2014, S. 99–107

[39] Lukowicz, Paul ; Ward, Jamie A. ; Junker, Holger ; Stäger, Mathias ; Tröster, Gerhard ; Atrash, Amin ; Starner, Thad: Recognizing workshop activity using body worn microphones and accelerometers. In: *International conference on pervasive computing* Springer (Veranst.), 2004, S. 18–32

[40] Mannor, Shie ; Peleg, Dori ; Rubinstein, Reuven: The Cross Entropy Method for Classification. In: *Proceedings of the 22Nd International Conference on Machine Learning*. New York, NY, USA : ACM, 2005 (ICML '05), S. 561–568. – URL http://doi.acm.org/10.1145/1102351.1102422. – ISBN 1-59593-180-5

[41] Mockus, Jonas: *Bayesian approach to global optimization: theory and applications*. Bd. 37. Springer Science & Business Media, 2012

[42] Moody, Glyn: After the quantified self, the quantified employee; but what about privacy? In: *Private Internet Access Blog* (2019), März. – URL https://www.privateinternetaccess.com/blog/2019/03/after-the-quantified-self-the-quantified-employee-but-what-about-privacy/. – (zuletzt aufgerufen am 21.06.2019)

[43] Peck, Robert: Mark Cuban: „Data is the new gold". In: *Credit Suisse* (2017), Juni. – URL https://www.credit-suisse.com/corporate/en/articles/news-and-expertise/mark-cuban-data-is-the-new-gold-201706.html. – (zuletzt aufgerufen am 21.06.2019)

[44] Poppe, Ronald: Vision-based human motion analysis: An overview. In: *Computer vision and image understanding* 108 (2007), Nr. 1-2, S. 4–18

[45] Powers, David M.: Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation. (2011)

[46] Python Software Foundation: *Python 3.7.4rc1 Documentation*. – URL https://docs.python.org/3/. – (zuletzt aufgerufen am 21.06.2019)

[47] Raskutti, G. ; Wainwright, M. J. ; Yu, B.: Early stopping for non-parametric regression: An optimal data-dependent stopping rule. In: *2011 49th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, September 2011, S. 1318–1325

[48] Reyes-Ortiz, Jorge-L ; Oneto, Luca ; Samà, Albert ; Parra, Xavier ; Anguita, Davide: Transition-aware human activity recognition using smartphones. In: *Neurocomputing* 171 (2016), S. 754–767

[49] Ronao, Charissa A. ; Cho, Sung-Bae: Human activity recognition using smartphone sensors with two-stage continuous hidden Markov models. In: *2014 10th International Conference on Natural Computation (ICNC)* IEEE (Veranst.), 2014, S. 681–686

[50] Ronao, Charissa A. ; Cho, Sung-Bae: Human activity recognition with smartphone sensors using deep learning neural networks. In: *Expert Systems with Applications* 59 (2016), S. 235–244

[51] Ronao, Charissa A. ; Cho, Sung-Bae: Recognizing human activities from smartphone sensors using hierarchical continuous hidden Markov models. In: *International Journal of Distributed Sensor Networks* 13 (2017), Nr. 1, S. 1550147716683687

[52] Ronaoo, Charissa A. ; Cho, Sung-Bae: Evaluation of deep convolutional neural network architectures for human activity recognition with smartphone sensors. In: *Proceedings of the Korean Information Science Society* (2015), S. 858–860

[53] Schmidhuber, Jürgen: Deep learning in neural networks: An overview. In: *Neural networks* 61 (2015), S. 85–117

[54] Seto, Skyler ; Zhang, Wenyu ; Zhou, Yichen: Multivariate time series classification using dynamic time warping template selection for human activity recognition. In: *2015 IEEE Symposium Series on Computational Intelligence* IEEE (Veranst.), 2015, S. 1399–1406

[55] Srivastava, Nitish ; Hinton, Geoffrey ; Krizhevsky, Alex ; Sutskever, Ilya ; Salakhutdinov, Ruslan: Dropout: a simple way to prevent neural networks from overfitting. In: *The Journal of Machine Learning Research* 15 (2014), Nr. 1, S. 1929–1958

[56] Swan, Melanie: The quantified self: Fundamental disruption in big data science and biological discovery. In: *Big data* 1 (2013), Nr. 2, S. 85–99

[57] Van Rossum, Guido ; Drake Jr, Fred L.: *Python reference manual.* Centrum voor Wiskunde en Informatica Amsterdam, 1995

[58] Vincent, Pascal ; Larochelle, Hugo ; Bengio, Yoshua ; Manzagol, Pierre-Antoine: Extracting and composing robust features with denoising autoencoders. In: *Proceedings of the 25th international conference on Machine learning* ACM (Veranst.), 2008, S. 1096–1103

[59] Yu, Tao ; Chen, Jianxin ; Yan, Na ; Liu, Xipeng: A Multi-Layer Parallel LSTM Network for Human Activity Recognition with Smartphone Sensors. In: *2018 10th International Conference on Wireless Communications and Signal Processing (WCSP)* IEEE (Veranst.), 2018, S. 1–6

[60] Zulkifli, Hafidz: *Understanding Learning Rates and How It Improves Performance in Deep Learning*. Januar 2018. – URL `https://towardsdatascience.com/understanding-learning-rates-and-how-it-improves-performance-in-deep-learning-d0d4059c1c10`. – (zuletzt aufgerufen am 21.06.2019)