xAILAB
EXPLAINABLE MACHINE LEARNING

Human Activity Recognition via Deep Learning based on active exoskeleton data

Master Thesis

Master of Science Survey Statistik

Christoph Zink

April 14, 2023

Supervisor:

1st: Prof. Dr. Christian Ledig
2nd: M. Sc. Joachim Fischer, German Bionic

Chair of Explainable Machine Learning
Faculty of Information Systems and Applied Computer Sciences
Otto-Friedrich-University Bamberg

Abstract

Repetitive strenuous physical activity may lead to various chronic musculoskeletal disorders, reducing the quality of life of the affected. To prevent these disorders the usage of exoskeletons has gained popularity in recent years. These wearable devices exert forces upon the body of the user and relieve the worker by providing a portion of the needed forces. For optimal relief, it is necessary to precisely control the force-generating elements, i.e. the motors within active exoskeletons. One approach is using built-in sensors to classify the user's current activity and adjust the support accordingly. As the application of human activity recognition on active exoskeleton data is an emerging field with little research to date, this study aims at gathering experience with the usage of deep learning models for this purpose.

The data for this study was recorded on the active Exoskeleton Cray-X produced by the company German Bionic, intended to support the lower back when performing repetitive lifting tasks.

Four neural network architectures were trained within this study: A hierarchical feature-based feedforward network, a feedforward network receiving the raw data in a windowed format, a convolutional neural network and a long short-term memory network. These networks were compared with the baseline method, a hierarchically classifying support vector machine. This comparison was done on a self-recorded test set with $n = 10$ subjects to evaluate the ability of these models to generalize. The subjects and five environments within this test set were not observed within the training data. Furthermore, the performance of these deep learning models was compared to a model representing the conventional approach to human activity recognition, to test if neural networks are appropriate for the given task.

The results indicate that neural networks outperform the baseline method consistently, with the best model achieving an accuracy of 76.79% and a Matthews correlation coefficient of 73.48% on the $k = 7$ activities of daily living. The trained models appear to be robust in their classification. This is a necessary condition for using them to control the exoskeleton. With a limited search for the optimal hyperparameter constellation, a feedforward neural network, classifying based on the normalized sensor values, outperformed the convolutional neural network and the long short-term memory network. This is interpreted as evidence, that using these specialized architectures may require a more thorough search for optimal hyperparameters.

The conducted experiments were limited to the optimization of selected hyperparameters. Therefore further research is needed to confirm and expand on the results of this study.

Zusammenfassung

Wiederholte körperlich anstrengende Tätigkeiten können zu einer Vielzahl von chronischen Erkrankungen des Muskel-Skelett-Systems führen, die die Lebensqualität der Betroffenen beeinträchtigen. Zur Prävention dieser Erkrankungen nimmt die Verwendung von Exoskeletten in den letzten Jahren zu. Diese tragbaren Geräte üben Kräfte auf den Körper des Benutzenden aus und entlasten, indem sie einen Teil der für die Bewegung benötigten Kräfte aufbringen. Für optimale Entlastung ist es notwendig, die krafterzeugenden Elemente, d.h. die Motoren in aktiven Exoskeletten, präzise zu steuern. Ein mögliches Vorgehen für diese Steuerung besteht in der Bestimmung der Aktivität des Nutzers anhand von eingebauten Sensoren und entsprechender Regulierung der Motoren. Da die Erkennung menschlicher Aktivität basierend auf Sensorwerten aus aktiven Exoskeletten ein wenig erforschtes Gebiet ist, zielt diese Studie darauf ab, erste Erfahrungen im Gebrauch von Deep-Learning-Modellen hierfür zu vermitteln.

Die Daten für diese Studie wurden mit dem aktiven Exoskelett Cray-X aufgezeichnet. Dieses wird für die Entlastung des unteren Rückens vom Unternehmen German Bionic entwickelt.

Für diese Studie wurden vier Arten neuronaler Netzwerke trainiert: Ein hierarchisch klassifizierendes featurebasiertes Feedforward-Netzwerk, ein Feedforward-Netzwerk, dass die Rohdaten als Input erhält, ein convolutional neural network und ein long short-term memory network. Diese Netze wurden mit einem Basismodell, einer hierarchisch klassifizierenden Support Vektor Maschine, verglichen. Es wurde ein Testdatensatz mit $n = 10$ Testpersonen aufgenommen. Dieser Datensatz wurde an fünf Orten aufgezeichnet, die nicht in den Trainingsdaten vorhanden sind. Der Vergleich der neuronalen Netzwerke mit dem Basismodell dient dem Test, ob die Leistung der Deep-Learning-Modelle angemessen ist.

Die Ergebnisse zeigen, dass die neuronalen Netze konstant besser klassifizieren als das Basismodell. Das beste neuronale Netz erreichte eine Genauigkeit von $76,79\%$ und einen Matthews-Korrelationskoeffizienten von $73,48\%$ bezogen auf $k = 7$ Aktivitäten des täglichen Lebens. Auf neue Daten angewendet scheinen die trainierten Modelle robust zu klassifizieren. Dies ist eine notwendige Bedingung, um sie für die Steuerung eines Exoskeletts zu verwenden. Bei einer begrenzten Suche nach der optimalen Hyperparameterkonstellation übertraf ein neuronales Feedforward-Netz, welches die normalisierten Sensorwerte als Daten erhält, das convolutional neural network und das long short-term memory network. Dies wird als Hinweis intrepretiert, dass deren Nutzung eine gründlichere Suche nach geeigneten Hyperparametern erfordern könnte.

Der Umfang der durchgeführten Experimente war begrenzt auf die Optimierung weniger Hyperparameter. Dies macht weitere Forschung erforderlich, um die Ergebnisse dieser Studie zu bestätigen und zu erweitern.

Acknowledgements

Firstly, I would like to thank my supervisors for their constant support and help. Both supervisors provided helpful feedback at all times and invested a lot of their time to mentor this project.

I would also like to thank Team Data at German Bionic for setting up the baseline method.

Without the help of the various testers, both inside and outside German Bionic, it would have been impossible to collect a new data set for validation and testing purposes. I would like to thank everyone who gave their time to support this project. Additional thanks go to Tobias Walter, Philipp Dehnen and Joachim Fischer who organized the borrowing of a Cray-X to record this data off-site.

I would also like to thank my family and friends for their support during this time, allowing me to focus on this project. Without their help this study would not have been possible.

Special thanks are due to Philipp Claes, Joachim Fischer, Rainer Zink and Miriam Pikulski, who spent much of their time proofreading this thesis and who assisted with many fruitful conversations.

Finally, I would like to thank all the people who have been involved in my scientific education and who have made this project possible. It is not possible to list them all here, but I would like to thank you all for teaching me, learning with me, inspiring me, and enabling me to tackle this project.

Contents

List of Figures

List of Tables

List of Acronyms

CNN	Convolutional Neural Network
HAR	Human Activity Recognition
HL	Hidden Layers
IMU	Inertial Measurement Unit
LSTM	Long Short-Term Memory
MCC	Matthews Correlation Coefficient
MS	Millisecond
MSD	Musculoskeletal Disorder
N/A	Not Available
NN	Neural Network
RAM	Random-Access Memory
ReLU	Rectified Linear Unit
SVM	Support Vector Machine

Notation

This section provides an overview of the notation used. The notation is similar to that used by Goodfellow et al. (2016).

a	A scalar (integer or real)
$\boldsymbol{a}$	A vector
$\boldsymbol{A}$	A matrix
A	A tensor
$\mathbb{A}$	A set
$\mathbb{R}$	The set of real numbers
$\{0, 1\}$	The set containing 0 and 1
$\{0, 1, \dots, n\}$	The set of all integers between 0 and n
$[a, b]$	The real interval including a and b
$(a, b]$	The real interval excluding a but including b
$\frac{dy}{dx}$	Derivative of y with respect to x
$\frac{\partial y}{\partial x}$	Partial derivative of y with respect to x
$\nabla_{\boldsymbol{x}} y$	Gradient of y with respect to $\boldsymbol{x}$
$\nabla_{\boldsymbol{X}} y$	Matrix derivatives of y with respect to $\boldsymbol{X}$
$\nabla_{\mathsf{X}} y$	Tensor containing derivatives of y with respect to X
$\frac{\partial f}{\partial \boldsymbol{x}}$	Jacobian matrix $\boldsymbol{J} \in \mathbb{R}^{m \times n}$ of $f : \mathbb{R}^n \to \mathbb{R}^m$

1 Introduction

Manual material handling is a common task in a multitude of professions. If this handling is ergonomically unhealthy, e.g. by applying forces unto the same tissues in a repetitive manner without the necessary breaks for regeneration, permanent damage to these tissues may occur (de Almeida et al., 2017). This permanent damage might take on a specific form of musculoskeletal disorder (MSD), depending on the damaged region and involved tissues. According to studies conducted by the European Union, the most common areas for these disorders are the lower back, as well as the upper limbs with the most typical cause being a combination of lifting tasks combined with poor posture and a high rate of repetition. 24.7% of workers within this study reported experiencing work-related backaches and 22.8% reported muscular pain. These work-related injuries cause a significant loss of quality of life for those affected, as well as financial losses for employing companies. In 2006 MSDs led to the loss of seven million workdays in France, resulting in a loss of 710 million euros for the affected companies (Schneider et al., 2010). To prevent the emergence of these disorders a wide range of different interventions have been proposed, e.g. limiting the weight of objects to be moved. These measures have not been able to completely prevent the occurrence of MSDs for a variety of reasons.

In recent years another approach to reducing the risk of MSDs has gained popularity: Supporting workers with load-reducing tools. One class of such tools are exoskeletons, i.e. devices that generate forces and thereby reduce the amount of stress on the muscles and bones of the worker. There are a multitude of requirements for these exoskeletons: They need to be adaptive to various environments, as well as different users with varying characteristics. Additionally, they need to generate the optimal amount of support exactly when needed to provide maximum relief. This support needs to be timed precisely, because unrestrained application of forces onto the worker would reduce the worker's ability to move freely, possibly endangering him. Ideally, the exoskeleton would know the amount of needed support at any given time and be able to adjust the provided support accordingly (de Looze et al., 2016).

An approach aimed at achieving this optimal support is by using built-in sensors to estimate the current activity of the worker in real-time and adjust the activation of the force-generating elements accordingly (Jaramillo et al., 2022). As there is little research to date exploring the possibilities and limitations of this approach, this study aims at gathering experience on the classification of human activity based on exoskeleton data. In the more general research area of human activity recognition (HAR), deep learning models represent the state of the art as they are able to provide accurate and flexible predictions (Gu et al., 2021). Based on this the main models explored within this study are deep learning models.

As this specific application of human activity recognition has little research to date, this explorative study aims at gathering and communicating experience regarding a variety of questions that are not answered by the existing literature.

The first question was concerned with the appropriateness of deep learning models. Therefore a comparison was made between the trained deep learning models and a baseline method representing the conventional approach to human activity recognition: Combining hand-crafted features with classification models. This approach relies on finding useful representations of the data in the format of features via human expert knowledge. The features of the baseline method were found using expert knowledge, enabling a comparison of this approach with the approach represented by the deep learning models.

If deep learning models prove to be a better alternative to the conventional approach, it is important for practitioners to correctly estimate the amount of effort needed to produce well-functioning models. Therefore the process leading to the final model for each model structure is laid out in a comprehensive manner.

Different types of deep learning models were trained in order to compare the performance of different structures of deep learning models for this task. Each of these structures can leverage the available information differently. The comparison allowed an estimation of which information is most important for the classification algorithm.

For application in practice, the trained models must be robust. Therefore the performance of the resulting models was evaluated on a self-recorded test set, containing $n = 10$ subjects in five locations. Neither the locations nor the subjects of the test set are present within the training or validation data.

To answer these questions 15 neural networks of each network type were trained following a fixed hyperparameter optimization schedule. This hyperparameter optimization schedule puts forth four final models, i.e. the best-performing model of each network type. These final models are evaluated on the test data and the accuracy, F-measure and Matthews correlation coefficient are reported. Furthermore, the average time needed for the inference step of the models is reported, as timely classification is a necessary condition for using these models in practice. In addition, the confusion matrices of the final models, as well as tables containing the activity-specific precision, recall and F-measure were reported.

The results were contextualized by comparison with similar research. It was discussed, to what extent the research questions were answered. The experience gathered within the hyperparameter optimization process was reported.

Opportunities for subsequent research were reviewed before a conclusion of the study was drawn.

2 Background

2.1 Human Activity Recognition

2.1.1 Definition and Application

This subsection aims to define human activity recognition and give insights regarding its applications.

Human activity recognition, in the following abbreviated as HAR, is a research field concerned with the correct classification of human behavior into several categories (Vrigkas et al., 2015). An activity is defined as any movement, gesture, or physical action undertaken by the human (Gupta, 2021). This definition implies that human activity recognition can be done at different levels of granularity: Most current research focuses on activities of daily living involving whole-body movements of a single person, e.g. standing, walking, or running. There is some research that focuses on the correct classification of gestures or environment-dependent activities, e.g. drinking coffee, that can only be correctly classified by using environmental information (Wang et al., 2019a). In addition, some of the more complex activities, e.g. playing soccer, may involve a mixture of several other sub-activities such as running, standing, or shooting a ball. These sub-activities may also be performed simultaneously, e.g. jumping and throwing a ball in basketball (Jobanputra et al., 2019). This problem structure forces practitioners to decide which activity level is relevant to the current study and whether classifying multiple activities at once is permissible or whether activities are defined as a set of mutually exclusive categories.

As a research area, HAR is predominantly researched in the field of applied machine learning and computer vision. Human activity recognition is used in a variety of fields:

In home automation and security, it is used to adjust the settings of various smart home devices depending on the user's current needs (Bianchi et al., 2019). In addition, it is used in active and assisted living (AAL) to improve the quality of life of the elderly or people with impairments by increasing safety and monitoring health status. For this purpose, sensor data from various sources is combined to achieve reliable detection in real-time.

In healthcare, it is used to gain insight into changes in personal fitness levels to prevent injury, speed recovery, or improve personal training plans (Wang et al., 2019b). HAR is used to detect abnormalities such as falls or strokes. In addition, HAR can help inform medical personnel of the exact location of the person in need of assistance (Bibbò et al., 2022). A related use case, the tracking of personal fitness via step-counting based on smartphone data, has gained widespread popularity in the last years (Naqvi, 2012).

HAR is used for indoor and outdoor surveillance and monitoring. It provides cost-effective, scalable and reliable monitoring, and detection of unwanted behaviors such as aggression or vandalism (M and Thillaiarasu, 2022).

It is used in tele-immersion applications such as virtual reality, by providing the means to be physically present despite geographical distance and enabling full-body interaction in real-time via 3D representations (Ranasinghe et al., 2016).

Additionally, it may be used for traffic scheduling, advertising and other use cases (Gu et al., 2021).

2.1.2 Machine Learning

To understand their concrete application in the field of HAR, the fundamentals of machine learning are revisited.

A machine learning algorithm is defined as an algorithm that learns from data. In practical terms, this means that the algorithm is able to increase the performance measure concerning the given task, when given more experience (Mitchell, 1997).

Machine learning is used for several tasks such as classification, classification with missing inputs, regression, transcription, machine translation, anomaly detection, synthesis and sampling, imputation of missing values, denoising and density estimation (Goodfellow et al., 2016). The task partially determines the structure of the algorithm, as the data type produced by the algorithm needs to match the one determined by the task.

The performance of the machine learning model refers to its ability to perform the given task in the intended way. To enable learning, choosing the right measure of performance is paramount, because the way the algorithm updates its internal state depends on the performance measure and therefore the final behavior of the trained model depends on the chosen performance measure. However it is not always feasible to mathematically describe the intended behavior of the algorithm, therefore constructing a performance measure that generates said behavior is often difficult (Goodfellow et al., 2016).

The experience refers to the data that is available to adjust the decision-making within the model to produce an output that is closer to the desired, correct output. The format of the experience determines whether a machine learning algorithm is supervised or unsupervised.

In unsupervised learning, the data set, consisting of examples or data points, is given to learn useful properties of the data set. Therefore density estimation, denoising and clustering are typical unsupervised learning tasks.

Supervised tasks are tasks where each element of the data set is associated with a label. The task is to learn the correct mapping from the given input to the correct label. The model tries to learn the true mapping $f_{truth}(\boldsymbol{x}) = y$. To do this, it is given examples in the form of training data. Using these examples, it can adjust its own approximation $\hat{f}_{model}(\boldsymbol{x}) = \hat{y}$ to more closely match the true function.

Classification tasks are supervised learning tasks, where $y \in \{1, ..., k\}$ and the set of possible values for y has no natural ordering. When the set of possible values for y has natural ordering, the task is referred to as a regression task (Bishop, 2006).

Based on the given definitions above, human activity recognition is a classification task, as the activities do not exist in natural ordering (Gupta, 2021). Therefore the following will mainly focus on supervised learning and classification tasks.

In practice usually not all possible values $\boldsymbol{x}$ with their respective y are known. This induces the need for correctly inferring the general patterns regarding the mapping from $\boldsymbol{x}$ to y based on the available data.

This data is treated as a representative subset of all possible data. This assumption is made to ensure, that the model can induce patterns about the real-world relationship from $\boldsymbol{x}$ to y. This assumption is violated when the present data is not representative of the real-world data-generating process, i.e. the relationships within the data are not identical to the relationships in the real world. This misalignment might cause a systematic error in the inference made by the model, even if it correctly learns from the available data (Zhang et al., 2023). Based on these considerations another criterion for the evaluation of the performance of a model becomes evident: The ability to generalize beyond the given dataset.

A model which correctly learned the patterns within a dataset might be able to classify the examples within the dataset with great accuracy. A formal definition of accuracy is given in Subsection 6.1.2. The accuracy might be misleading when the model did not generalize well, meaning that the accuracy on new, unseen data is substantially lower than the accuracy on the original data set. This phenomenon is called overfitting. It could entail basing the classification on seen noise within the dataset, which does enable the model to classify the dataset better, however when confronted with new data, this noise is not indicative of the true label (Mitchell, 1997). Overfitting can be viewed as overestimating the amount of signal within the available data. The available data is a result of a mixture of signal, i.e. meaningful information which can be used to detect the label in all possible data, and noise.

There is also an opposed phenomenon known as underfitting: The model underestimates the amount of signal within the given data and does not learn everything about the patterns connecting $\boldsymbol{x}$ and y. An underfitting model has similar levels of performance on the known data set and on new data, however, this level of performance is substantially lower than theoretically possible based on the information within the data set (Zhang et al., 2023).

Ideally, the trained model would neither under- nor overfit, meaning it extracted the real-world patterns within the dataset optimally without learning anything peculiar to the present dataset. To detect underfitting domain knowledge regarding the performance of other models on similar tasks is needed, to detect overfitting the performance of the model on new, unseen data is tested. In practice this means dividing the available data into two sets: The first part is used to train the model, i.e. to adjust the internal parameters. This set is called the training data. The remaining data is used to estimate how well the trained model performs on new data. Therefore this subset of data is called the test set. The difference in the performance of the trained model on training and test set allows the practitioner to estimate the model's ability to generalize and detect overfitting. As the concrete learning process

might be depending on some external parameters, the so-called hyperparameters, a part of the training data may be separated to find the best constellation of these parameters. This part of the original dataset is called the validation set (Bishop, 2006). These hyperparameters influence how many and how the parameters of the model are being learned. The hyperparameters of the model itself are fixed once the training starts. The best constellation of hyperparameters allows for the best learning and generalization of the model. To get reliable estimations of the performance of the model it is important to choose the data for the validation and test set at random, otherwise sampling bias might lead to biased estimations.

There are two main ways to estimate the performance of the model on new data. In the holdout method, the test set is chosen once and then withheld from the training to be used afterward for the evaluation of the model. This estimation is rather unstable, depending on the amount of test data and the specific samples drawn from the dataset. A more stable way is to separate the available dataset into k parts and then training k models, where each model is being trained on different $k-1$ subsets and evaluated on the subset not used for training. Afterward, the performance of the models is averaged and the resulting estimation of the performance has a lower variance when compared to the holdout method (Bishop, 2006). This method is known as K-fold cross-validation.

When it comes to controlling the learning process and avoiding under- and overfitting, another important aspect to consider is the representational capacity of the model. The representational capacity is defined as the set of all possible functions that could theoretically be learned by the model. Which functions the model can learn depends on the chosen structure of the model and the way the model's parameters are updated. The parameters are internal states of the model that control the way it produces output. Models with higher representational capacity have a higher probability of having the wanted function within their set of possible functions. In practice it is not useful to always use models with high representational capacity. This is caused by the danger of overfitting: When providing the model with too much capacity it might use it to memorize non-generalizable peculiarities of the dataset. A necessary condition for creating a well-fitted model is starting to train a model with appropriate representational capacity.

Various architectures for machine learning models have been proposed, as well as different ways of updating these models according to new data. For the correct usage of machine learning it is important to understand that each model utilizes a set of prior assumptions. These assumptions shape the way the model learns from data. When these assumptions are appropriate and comprehensive, the model is well-equipped to handle the problem and to learn reliably and fast from the given training data. However, when these assumptions are violated, learning might prove difficult or impossible (Mitchell, 1997).

2.1.3 Sensor Data

The following entails a description of various sensors, as well as the data produced by them. This is being done, as these sensors provide the data used for the training and evaluation of human activity recognition systems.

The data for HAR stems from sensor measurements. These sensors can be characterized by their position: They are either body-worn sensors, object sensors, or environmental sensors. Sensors are called object sensors when they are attached to a specific object to detect its motion, while environmental sensors are sensors that detect the interaction with the respective environment (Wang et al., 2019a). Some research has been done regarding the integration of various sensor types (Vepakomma et al., 2015).

Accelerometers are the most commonly used type of body-worn sensors. They measure the acceleration of an object with high resolution. This acceleration data contains information about the spatial location of the sensor and the object or body part to which the sensor is attached. One advantage is anonymity since it is hardly possible to obtain private information via these sensor readings (Islam et al., 2022).

Usually, accelerometer data is combined with gyroscope data which contains information about orientation and angular velocity (Wang et al., 2019b). Accelerometers and gyroscopes are often combined into a common sensor unit called an Inertial Measurement Unit (IMU). The number of spatial dimensions measured by the IMU determine the way an IMU is referred to. Thus, an IMU that contains both a triaxial accelerometer and gyroscope is referred to as a six-axis IMU. An IMU may contain the same sensor type multiple times (Pei-Chun Lin et al., 2012).

Other body-worn sensors are magnetometers, measuring the direction, strength, or relative change of a magnetic field at a particular location. A magnetometer can be a part of an IMU.

There is a wide range of small IMUs that require little power, so they can be powered by small power supplies (Fakhri et al., 2014). Their small size allows them to be integrated into other devices such as smartphones or to outfit a single person with multiple sensors without restricting their range of motion. As these IMUs are easily accessible and integrated into the user's real life, as they are integrated into smartphones, some research has been done on HAR based on smartphone data (Bayat et al., 2014).

Other wearable sensors include barometers that measure air pressure. This information has been successfully used to determine a user's location in a multi-story building (Haibo et al., 2016).

Additionally, there is the possibility of directly detecting the activation of the muscle via electromyography, i.e. recording electrical activity within the muscle. As the utility of these devices is limited, research based on this data source is limited to specialized applications (Atzori et al., 2014).

All of the above sensor types can also be attached to objects close to the person to detect the interaction between the person and the object (Wang et al., 2019a).

Object sensors are typically used in addition to wearable sensors to detect more complex activities, as using only object sensors for HAR would limit the available data to periods when the human is interacting with a sensor-equipped object.

Ambient sensors include sensors such as radars, sound sensors, temperature sensors, or cameras. HAR based on ambient sensors has been researched mainly in the context of smart home environments. Utilizing these sensors is difficult, as they are prone to be influenced by environmental factors and thereby introduce a significant amount of noise into the sensor readings. This, as well as data privacy concerns, have led to a shift from camera and vision-based HAR to HAR based on body-worn sensors (Islam et al., 2022).

2.1.4 Data Processing

After the collection of the data from the sensors, several processing steps may be necessary before the classification is obtained by the machine learning algorithm: Preprocessing, segmentation, feature extraction, dimensionality reduction and balancing.

The raw sensor data might be subject to noise because of several reasons: The sensors might be miscalibrated which induces the need for removing or compensating for several errors before using the sensor outputs (Rui Zhang et al., 2014). The sensor may be malfunctioning or the sensor readings may be affected by ambient noise. There is also the possibility that the position of the sensor may change, particularly with body-worn sensors where certain activities may cause the sensor to move from its original position.

To handle this noise a wide range of preprocessing filtering techniques have been applied, such as the Wavelet filter, Kalman filter and the Low-pass filter (Minh Dang et al., 2020).

Sensor sampling rates are typically small compared to the duration of human activity. As these activities involve time-related patterns of movement, a singular sensor reading typically does not contain enough information to correctly estimate the activity seen. Therefore segmentation is required to aggregate enough sampled data into a collection to correctly estimate the activity.

There are various segmentation approaches available: Time-driven windows segmentation separates the stream of sensor readings into fixed-sized windows. Event-driven windows segmentation segments the sensor data into windows based on estimated events. The action-driven window segmentation detects the windows, where individual activities occur (Minh Dang et al., 2020).

There are two main ways to extract useful information from the segmented data. Representation learning aims at learning which features are useful. This approach is discussed in detail in Subsection 2.3.1. Before representation learning became feasible for widespread application, hand-crafted feature creation and selection was the predominant approach to bringing the data into an appropriate representation

for the classifier. As the advantages of representation learning can only be fully understood after understanding the traditional approach, the following aims at giving an overview of the conventional approach relying on hand-crafted features. Figure 1 illustrates the process of HAR via the traditional approach.

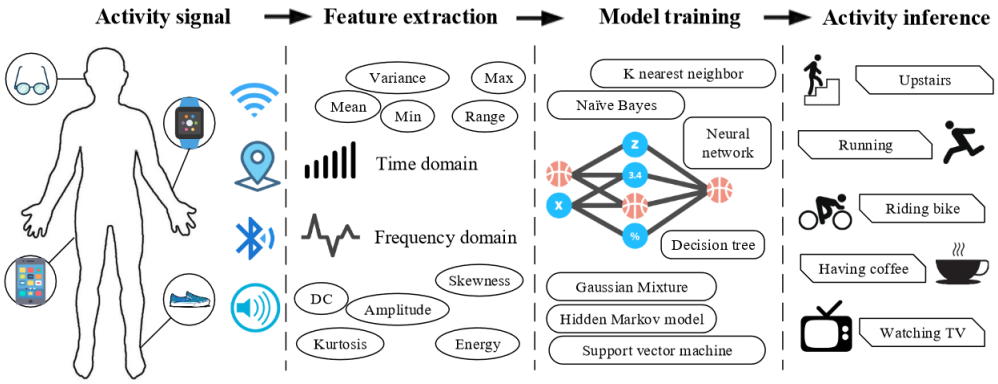

Figure 1: Process of Activity Recognition following the conventional approach (Wang et al., 2019a)

These segmented windows are used to extract a set of meaningful features. There are three general approaches to the generating of features: The time domain approach, the frequency domain approach and the wavelet transformation.

The time domain approach extracts information about the amplitude variations of the signal over time. Therefore, the features typically consist of statistical moments of the seen distribution within the window, such as mean, variance, skewness and kurtosis (Suto et al., 2017). Using this approach has the advantage that no transformations are required beforehand, allowing these features to be computed quickly even on computationally limited edge devices (Dargie, 2009). On the other hand, these features do not capture the information inherent in the signal frequency.

The frequency domain approach is based on the variation of the frequency of a given signal over time. Features based on the frequency domain approach contain information about the proportion of received signals within specific signal bands. Examples of frequency domain features include spectral entropy, spectral power, peak power and peak frequency. Compared to features based on the time domain approach, the calculation of features based on this approach is computationally expensive (Minh Dang et al., 2020).

Wavelet transformation decomposes a signal into several functions called wavelets. This provides a high-resolution representation of the original signal via the decomposed functions, which can be used to calculate each wavelet's energy. Examples of features based on this approach include root mean square and mean absolute value. Similar to features from the frequency domain approach, the computation of these features is computationally expensive (Minh Dang et al., 2020).

After generating a set of features, this set is typically reduced to a set of the most informative features according to an evaluation criterion in a process known as

feature selection. This is done to reduce the dimensionality of the input data. An explanation of why this is desirable is given in Subsection 2.3.1 in the discussion of the curse of dimensionality. Another advantage of feature selection is that it allows the machine learning algorithm to compute the classification faster, as it does not have to deal with redundant information. Unlike other dimensionality reduction techniques (e.g. principal component analysis), feature selection does not alter the original representation of the data (Sánchez-Maroño et al., 2007).

Depending on the training data, which can be labeled, unlabeled, or partially labeled, there are three classes of feature selection algorithms, i.e. supervised, unsupervised and semi-supervised feature selection algorithms. Depending on the feature selection process, the algorithm can be classified as a filter, wrapper, or embedded method.

A filter method works by calculating a performance score for each feature and then filtering features based on this score. This technique is popular due to its wide applicability and lower computational requirements. In addition, feature selection can be applied before the final classification algorithm is decided (Chandrashekar and Sahin, 2014).

Wrapper methods work by training models with subsets of the available features and selecting features based on the performance of the respective models. Wrapper methods are therefore based on the assumption that informative features help models in different constellations. Wrapper methods can lead to better performance compared to filter methods, as wrapper methods can take into account the appropriateness of the prior assumptions of the model regarding the data. In addition, wrapper methods are able to account for the effects of highly correlated variables that contain similar information. These methods are computationally expensive, as different models with different subsets of features need to be trained and evaluated (Jovic et al., 2015).

With embedded feature selection, feature selection is embedded in the training process, meaning that the model learns to use the most informative features (Jovic et al., 2015).

The process of feature creation and selection is guided by the domain knowledge of the experts for the respective tasks. The choice of which features to create and which selection methods to use is typically based on an informed process of trial and error (Wang et al., 2019a).

When the labels of a dataset appear for an unequal amount, the dataset is called imbalanced. When performing classification this might be problematic, as the performance measure might depend predominantly on the majority classes. As a result, the machine learning model might learn to classify the majority classes well while the classification of the minority classes remains erroneous. To prevent this balancing techniques are used. These techniques are categorized either as undersampling or oversampling techniques. Undersampling techniques refer to techniques which remove samples from the majority classes, until the resulting dataset is less im-

balanced. Oversampling aims at adding additional samples to the minority classes (Ganganwar, 2012).

For the final classification of the seen activity based on these resulting sets of features, various classes of typical machine learning models have been used successfully. These include tree-based models, Support Vector Machines, Hidden Markov Models and Nearest Neighbor algorithms (Chen et al., 2021).

2.2 Human Activity Recognition based on Exoskeleton Data

The following section aims to inform about the application of human activity recognition on exoskeleton data. First, basic knowledge about exoskeletons is given in Subsection 2.2.1. Then the challenges of HAR based on exoskeleton data are discussed in Subsection 2.2.2, before a review of the current state of the art is given in Subsection 2.2.3.

2.2.1 Exoskeleton

Exoskeletons are wearable devices that generate forces or torques on at least one human joint to support the execution of physical activities (Toxiri et al., 2019). Based on this definition exoskeletons might be characterized as follows: The first categorization of exoskeletons is based on the method utilised to generate forces. There are currently four categories of force-generating exoskeletons: Passive, semi-passive, active and hybrid exoskeletons.

Passive exoskeletons are devices that store the energy generated by the user's motion in elastic elements and support the user when these elements decompress (de Looze et al., 2016). Three types of elastic elements are used in passive exoskeletons: Springs, flexible beams, or elastic bands (Ali et al., 2021). Depending on the elastic elements, these devices can be lightweight and worn under clothing. The smaller size is an advantage as it allows the user to work in smaller environments, use additional protective equipment, or hide the use of the exoskeleton, which may be desired when using these devices in public.

The disadvantage of passive exoskeletons is their lack of flexibility, which means that these elastic elements statically apply forces regardless of current need. In addition, the amount of support these devices can provide is fixed, as it depends on the elastic materials incorporated into the exoskeleton and cannot be adjusted. Furthermore, the amount of support provided by passive exoskeletons is limited by the amount of energy given by the user, which could mean that the support shrinks as the user becomes fatigued and unable to stretch or compress the elastic elements. Some evidence suggests that the use of passive exoskeletons shifts the load to other muscle groups, which may be desirable depending on the application (de Looze et al., 2016).

Semi-passive exoskeletons are exoskeletons that rely on elastic materials to generate the supporting forces while incorporating mechanisms to modulate the behavior of these materials. This regulation can be performed by the user or automated based

on built-in sensors (Grazi et al., 2022). As a result, semi-passive systems have a higher degree of adaptivity compared to similar passive systems (Grazi et al., 2020). A disadvantage is that these additional elements increase the weight of the exoskeleton and if the modulation is sensor-based, care must be taken to ensure that the support is provided at the correct time.

Active exoskeletons are characterized by generating forces and torques themselves via built-in motors or pneumatic actuators. The advantage of these exoskeletons is their flexibility, as the activation of the motor can be adjusted according to the user's needs. As these devices are self-powered, they can provide the majority of the power needed to perform the activity. Some evidence suggests that the usage of active exoskeletons reduces muscle activity in the supported regions (de Looze et al., 2016).

As active exoskeletons are capable of changing the user's body position, measures must be taken to ensure that the device does not exert forces that could endanger the user. Another disadvantage of active exoskeletons is their dependence on power sources. This dependence might limit the amount of time the device can provide continuous support.

For an active exoskeleton to function properly, several parts are required: Motors to generate the supporting force, straps to transfer these forces to the user's body, sensors to detect the user's current condition and a power supply to power the motor. All of these components add to the weight of the exoskeleton, potentially reducing the user's range of motion and ability and willingness to wear the exoskeleton for extended periods.

The combination of passive and active assistive support in a hybrid exoskeleton is a new area of research and there are few studies to date exploring the potential of this type of exoskeleton (Missiroli et al., 2022). Based on the little amount of available research a more in-depth explanation is not provided.

Exoskeletons may be categorized by the supported region. Lower body exoskeletons support the lower limbs, e.g. legs and feet, while upper body exoskeletons support the upper limbs. There are also full-body exoskeletons and exoskeletons that support only a single joint.

Another categorization is concerned with the used materials:

Soft exoskeletons are devices that consist of multiple garments placed around the supported region. Support is generated by pulling these garments together by decompressing the elastic material, typically via a cable or strap. Overall, most soft exoskeletons are passive exoskeletons that rely on the forces generated by changing the state of the elastic material (Ali et al., 2021).

Rigid exoskeletons use hardened, inelastic parts that are typically required to transmit the force generated in the actuators to the user. Rigid exoskeletons often increase the amount of space required for the user, reducing their usability in certain scenarios. While the forces applied by soft exoskeletons are usually parallel to the user's body, the forces applied by rigid exoskeletons are typically perpendicular to

the user (Toxiri et al., 2019). This is illustrated in Figure 2. Misalignment problems can occur when using a rigid structure (Tiseni et al., 2019).

Some work has been done on the combination of soft and rigid components (Tiseni et al., 2019).

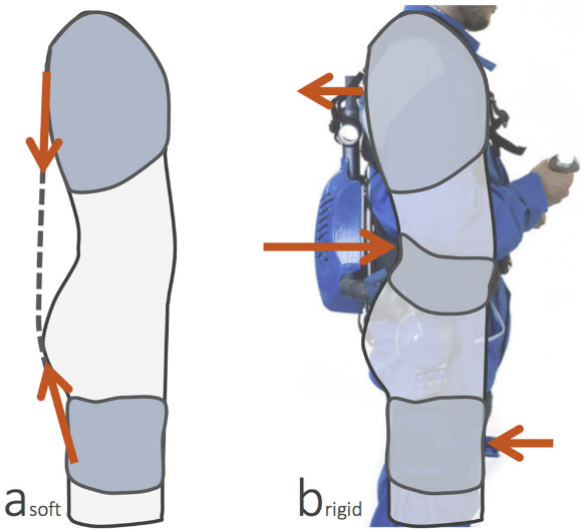

Figure 2: Forces applied via exoskeletons (Toxiri et al., 2019)

2.2.2 Requirements and Challenges

Below is a discussion of the motivation for HAR on exoskeleton data, as well as a discussion of potential problems with the detection of human activity based on exoskeleton data.

The main problem with passive exoskeletons is their lack of flexibility to provide support only when needed. Active exoskeletons have the advantage of being able to regulate the current level of support by controlling the internal motors. Ideally, the exoskeleton would be able to detect the user's current need for support and provide it without input from the user (Jaramillo et al., 2022).

Since the amount of support required varies between activities and even within an activity, a correct classification of the current activity could provide the means to successfully adapt the support to the user's needs (Poliero et al., 2019). This classification would have to fulfill several requirements to be useful for managing the support of the exoskeleton:

The classification must be robust, i.e. not influenced by external, irrelevant factors, and able to correctly classify the current activity regardless of the user's concrete environment. A classification algorithm that is susceptible to environmental influence is less reliable and could therefore hinder acceptance and usage (Poliero et al., 2019). Classification should be performed in real-time, i.e. with as little delay as possible, as delays reduce the usability of the device. Another requirement is the

ability to perform the classification with as little processing power as possible. This is necessary to enable edge computing, i.e. the classification on the exoskeleton. This is necessary because sending the data to servers to perform the classification would introduce a dependency on a reliable connection to these servers, as well as greater latency. Another desirable attribute of human activity recognition on exoskeleton data is stability, i.e. not producing noisy and rapidly changing classification results when the user is performing similar activities. Rapidly changing classification might lead to quickly changing behavior of the exoskeleton, which might make it less suitable for real-world application. The overall accuracy of the model must be adequate, otherwise, the exoskeleton would provide support below its capabilities or even endanger the user by providing unwanted support (Toxiri et al., 2019).

A more in-depth review of the respective problem is given to facilitate the understanding of the current solutions.

One problem lies in the positioning of the sensor: As body-worn sensors may change their position after prolonged wearing of the sensor, the processing needs to be flexible enough to robustly classify the activity, even if the sensor position changes.

Additionally, the classification needs to be robust against different body types, user states and environments. As age, sex, height, weight, fatigue, execution speed and many other factors influence the specific way an activity might be performed, the classification needs to be flexible enough to classify vastly different sensor inputs into the same activity class. The same activity has a wide range of different sensor readings, as all the above-mentioned factors add variety. This problem is known as intraclass variety (Vrigkas et al., 2015).

Another problem that needs to be overcome for reliable, accurate classification of human activity based on sensor data are the limitations imposed by reliance on domain knowledge. As seen in Subsection 2.1.4, the traditional approach to HAR relies on feature creation and selection based on the domain knowledge of human experts. However, this domain knowledge is often limited to shallow features. Complex, high-level features are hard to explore so the input for classification algorithms was often limited to sets of shallow features (Hammerla et al., 2016). There is the possibility that necessary information has been removed by reducing the available data to sets of shallow features and therefore the resulting models would underfit (Wang et al., 2019a).

2.2.3 Literature Review

The following aims at providing an overview regarding the current state of the art of human activity recognition based on active exoskeleton data.

The research regarding human activity recognition based on active exoskeleton data is an emerging field. As the research and usage of exoskeletons begin to accelerate, some research is being done regarding HAR on active exoskeleton data. However, the amount of literature in this field is still limited to singular investigations. Large-scale studies are still missing from the literature (Pesenti et al., 2023). For this thesis,

the literature review has been limited to studies, where the focus of the research is similar to the current study.

There is evidence that it is possible to use IMU data in a combination with the information regarding the angles of the legs to use Support Vector Machines to differentiate three activities of daily living with high accuracy. The authors of this study performed the recognition to modulate the exoskeleton's behavior based on real-time classification of the seen activity direct onboard the exoskeleton, i.e. via edge-computing (Poliero et al., 2019).

In human activity recognition deep learning has become the predominant approach to classifying human activity (Jobanputra et al., 2019). The main reason for this is the ability of deep learning models to learn useful representations of the data themselves. By doing so they can avoid some of the dependency on human domain knowledge. These models have greater representational capacity than other machine learning models. As a result, they have the capacity to model more flexible functions. A more in-depth explanation of this behavior is given in Subsection 2.3.1. The metrics used in the discussion of the literature below are laid out in Subsection 6.1.2.

A recent study by Jaramillo et al. (2022) concerned with real-time HAR classification based on sensor values of active exoskeletons trained models for distinguishing $k = 8$ activities of daily living. The data basis for this experiment was collected from an exoskeleton intended for the relief of the lower back, developed by Hyundai Rotem. Said exoskeleton contained a nine-axis IMU with a triaxial accelerometer, gyroscope and magnetometer. An explanation of this sensor types is given in Subsection 2.1.3. Additionally, the models used knowledge regarding the angle of the legs. The data used to train, validate and test the model came from $n = 4$ subjects, between 25 and 30 years of age, recorded in a singular environment. Regarding preprocessing the data was zero-centered and then scaled to be within $[0, 1]$. Additionally, a moving-average filter was applied. The authors experimented with five different deep learning architectures: Convolutional neural networks, recurrent neural networks, long short-term memory networks, bi-directional long short-term memory networks and neural networks equipped with gated recurrent units. However, they do not communicate how many different hyperparameter constellations they tried before reaching their final models. A different protocol was used, i.e. the order of activities for training and test data differs. The authors reported various models with an inference time smaller than nine milliseconds and an accuracy of over 86%, with the best model achieving an inference time of 4.97 ms and an accuracy of 97.56%. As the same subjects were used for the training and test set, as well as recording both sets took place in the same environment, it remains questionable how well these models generalize beyond this environment and these four subjects.

Another recent study by Pesenti et al. (2023) explored the possibility of simultaneously classifying the seen activity and the manipulated weight. The data for this was collected from up to five IMUs, using the built-in accelerometer and gyroscope. The authors explored the effects of removing IMUs from the available data pool on the performance of resulting models. The data was collected from $n = 12$ sex-balanced

subjects, aged around 25. The raw data was passed through a zero-lag, fourth-order Butterworth filter and afterward scaled to be within $[-1, 1]$. Regarding the overall structure of the model, the networks employed consisted of two long short-term memory unit layers with 100 and 50 units, before the data was inserted into a fully connected feedforward layer with 20 units. This feedforward layer was connected to two output layers because the activity recognition task was split into two tasks: The first concerned itself with the separation of standing, walking and interacting with the payload, whereas the second task classified lifting or lowering the payload. For the first task, a median accuracy of 89.49% and maximum accuracy of 91.90% was reported, as well as a median F-measure of 89.92% and a maximum of 92.08%. For the interaction task, the median accuracy was 96.33%, the maximum 97.14%, the median F-measure 91.68% and a maximum score of 92.38%. This performance was based on subject-specific models, i.e. for each of the $n = 12$ a different classification model was trained. The protocols for the training and test set were different, however, the involved subjects, as well as the environment was identical in both sets. Again the authors did not convey how many hyperparameter constellations were tried before arriving at the final model structure.

2.3 Deep Learning

2.3.1 Fundamentals of Deep Learning

This subsection aims at explaining the fundamental structure of deep learning models. This is needed to understand the structure and order of the experiments reported in the following chapters.

The inspiration for the development of artificial neural networks stems from the dream of creating human-like intelligence. The natural approach was to build structures with an internal order similar to the one of the human brain. Within the human brain complex calculations are performed by aggregating the calculations made within a multitude of individual cells, called neurons. Each of these neurons is connected to a small subset of other neurons, from which it gets information in the format of chemical signals and is connected to another small set of neurons, to which it returns information (Russell et al., 2016). Overall, it is a highly-parallelized structure, in which iteratively gathering and aggregating information enables the processing of complex tasks. However modern artificial neural networks are at best loosely inspired by the human brain and the development of these networks is more driven by principles of mathematics and engineering than by recreating biology (Goodfellow et al., 2016).

The current success of artificial neural networks can be explained by their ability to mitigate the effects of a phenomenon called the curse of dimensionality. This phenomenon refers to the problem within machine learning (ML), that the number of possible configurations for $\boldsymbol{x}$ grows exceedingly fast when the dimensionality of $\boldsymbol{x}$ increases. For a supervised learning task, which tries to learn the true mapping $f(\boldsymbol{x}) = y \ \forall \boldsymbol{x}$, it poses the problem that the available data will probably not fill

the whole possible feature space. If the dimensionality of $\boldsymbol{x}$ increases, the amount of space for which the model does not have examples increases quickly (Bishop, 2006). Therefore the ML algorithm needs to generalize from the available data appropriately to solve the given task. This ability depends on the prior assumptions of the model, which guide the learning of the model. Conventional machine learning models often implicitly make use of the local constancy prior, which states that the function should not change within a small region. This means that $f(\boldsymbol{x}) \approx f(\boldsymbol{x}+e)$, for any small change e (Goodfellow et al., 2016). Relying solely on this prior can not be enough to tackle problems involving high-dimensional data, as the respective feature spaces are too wide and therefore the inference based predominantly on the most similar, i.e. nearest example becomes too noisy.

Neural Networks can partially handle this problem by incorporating another prior assumption. This prior is called the manifold hypothesis and states that within the high-dimensional feature space, the data is concentrated within a connected lower-dimensional region, the manifold. For real-world application this assumption seems to be approximately true for many tasks, e.g. for the translation of text: The existing words do inhabit a small space within the space of all possible words, as several constraints, such as length, limit the space of developed and used words (Goodfellow et al., 2016).

To understand the structure of a neural network it is crucial to first understand the building blocks of these networks, the units. A unit receives input $\boldsymbol{x}$ from other units. This input is weighted via $\boldsymbol{w}$, meaning each unit can weigh information from previous units differently. The weighted sum of these inputs is then added to the bias b of the unit, a numerical term influencing the general tendency of the unit to produce high activation. Equation 1 illustrates, how the activation of a unit is depended on the input vector coming from previous units $\boldsymbol{x}$ and the unit-specific weighting of this information via $\boldsymbol{w}$ and the bias b.

$$z = \boldsymbol{w}\boldsymbol{x} + b \tag{1}$$

This activation z is put through an activation function, after which it is called the output of the unit and is given to the next units. This activation function is typically non-linear. This is because a network with only linear activation functions would itself be a strictly linear function incapable of learning non-linear functions (Zhang et al., 2023).

Usually, several units are combined into a layer, with the name of the layer depending on the position of the layer. The first layer, which receives the input directly from the data, is called the input layer. The output of the final layer is the output of the network, therefore this layer is referred to as the output layer. The layers in between are called hidden layers (HL). These namining conventions are illustrated in Figure 3. The number of units within a layer is called the width of the layer, whereas the number of layers is known as the depth of the network. With the advent of better algorithms and hardware, training deeper networks became feasible, hence the name deep neural networks or deep learning.

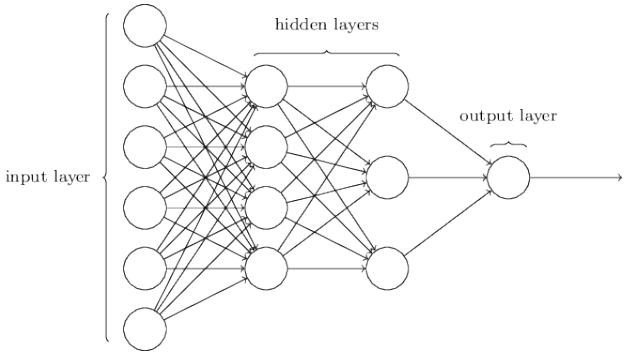

Figure 3: Structure of a Neural Network (Nielsen, 2015)

Another advantage of neural networks is based on their ability to learn representations of the data. The calculations in the neural network may be interpreted in the following manner: The hidden layers bring the original representation of the data into a format, which is better suited for the final classification. Afterward, the classification is done based on this informative representation. Updating the weights and biases within the learning process equates to finding the best-suited classification function and the best representation of the data, on which to base the final classification. Neural networks are therefore able to learn high-level features and abstract representations of the data as a part of the training process (Mitchell, 1997). This avoids the reliance on human experts to find the best representation of the data, which is advantageous, see Subsection 2.2.2.

During the training the weights and biases of each unit within the network get updated iteratively, to match a function with less error. The error of the network is calculated based on the distance between the outputs of the network and the true, expected output. Depending on the task and data type of the output type of the network, different cost functions are available to measure the error of the network.

For this study, the most important loss function, used for all networks within Chapter 5, is the Cross-Entropy Loss. To leverage the available information optimally, the concept of maximum likelihood is used by this loss function. As the data is assumed to be independent, the conditional distribution of the labels given the observed data $P(\boldsymbol{Y}|\boldsymbol{X})$ may be re-written as

$$P(\boldsymbol{Y}|\boldsymbol{X}) = \prod_{i}^{n} P(\boldsymbol{y}^{(i)}|\boldsymbol{x}^{(i)}) \tag{2}$$

As maximizing this likelihood is equivalent to minimizing the respective negative log-likelihood, this may be written as:

$$-\log P(\boldsymbol{Y}|\boldsymbol{X}) = \sum_{i=1}^{n} -\log P(\boldsymbol{y}^{(i)}|\boldsymbol{x}^{(i)}) = \sum_{i=1}^{n} l(\boldsymbol{y}^{(i)}, \hat{\boldsymbol{y}}^{(i)}) \tag{3}$$

With the loss function l being calculated over all k classes in the following manner:

$$l(\boldsymbol{y}^{(i)}, \hat{\boldsymbol{y}}^{(i)}) = -\sum_{j=1}^{k} y_j \log \hat{y}_j \tag{4}$$

Equation 4 is similar to the definition of entropy, motivating the name of this loss function. This loss function may either be interpreted as maximizing the likelihood of the observed data or as minimizing the level of surprise encountered upon prediction. For discrete classification tasks this is a popular choice regarding the loss function, based on the useful properties utilized by this loss, as well as enabling efficient computation of gradients (Zhang et al., 2023).

As the training problem of deep neural networks is typically nonconvex, the optimal parameter values cannot be found analytically. After input data was passed through the network in a forward pass, the loss $J(\theta)$ is calculated based on the chosen cost function J and the parameters of the model θ. To reduce the error of the network the information is required how the loss depends on the current weights and biases θ of the model. This information is present within the gradient $\nabla_\theta J(\theta)$. The influence of each parameter on the overall cost of the current network may be calculated via the chain rule of calculus. In the scalar case, this rule states that for the functions $y = g(x)$ and $z = f(g(x))$ the derivative of z respective to x is

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx} \tag{5}$$

This rule can be used to calculate the gradient for nested functions. A deep neural network can be viewed as a collection of nested functions, where the final output is created by applying all the functions represented by the layers in a nested fashion, see equation 6.

$$\hat{\boldsymbol{y}} = f_4(f_3(f_2(f_1(\boldsymbol{x})))) \tag{6}$$

Therefore, the chain rule of calculus can be used to calculate all the needed gradients. Generalizing this rule beyond the scalar case for $\boldsymbol{x} \in \mathbb{R}^m$, $\boldsymbol{y} \in \mathbb{R}^n$, with g mapping from $\mathbb{R}^m$ to $\mathbb{R}^n$, yields

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j}\frac{\partial y_j}{\partial x_i} \tag{7}$$

Using vector notation this can be written as

$$\nabla_{\boldsymbol{x}} z = \left(\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}\right)^T \nabla_{\boldsymbol{y}} z \tag{8}$$

where $\dfrac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}$ is the $n \times m$ jacobian matrix of g, i.e. the matrix containing all partial derivatives. The backpropagation algorithm consists of performing such a jacobian-gradient product for each operation within the computational graph of the neural network. When doing so for a network, several subexpressions would need to be evaluated several times. The backpropagation algorithm stores these subexpressions,

thus increasing the evaluation speed, especially on large networks (Goodfellow et al., 2016).

In practice, calculáting the gradients on the entire dataset is avoided due to memory issues, as the size of the available datasets often exceeds the available RAM. Knowledge from statistical theory is used to approximate the information of the whole dataset by using smaller subsets of the training data. The learning process, therefore, consists of doing a forward pass of all the data within the chosen subset, called minibatch, calculating the loss and the respective gradient for every parameter and updating the parameters, before passing the next minibatch through the network. In practice using small batch sizes has proven to improve the model's capability of generalizing. This might be due to the noise injected via the randomly sampled examples within each minibatch, which might avoid halting in local minima (Mitchell, 1997). When updating the parameters the gradients are added to the current parameter value, after weighting the gradient by the learning rate μ. Choosing a high learning rate will speed up the initial learning but may prevent the learning from getting close to the desired minimum. A lower learning rate may require a longer learning process but may allow approaching the desired minima more closely. It has become common practice to change the learning rate during the training process. There are several algorithms to do this, one widely accepted algorithm is called adam, derived from adaptive moment estimation (Kingma and Ba, 2017). This widespread acceptance stems from a multitude of useful properties for stochastic optimization, as well as good empirical results.

The activation function chosen for the units depends on the given task. The activation function of the output layer is determined by the expected data type: When doing regression tasks, the final layer may contain units with a linear activation function, whereas for multi-class classification a typical activation function is the softmax function. Each unit within the output layer corresponds to one of the k possible classes and the network returns a vector of length k containing the probabilities, that the given input data belongs to the respective class. The softmax function is

$$softmax(\boldsymbol{z})_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)} \tag{9}$$

with $z_i = \log \tilde{P}(y = i|\boldsymbol{x})$. For mutual exclusive classification, the final classification is simply the class with the highest probability assigned to it, which is the class with the highest value within $\boldsymbol{z}$ (Zhang et al., 2023).

The activation function determines how a unit is referred to, so a unit with the softmax activation function is called a softmax unit. As for the activation function of the hidden units, these are typically non-linear functions that introduce non-linearity into the neural network. This nonlinearity is desired because solving complex tasks typically involves handling non-linearity. There are various possible activation functions for the hidden units available, a usually acceptable choice is using Rectified Linear Units (ReLU) with the activation function $f(x) = \max(0, x)$, which has the useful property of having the derivative

$$f' = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \end{cases} \tag{10}$$

The gradient $f'(0)$ is not defined, however, this can be ignored, as the input 0 is typically a product of underflow, i.e. rounding of small non-zero values, therefore returning using either 0 or 1 is heuristically justified (Goodfellow et al., 2016).

Regarding the structure of artificial neural networks another important theorem is the universal approximation theorem, which states that a network with a linear output layer and at least one hidden layer with a "squashing" activation function can approximate any Borel measurable function from one finite-dimensional space to another with any desired nonzero amount of error, given enough hidden units (Hornik et al., 1989). Any continuous function within a closed and bounded subset of $\mathbb{R}^n$ is Borel measurable. Universal approximation theorems have also been proven for a wider class of activation functions, including ReLUs (Leshno, 1993). This theorem provides the theoretical basis, that the model class of neural networks is in general appropriate to handle any task given the conditions above. It has been shown that using models with less depth increased the needed width dramatically. Therefore deeper networks are being used, which can represent the desired function with fewer units. Being able to represent the wanted function does not mean that this function gets learned within the learning process (Mitchell, 1997). This may be because the learning halts in local minima, as a result of gradient-based learning.

As a consequence of the theorem above it may be stated that depending on the size of the network, artificial neural networks have a large representational capacity, as defined in Subsection 2.1.2. To guide the learning process to generalize better, several regularization methods are applied. These are intended to influence the learning process by adding additional constraints so that the resulting model generalizes better (Goodfellow et al., 2016). It has been shown that bigger, well-regularized models typically outperform smaller models with little regularization. As there are various ways in which a neural network may be regularized, the following discussion only entails methods used within Chapter 5.

The first regularization method is dataset augmentation. The idea is that using larger datasets makes it more difficult for a model with a fixed representational capacity to overfit. Via dataset augmentation, the existing data is used to generate additional, similar training examples. In general, these additional samples might be generated by adding small, but for the classification irrelevant noise to the examples and adding these noised examples to the dataset (Gu et al., 2021). The enhanced performance may be explained by the inclusion of examples, which probably exist in reality, but were not included in the original training data (Goodfellow et al., 2016). Figure 4 shows the creation of augmented data by either adding small random noise or a small constant term.

Regularization by dropout refers to randomly removing a portion of the hidden units and then training and adjusting the parameters of the remaining units. For each minibatch, another randomly selected portion of the hidden units is excluded from

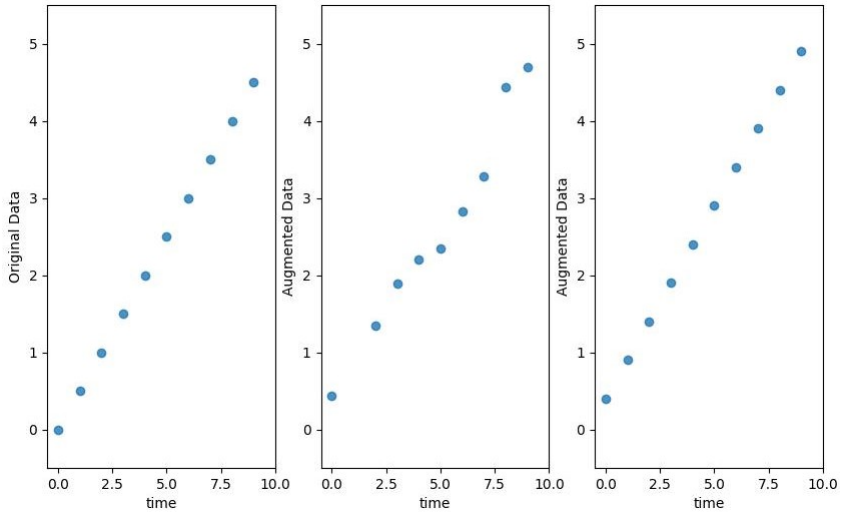

Figure 4: Examples for Data Augmentation

forward propagation and learning. After training, the parameters of the model are rescaled according to the dropout rate used. This adds uncertainty to the presence of multiple features for the final classification, forcing the model to base its final classification on the aggregation of multiple pieces of information, rather than relying on singular features that may not be present. Another way to understand the effect of dropout is to interpret the final model as a mixture of different submodels that share parameters. By interpreting dropout in this way, the additional performance on the validation data can be explained by the aggregation of different models, making the resulting model an ensemble model (Hinton et al., 2012). The effectiveness of this method has been demonstrated for a wide range of networks, albeit at the cost of additional training time or making larger networks necessary (Labach et al., 2019). Figure 5 illustrates dropout.

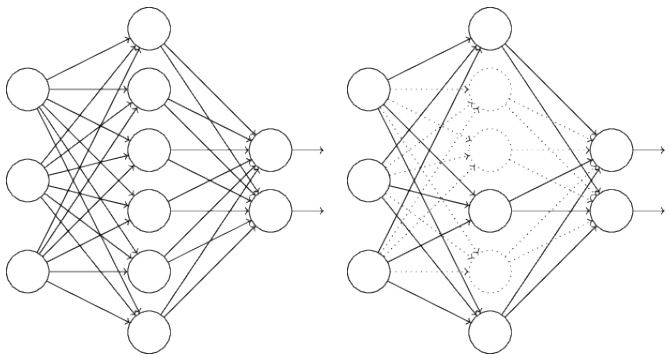

Figure 5: Full Network vs. Network affected by Dropout (Nielsen, 2015)

Another regularization method is early stopping. This means stopping the learning process on the training data, as soon as the performance of the model on the validation data does not improve anymore. This is motivated by the idea, that this stopping point marks the time when the model stopped learning generally use-

ful patterns and began overfitting on the training data by learning overly complex functions based on observed noise (Caruana et al., 2000).

Batch normalization refers to the process of z-standardizing the inputs of the respective layer, i.e making the resulting distribution centered around zero with a standard deviation of one. While the exact reasons for the success of this method remain not fully known, reliable enhancement of network performance has led to widespread usage (Zhang et al., 2023).

2.3.2 Convolutional Neural Networks

There are various classes of neural networks which differ in design to enable better processing of certain characteristics of the input data. One expansion on the classical fully-connected feedforward network is the convolutional neural network (CNN), which is defined as a feedforward network that has at least one layer performing the convolution operation. This operation consists of multiplying a certain weight matrix, also known as a receptive field, over a part of the input data, where the size of the receptive field is smaller than the size of the input data. This field gets moved over all of the grid-like input data and results in a grid-like feature map (LeCun et al., 1998). Figure 6 illustrates a possible result by showing a color-coded feature map besides the original input image.

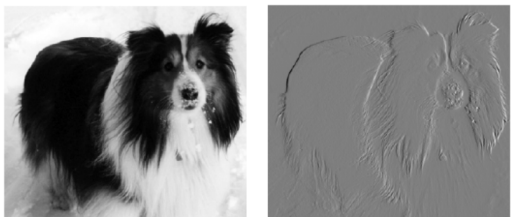

Figure 6: Input Image and resulting feature map (Goodfellow et al., 2016)

Applying convolution leverages various ideas: Sparse interactions means that the number of parameters used may be reduced significantly by using a convolution layer instead of a fully connected layer. This is because not every input unit is connected to every output unit via a separate weight parameter. Matrix multiplication with m inputs and n outputs requires a runtime of $O(m \cdot n)$, whereas applying a kernel or receptive field, which limits the number of connections from the input to the output to k makes calculation in $O(k \cdot n)$ possible. In deeper networks, the later layers still can indirectly interact with large proportions of the input, resulting in a significant reduction in used parameters, while keeping the performance of said network high. Applying the same kernel across all of the input data can be interpreted as a form of parameter sharing, as the same set of parameters gets used for the entirety of the input data, resulting in smaller memory requirements compared to models, where each input unit is connected to all the output units. This parameter sharing causes the convolution layer to be equivariant to translation. An equivariant function is a function, where a changed input results in an equally changed output, i.e. $f(x)$ is

equivariant to the function g if $f(g(x)) = g(f(x))$. Specifically, this translates to the convolution layer being able to detect certain patterns within the grid-like topology of the input data, creating feature maps displaying the occurrence of the feature described by the receptive field. Additionally, multiple convolution layers may be combined within a network, enabling the detection of high-level patterns from the data in an efficient manner (LeCun et al., 1998).

The various hyperparameters for the construction of a convolution layer are as follows: The kernel size determines how many datapoints are evaluated at once. Bigger kernel sizes, i.e. local receptive fields, are able to detect more complex patterns, albeit at the cost of additional parameters. Small kernel sizes have been shown to work well (Grzeszick et al., 2017). The number of filters determines how many kernels are being applied to the input, where each filter detects its own features. Therefore applying various k filters to the same input results in k feature maps. The stride determines the step length between the application of the kernel to the data. Choosing a stride higher than the kernel size is discouraged, as this would entail not using certain data points at all. Using a convolution layer introduces the prior assumption into the network, that the function to be learned by the layer only contains local interactions and is equivariant to translation (Goodfellow et al., 2016).

Typically pooling is performed after convolution. This operation gathers the information within a region by a summary statistic. This makes the internal representation of the network approximately invariant to small translations of the input, meaning that small modifications of the input hardly change the pooled outputs. This is especially useful when the detection of the presence of a pattern is more important than knowing the exact location of the pattern. Adding a pooling layer adds the prior assumption, that each unit should be invariant to small translations. Regarding the summary statistic for pooling, there are several options, with using the maximum value within a region a commonly used one (Wang et al., 2019a). Another hyperparameter for pooling is the aggregation size, determining how many elements of a sample are to be aggregated by the singular summary statistic. Analogue to the convolution there is also the stride hyperparameter, regulating how big the overlap between the regions to aggregate is.

When processing data in a grid-like structure, e.g. pictures or time-series data, typically convolution and pooling operations are performed before processing the resulting feature maps within fully connected layers (Gu et al., 2021). As these feedfordward layers process input in the format of vectors, the feature maps get flattened, i.e. transformed to a vector.

2.3.3 Recurrent Neural Networks

Another relevant extension of feedforward neural networks are recurrent neural networks, a group of networks specialized in the processing of sequential data (Gu et al., 2021). This specialization enables them to successfully handle long-term dependencies.

Similar to convolutional neural networks, these networks make heavy use of parameter sharing, which makes it possible to generalize across several parts of the model. This enables the detection of patterns regardless of the specific point in time of occurrence. This parameter sharing enables most recurrent networks to handle sequences of arbitrary length, even if this length was not observed in the training data. Feedforward neural networks and convolutional neural networks may be viewed as a directed acyclic computational graph going layer per layer from the input to the output. Recurrent networks extend this by allowing cycles within the computational graph. When the state $\boldsymbol{s}$ depends on the previous state of the system and its parameters, it may in general be written as $\boldsymbol{s}^t = f(\boldsymbol{s}^{(t-1)}; \theta)$. In a process called unfolding the computational graph may be unrolled for any finite number of steps in time T by replacing $\boldsymbol{s}$ with the definition. Therefore with $t = 3$, the previous function may be unrolled as

$$\begin{aligned} \boldsymbol{s}^{(3)} &= f(\boldsymbol{s}^{(2)}; \theta) \\ &= f(f(\boldsymbol{s}^{(1)}; \theta); \theta) \end{aligned} \tag{11}$$

The above-mentioned parameter sharing can be seen, as the same θ gets processed by every call of the function f. This represents the usage of the same transition function for every time step. Applying this unfolding results in a graph without recurrence, which may be seen as a directed acyclic graph (Zhang et al., 2023). In recurrent neural networks, the state of the hidden units may therefore be written as

$$\boldsymbol{h}^t = f(\boldsymbol{h}^{t-1}, \boldsymbol{x}^t; \theta) \tag{12}$$

The forward pass through such a network consists of initializing $\boldsymbol{h}^{(0)}$, then for each $t \in \{1, ..., T\}$ the activation

$$\boldsymbol{a}^t = \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)} \tag{13}$$

gets computed, where $\boldsymbol{b}$ is the bias vector, $\boldsymbol{W}$ is the weight matrix weighting the influence of the previous hidden state and $\boldsymbol{U}$ is the weight matrix applied to the new input data at time t. Afterwards the hidden state $\boldsymbol{h}^{(t)}$ gets computed by applying the activation function to $\boldsymbol{a}^{(t)}$. The output $\boldsymbol{o}^{(t)}$ is then generated via the bias vector $\boldsymbol{c}$ and the matrix weighting the influence of the hidden state to the output, $\boldsymbol{V}$:

$$\boldsymbol{o}^{(t)} = \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)} \tag{14}$$

When the task is a classification based on past events, the network learns to treat $\boldsymbol{h}$ as a lossy summary of past events. Learning in these networks might be done via the backpropagation algorithm applied to the unfolded computational graph. This process is known as back-propagation through time. As this process is inherently sequential, parallelization is not possible, resulting in $O(T)$ runtime (Goodfellow et al., 2016).

Long-term dependencies are difficult to process for these recurrent neural networks because this information is stored within the hidden states and the information

within these will either vanish or explode, depending on the weight matrix $\boldsymbol{W}$, because as the information within gets transported through time, this matrix will be multiplied with itself multiple times.

The Long short-term memory (LSTM) model is a specialized version of recurrent neural networks aimed at mitigating these problems by using LSTM cells. The input and output of these LSTM cells are identical to those used by recurrent neural networks, however, LSTM cells have an additional system of gating units attached to them, which control how and if past information flows through time (Hochreiter and Schmidhuber, 1997). This system of gating units within an LSTM cell is displayed in Figure 7.

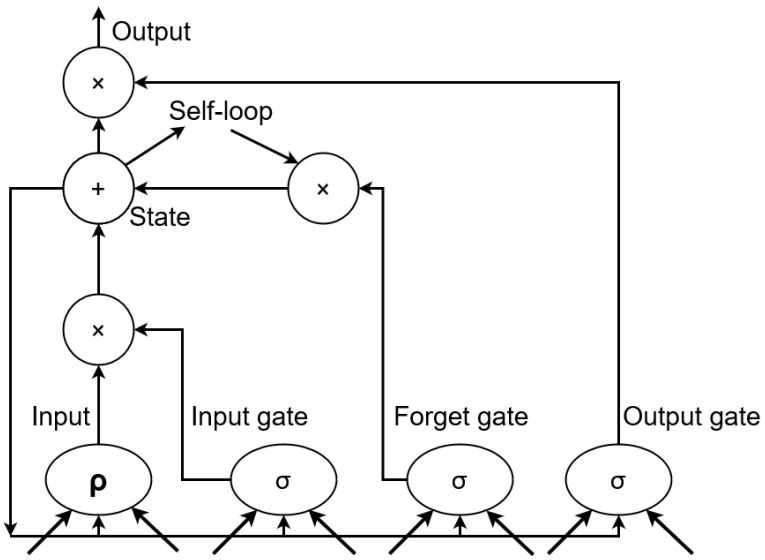

Figure 7: LSTM Cell

The central unit is called state unit $s^{(t)}$, which has a self-loop. The weight of this self-loop is determined by a forget-gate, which uses a sigmoid activation function to ensure that this weight $w \in (0, 1)$. The activation of the forget gate for one LSTM cell at time t is being computed as

$$f^{(t)} = \sigma\big(b^f + \sum_j U_j^f x_j^{(t)} + \sum_j W_j^f h_j^{(t-1)}\big) \tag{15}$$

With b^f denoting the bias of the forget gate, U^f denoting the weight applied to the input, W^f denoting the weights applied to the previous hidden states, also known as recurrent weights and $x^{(t)}$ denoting the external input at time t. Each LSTM cell additionally has an external input gate g, which determines, the weight with which the new external input is being used to update the internal state of s^t. Additionally, there is an output gate q which uses present and past information to determine the weight applied to the state before outputting. The calculation for all gates and the external input e is analog to the one for the forget gate f, however each with their

own bias $\boldsymbol{b}$ and weight matrices $\boldsymbol{U}$ and $\boldsymbol{W}$. The activation function for the gates is typically the sigmoid activation function, whereas the activation function for the input may be any generic squashing function ρ (Goodfellow et al., 2016).

The activation of the state unit $s^{(t)}$ is therefore

$$s^{(t)} = f^{(t)} s^{(t-1)} + g^{(t)} e^{(t)} \tag{16}$$

The output h^t of the LSTM cell is being controlled by the output gate q^t, resulting in

$$h^{(t)} = \tan\left(s^{(t)}\right) q^t \tag{17}$$

With this architecture, LSTMs have been successfully applied to a wide range of time-related data, proving their ability to successfully handle long-term dependencies within the data (Hammerla et al., 2016).

3 Data

The following chapter aims to explain the origin of the data, as well as the preprocessing steps applied to all the data used within this study.

3.1 Recording Device

The data for this research was generated by the active, rigid, low-back support exoskeleton Cray-X manufactured by the company German Bionic. The devices used within this study are Cray-X of the fifth generation, displayed in Figure 8. It is an powered exoskeleton, which generates forces itself and applies these forces via hardened parts on the user's body. An introduction of exoskeletons is given in Subsection 2.2.1. The intended usage of the Cray-X is relieving workers when performing lifting tasks by straightening the user, when an upwards movement is detected from a bent position. Force is being applied to the user via straps on the torso, the thighs and the lower back to bring the user from the bent position into a straight standing position (Schmidt, 2021b). The support generated by the device is modulated by manually set support parameters, determining the scale of the given support.

Figure 8: Cray-X of the fifth generation (Schmidt, 2021a)

The data recorded by the exoskeleton includes data from an IMU. Analog to other research, this data was reduced a priori to the data from the built-in accelerometer and gyroscope, i.e. the IMU was treated as a 6-axis IMU (Pesenti et al., 2023). This was done based on the assumption, that the magnetometer data would add little

additional information. This assumption was made, as the state of the art contains reports of well-performing models using only accelerometer and gyroscope data. Additionally, the angle of the thighs, as well as the torso respective to the hip is measured. These sensors are introduced in Subsection 2.1.3. The raw data contained 9 dimensions: The triaxial accelerometer, the triaxial gyroscope, the angles of the left and right leg, as well as the tilt angle of the torso. The data was available in high resolution, i.e. the angles were observed with a frequency of at least 10 Hertz, the IMU data with a frequency of at least 60 Hertz. Hertz is defined as events per second, therefore a frequency of 100 Hertz denotes that a given event happens 100 times within a second.

3.2 Definition Activities

This section entails the definitions of the seven activities distinguished within this study.

The first activity is lifting, which includes the process of a standing human lowering the torso without holding any weight, picking up a weight and placing the weight at a higher position before lowering again.

The second activity was dropping, which was defined similarly to lifting, albeit that the weight was lowered from a higher position to a lower position. Similar to lifting, the activity of dropping was defined in a way that includes the process of moving the body towards the next weight, i.e. the dropping data contained data in which the subject straightens the body in order to grab an object from a higher position.

Holding was the third activity, which included a static holding of an object in front of the user's body in a bent position. Only slight movements sideways were allowed within this activity because otherwise, the seen activity would shift to lifting or dropping.

Holding, lifting and dropping were defined as activities performed in a standing position.

Three activities concerned with modes of walking were distinguished. These include walking normal, i.e. on flat ground, as well as walking up stairs and walking down stairs. The activity walking normal includes movements of pushing and pulling a trolley or a cart while moving on a flat surface. Moving backward or in circles is included in the activity walking normal.

The last activity was resting, which summarized the activities of sitting and standing, both without the handling of weights.

The definition of these activities, determining the label given to the raw sensor data, was set a priori and not adjusted throughout the study. These activities are commonly used within the literature and can be aggregated under the term activities of daily living, as mentioned in Subsection 2.1.1. The amount and granularity of the chosen activities follows the state of the art to enable comparison. The activities

were defined as such, as it was assumed that the trained models would be able to separate the respective sensor data reliably.

These definitions deal with the whole-body movements of individuals. These activities were defined as mutually exclusive, i.e. a person could only perform one activity at a time.

3.3 Data Acquisition

3.3.1 Training Data

The training data was recorded in the testparcour of the company German Bionic, where testers perform predefined tasks. The definition of these tasks was done by experts to utilize the resulting data for the training of human activity recognition systems. The chosen tasks are similar in structure to those encountered in the logistics industry.

At the beginning of a task, the testers started a recording of the sensor data on the device so that the raw sensor data could be linked to the activity. Examples of these tasks are walking in a circle for 30 seconds or lifting multiple weights on a board. Every recording contained a singular activity for a minimum duration of 20 seconds and up to 10 minutes. The testers can manually change the scale of support provided by the exoskeleton between the recordings. During the tasks, the testers were not under direct surveillance. As the subjects were employed over long periods, some familiarisation with the tasks, the exoskeleton and the environment is expected. The testers are able to choose the order in which the tasks are being performed, therefore the training data could contain the same activity performed at different levels of fatigue.

For the training data, nine testers were included, of which seven were male. The testers were between 19 and 53 years old with an average of 32 years.

The training data results from recordings made by various devices. However, these devices are all fifth-generation Cray-X exoskeletons with identical compositions. Therefore, it is assumed that differences in the behavior of these devices are negligible.

The training data was recorded from February 2022 until October 2022.

The lifting and dropping data includes moving the weight from various heights to other varying heights. The used objects for lifting, dropping and holding range from five to 20 kg in weight. The walking normal data included walking around corners and walking backwards. For walking up or downstairs the training data contains data from a singular staircase. This staircase was not a spiral staircase, i.e. walking stairs contained only slight shifts sideways. The steps of this staircase had the same length, width and height. The resting data includes sitting and standing data, with chairs of differing height present within the dataset.

It was not possible to map the recordings in the training data to the individual tester.

3.3.2 Validation Data

Analog to the training data, the validation data was recorded in the testparcour of German Bionic. In contrast to the training data, the validation data was explicitly recorded for this study.

The volunteers used for the validation data were recruited from the employees of the company German Bionic.

The volunteers used for the validation data were informed of the goal of the recordings and had several minutes of preparation to find the preferred support settings of the exoskeleton. The subjects were instructed to perform the tasks in a manner that is natural to them. Afterward, each volunteer performed an individual protocol of tasks. This protocol ensured, that the resulting activities were observed for similar times. The protocols for the subjects included the same tasks, albeit in a different order to include recordings of the same activity at different levels of exhaustion within the dataset. During the protocol, the subjects were under observation by the author. Based on the length of the protocol, the author gave instructions regarding the next tasks whenever the volunteer was standing or sitting, resulting in the observation of more resting data, see Figure 10. The subjects were familiar with the exoskeleton, but none had a history of continuous usage like the subjects observed in the training data. Performing the individual protocol took between 15 and 25 minutes, depending on the resting time needed for recovery and understanding the following tasks. Between each task a break was done, in which the subject rested. During these 10 seconds of resting the subject was able to change the level of support given by the exoskeleton depending on current and future need. An example of a used protocol is given in Appendix A.3.

The validation data consists of recordings made by $n = 4$ subjects between 20 and 26 years of age. Of these four subjects, three were male. No recording of the author is included in the validation data. None of the subjects in the validation data appear in the training or test data.

The validation data was recorded on different Cray-X exoskeletons of the same generation. As a result, these exoskeletons share a common internal structure and identical components. Differences due to the use of varying devices are therefore assumed to be negligible.

The validation data was recorded on three days throughout October 2022.

Analog to the training data the validation data includes the handling of objects of differing weights. These objects were moved to different heights to include a wider variety of modes of performing these activities. Various modes of walking normal are included in the validation data, e.g. pushing and pulling a cart. The staircase observed within the training data is identical to the one within the validation data. In general the weights, stairs and chairs utilized for the recording of the validation data are identical to the ones observed within the training data, see Subsection 3.3.1.

3.3.3 Test Data

The test data was not recorded at the location observed in the training and test data. This was done to prevent that information about the unseen test data are present in the training data, i.e. to prevent that information about the shared environment aid in the correct classification of test data. This was done to reliably estimate the performance of the models in a real-world scenario, where none of the users have been observed within the training data and the environment differs from the environment of the training data. Changing the environment was done as it is assumed, that the environment influences the specific way an activity is performed, e.g. the height and width of a stair tread influence the specific mode of walking stairs. Similar to the validation data the test data was recorded for this study.

The volunteers used for the test data were recruited from the author's acquaintances.

Analog to the validation data the volunteers observed within the test data were informed of the intended usage of the data and the purpose of the study. As none of the subjects observed within the training data had experience with handling active exoskeletons, five to 10 minutes were given for acclimatization. An explanation of the behavior and the steps needed to change the support of the exoskeleton were given. The subjects were instructed to perform the tasks in a way that feels natural and comfortable to them. The individual protocols contained the same tasks with different ordering. Similar to the validation data the next task was explained after the previous one, while the subject was resting. During this resting phases the subjects were allowed to change the support of the exoskeleton according to current and future need. Performing the individual protocol took a similar amount of time as performing the protocols of the validation data, i.e. 15 to 25 minutes.

The test set was collected to evaluate the performance of various models on unseen data.

The test data contains $n = 10$ recordings from subjects between 23 and 53 years of age. The median age was 26 years. Seven of the 10 subjects were male. The data was recorded in five different locations, none of them was the location seen in the training and validation data. No data of the author was included in the test data.

The test data was recorded on one Cray-X of the fifth generation.

The test data was recorded over four days in January 2023.

For each environment, different objects were used for lifting, dropping and holding. The weight of these objects was within the boundaries observed within the training data. Several modes of walking normal are included in the test data, such as moving a cart. The test data contains recordings from two spiral staircases. The resting data contains a mixture of sitting and standing, both equally observed for each subject, albeit with different chairs at varying locations.

As one research question is concerned with estimating the performance of the trained models in a real-world scenario, avoiding leaking information from the test set into the training set became a priority. Based upon this the allocation of data to the

training, validation, or test set was not randomly chosen. This was done to prevent observing the environments or subjects of the test data within the training data. The design of this study did not allow for the usage of K-fold cross-validation, as explained in Subsection 2.1.2, for several reasons: The first being that it was not possible within the scope of this study to map the training data to a tester which made the respective recording. Based on this the training data had to be used as a joined dataset. The second reason is that the baseline method, which was used as a comparison, was trained on the training data. Comparing the performance of models trained on different datasets cannot be used to answer questions regarding the better-suited architecture. Based on these reasons the test data was carefully constructed to contain varied data. This was done to get a stable estimation of the performance of the models without relying on k-fold cross validation.

It is important to consider that the test data was not created by a random sample of a greater population. Therefore results based on the usage of this set are not representative. Instead, they are to be understood as first attempts to answer the research questions. Generating a representative sample would require the definition of a population, from which this sample is drawn via a specified sampling procedure.

3.4 Preprocessing

This section aims at describing the preprocessing applied to all the datasets described in Section 3.3.

3.4.1 Data Cleaning

Data Cleaning refers to removing or correcting incorrect or incomplete data within a dataset.

Errors induced by miscalibrated sensors were avoided, as the sensors used within the exoskeletons were correctly calibrated before the construction of the respective exoskeleton.

No filtering technique for smoothing the sensor values has been applied.

Before the files were labeled, outliers in the sensor readings were searched. This consisted of checking if the seen sensor values lie within boundaries of valid sensor values. When the sensor values of a file lay outside these boundaries, a human could inspect and remove the respective file, if necessary. This inspection was done by human experts at German Bionic.

3.4.2 Labeling

The files were labeled manually, meaning the activity label was attached to the file containing the respective raw sensor data only after being checked by a human. When doing this inspection the data was cut if needed, keeping only data from the

respective activity. Faulty and erroneous recordings were removed. During labeling the human inspector visually checked if the data is appropriate and as expected for the respective activity class. This labeling was done at German Bionic by experts familiar with sensor readings and the activities. Ambiguous, faulty, or erroneous data was removed within this process and was not used for the respective data sets. Based on this manual inspection the resulting data quality is assumed to be adequate, as every file was inspected by a human expert and confirmed, that the data is valid.

The training data was labeled by several experts at German Bionic, while the validation and test data were labeled by the author.

The following displays the distribution of the activities seen in the labeled data for the training data in Figure 9, validation data in Figure 10 and test data in Figure 11.

The amount of available training data differed widely depending on the observed activity: The most-observed activity, walking normal was observed over 750 times more often than the least observed activities, walking up and walking down stairs. Figure 9 displays the distribution of the raw, labeled training data in minutes per activity. The overall amount of training data is larger than typically observed within the literature, albeit being highly imbalanced (Grzeszick et al., 2017).

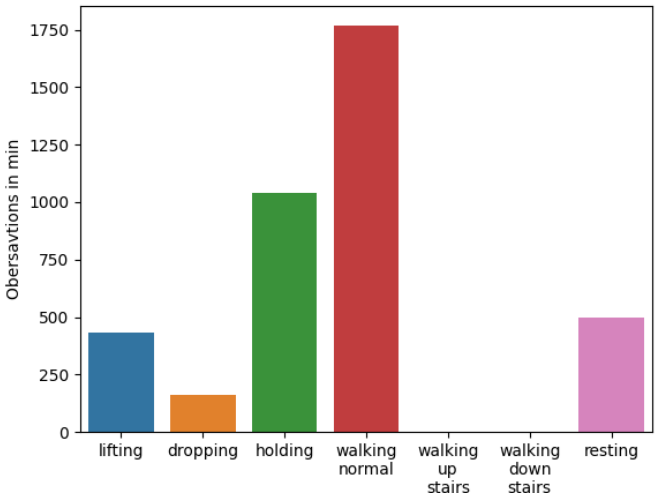

Figure 9: Amount of raw training data in minutes per activity

Figure 10 displays the distribution of labeled validation data in minutes per activity for the validation data. The activity resting was observed more frequently due to reasons explained in Subsection 3.3.2.

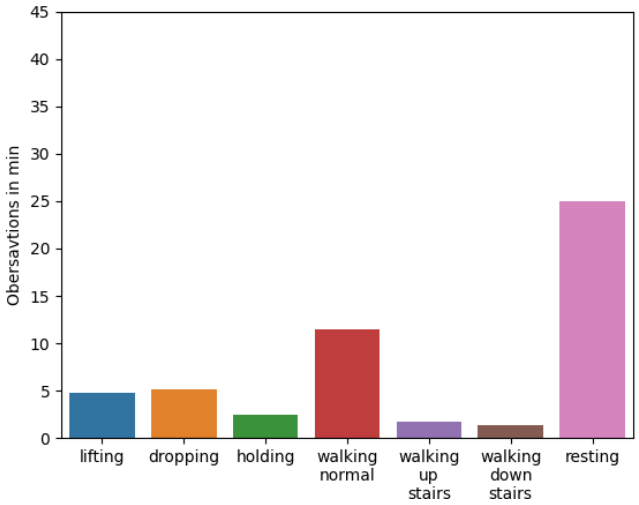

Figure 10: Amount of raw validation data in minutes per activity

Figure 11 shows the distribution of the resulting labeled test data in minutes. Similar to the validation data the test data contains more walking normal data than most other activities, as this activity has several modes in which it was performed, see Section 3.2.

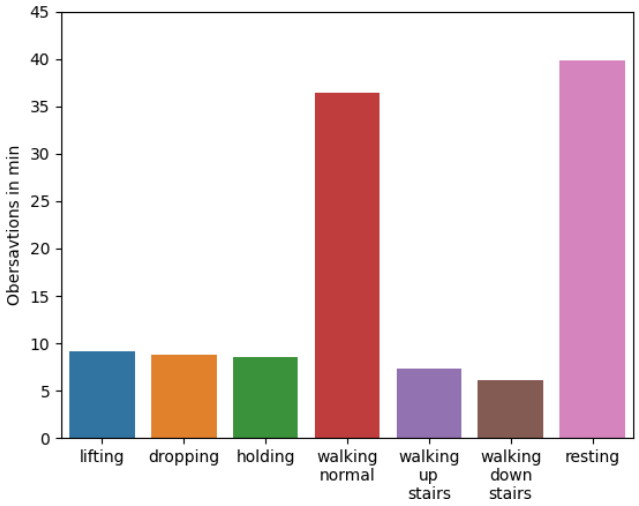

Figure 11: Amount of raw test data in minutes per activity

3.4.3 Resampling

After labeling the data, some general-purpose preprocessing was applied. This includes resampling the data, i.e. gathering various sensor readings into a singular data point. This was being done to ensure that the aggregated sensor readings are

evenly spaced and a fixed period contains the same amount of readings. Resampling was done guided by the literature and resulted in a resampling rate of 100 Hertz, meaning that each resampled datapoint contained the information of 10ms (Pesenti et al., 2023). To fill gaps within the sensor readings various possibilities exist. It was chosen to use forward filling and averaging of the resulting sensor readings to produce the resampled data points. This decision was made by experts, which used these methods and parameters to produce the data for the training of baseline method.

An illustration of the resampling process used for this study is given in Appendix A.2.

3.4.4 Creation of Windows

This collection of resampled data points was used to generate windows. The approach used for the generation of windows is the time-driven windows segmentation approach, see Subsection 2.1.4. It divides the stream of sensor readings into windows of a fixed length. The chosen window size was one second. This window size was inspired by the literature, where similar lengths are common (Grzeszick et al., 2017). As a result of the prior resampling, the amount of data points within each window is identical. Each window contains the data for all $m = 9$ axis of measurement, see Section 3.1, for $n = 100$ observations, making the resulting data matrix a $9, 100$ matrix. To generate additional data, the windows were created with a 50% overlap, i.e. a new window was created beginning at the middle of the previous one (Valarezo et al., 2017). Doing so almost doubled the available data. This procedure may be seen as a data augmentation technique, see Subsection 2.3.1, to generate additional examples from the existing data, without the risk of generating adversarial examples.

3.4.5 Balancing

Balancing a dataset refers to adjusting the dataset, so that the labels in the resulting dataset appear with the same frequency. This is being done, as using unbalanced datasets for the training of models may affect the classification behaviour in an undesired manner. An introduction of balancing is given at the end of Subsection 2.1.4.

As seen in the Figures 9, 10 and 11 the datasets are imbalaced. Therefore balancing was applied to all datasets.

For this study oversampling was used, i.e. before the dataset was inserted into a model for training, validation, or testing purposes, the data was balanced by adding additional samples of the minority classes until the dataset is balanced. These additional samples were inserted unaltered. The same oversampling algorithm was used for all datasets and models.

The windows introduced by this oversampling technique contain the same information as the windows already present in the data set, so the additional samples might be interpreted as placing a higher weight or importance on the original samples (Fernandez et al., 2018). The data was balanced by oversampling because experimental results show that this balancing method results in better model performance when compared to undersampling, which can be explained by not discarding information of the majority classes (Mohammed et al., 2020).

This balancing was applied after all model-specific processing was applied to the windowed data. This model-specific preprocessing is explained at the same location as the respective model in Chapters 4 and 5.

4 Baseline Method

This section aims at explaining the structure of the hierarchically classifying Support Vector Machine model. This model is used as a reference for the performance of the neural networks explained in Chapter 5.

4.1 Motivation

This model, from now on referred to as the baseline method, was built as a reference for the results achieved by the various neural networks. The model represents the conventional approach to HAR, before the advent of deep learning in HAR and was constructed by using the domain knowledge of experts. These experts are experienced with the construction of human activity recognition systems based on active exoskeleton sensor data. As a consequence, the baseline method is seen as representative of the conventional approach. This conventional approach is laid out in Subsection 2.1.4 and illustrated in Figure 1.

4.2 Data Processing

The processing steps for the data for this method closely follow the process described in Subsection 2.1.4, i.e. preprocessing, segmentation, feature extraction, dimensionality reduction and balancing.

The same processing steps were applied to the training, validation and test data in order to use them with this model. The origin of these datasets is laid out in Section 3.3. The general preprocessing as laid out in Section 3.4 was applied, i.e. the raw data was cleaned, labeled, resampled and cut into windows of a fixed length. The preprocessing and segmentation of the data were therefore identical to all other models used within this study.

4.2.1 Feature Extraction

After generating the resampled windows from the data, as explained in Chapter 3, the data in the format of windows was used to generate a set of features. This set contained features from the time domain, as well as the frequency domain approach, as explained in Subsection 2.1.4. As a result of this process the features, which were calculated on the sensor values within the window were attached to the respective window, and the raw sensor values were discarded. This was being done, as it was assumed, that all relevant information within the sensor readings of the window is present within the generated features.

4.2.2 Dimensionality Reduction

This set of features was then reduced to a set of features to be used for the classification. The set of features to keep resulted from a process of combining domain knowledge and filter and wrapper feature selection techniques. These techniques are introduced in Subsection 2.1.4. The features constructed and kept for the classification of samples cannot be made publicly available, as these are company secrets owned by German Bionic.

4.2.3 Scaling

As the used classification models, Support Vector Machines, are sensitive regarding the scale of the input data, the features needed to be normalized. As the Baseline Method consists of three Support Vector Machines, i.e. the top-level, the lifting and the walking classifier, each of the classifiers had another set of features as input.

The input features for each classifier were standardized either by z-normalization or maximum absolute normalization. The standardization type was chosen based on domain knowledge. The parameters used for normalization were computed on the training data, but used for the validation and test data as well.

Z-normalization refers to a rescaling of the data $\boldsymbol{x}$ by

$$\boldsymbol{z} = \frac{\boldsymbol{x} - \mu}{\sigma} \tag{18}$$

The resulting distribution has a mean of 0 and a standard deviation of 1 (Gu et al., 2021). When the original distribution was a normal distribution, the resulting distribution is known as a standard normal distribution. When maximum absolute normalization was chosen, the data $\boldsymbol{x}$ was rescaled via

$$m = \frac{\boldsymbol{x}}{max(|\boldsymbol{x}|)} \tag{19}$$

The resulting values are within $[-1, 1]$.

4.2.4 Balancing

Afterward, this set of files, each file containing only for the classification necessary features, was balanced via oversampling, as explained in Subsection 3.4.5. The data was balanced in a way that ensures that classes separated by each classifier are balanced, i.e. appear equally often.

The windows were saved as individual files. This is not necessary when training a Support Vector Machine, but it was useful for training the neural networks, as these are trained on minibatches. Saving the windows as individual files allowed loading into Random-Access memory (RAM) only the windows needed for the current minibatch. As the neural network introduced in Section 5.3 is trained on the same data as the baseline method, the data format needed to be appropriate to train a neural network.

4.3 Classification

The final classification resulted from a hierarchical classification process. The first model, also known as the top-level classifier, separated the three general classes lifting, walking and resting. When lifting was classified the so-called lifting classifier was called to separate the three activities lifting, dropping and holding. When a sample was classified as walking by the top-level classifier, the walking classifier was called to classify this data as walking normal, walking up stairs or walking down stairs. The definition of these activities is given in Section 3.2. For this classification each classifier had its own set of features available. Figure 12 illustrates this classification process.

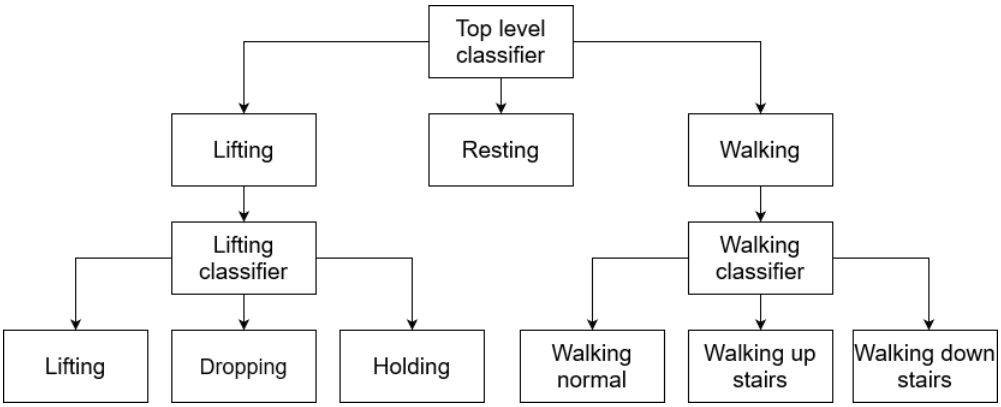

Figure 12: Flowchart of the Classification of the Baseline Method

The classification based on these normalized features was done by a machine learning model called a Support Vector Machine (SVM). This is a typical model for HAR, which works by finding the hyperplane, which separates the respective classes, represented by vectors of the examples, the best, i.e. with the widest margin (Yin et al., 2021). As this is only possible when the data is linearly separable, the algorithm may apply the kernel trick, which is based on the fact, that in a space of high-enough dimensionality, any set of vectors is linearly separable. Using this kernel trick enables the algorithm to find well-mannered functions, which separate the sets in a high-dimensional space, as well as enabling the retransformation to the original, lower-dimensional space (Bishop, 2006).

Each classifier utilized a Support Vector Machine as the classification algorithm, however with individual sets of features and label classes as output. Depending on the classifier the definition of the respective labels changed. The lifting class of the top-level classifier contains the activities lifting, dropping and holding and the walking class of this classifier contained walking normal, as well as walking stairs. This definition of the classes for the top-level classifier was necessary to enforce, that the top-level classifier calls the correct classifier when classifying activities.

The classifiers were trained exclusively on the training data. For the optimization of hyperparameters of the classifiers a randomly chosen subset of the training data was used as validation data. The validation data as laid out in Subsection 3.3.2 was

not used for this purpose. This is because the baseline method was trained before the beginning of the study and treated as fixed throughout this study.

The training of the baseline method with the respective classifiers was done beforehand by human experts at the German Bionic, see Section 3.1. The structure of the baseline method as a hierarchically classifying collection of Support Vector Machines was decided by these experts.

5 Method

5.1 Motivation

As seen in Subsection 2.2.3 there is little research to date regarding human activity based on active exoskeleton data. As the experience within the field is limited to singular explorations, which still leave important research questions unanswered, this thesis aims at providing the first general insights.

Regarding the ability to generalize the present literature does not give insights into the performance of the models when applied to new persons or within new environments. The number of subjects included in previous studies was typically small, making it plausible to assume that the implemented models might have overfitted on the observed subjects. When the same subjects or the same environment is used for training and testing, the assumption that the test set consists of unseen data is violated, which might help overfitted models to perform well on the test data. The phenomenon of overfitting is introduced in Subsection 2.1.2. As the model´s ability to generalize is crucial for application in practice, estimating the robustness of the models is an important issue, which is currently disregarded by the literature. This thesis tries to estimate the performance of trained models in an unbiased manner by a careful creation of the test set, see Subsection 3.3.3.

As large-scale studies are still missing within this area of research, it is unclear which information is the most important for correct classification. To find evidence this study compares various structures of neural networks, which handle information differently. The assumption is made, that the best model was able to perform in that manner because it had the most useful information available. Based on this assumption it is inferred, which information is the most important for the success of the various network types.

For practitioners, it is important to know which effort is necessary to achieve acceptable performance with the models. As the current literature often conceals the number of hyperparameter constellations which were tried before reaching the final model, this thesis aims at communicating this effort (Hammerla et al., 2016). Besides communicating experience to foster realistic expectations, this is being done to enhance reproducibility.

5.2 Hyperparameter Optimization Schedule

To enable a comparison between the different models, all architectures trained throughout Chapter 5 have undergone a process of hyperparameter optimization, which is as similar as possible. This includes that all the models were trained for 10 epochs. This number of epochs was chosen based on the comparatively large amount of available training data even before balancing, see Figure 9. The hyperparameters were optimized one after the other with the order of optimization based on the results of large-scale experiments by Hammerla et al. (2016). In addition, the

importance of the hyperparameter influenced the number of experiments performed to find the optimal hyperparameter value. More important hyperparameters were investigated more thoroughly. The number of experiments conducted was limited by the resources available to this study.

The initial model resulted from a preliminary exploration, which ended as soon as a model with over 70% accuracy on the validation data was detected. For this exploration 5% of the available training data were used, to find a hyperparameter constellation that enables learning from the available data. The hyperparameters used in the preliminary exploration were based on domain knowledge of the author, as well as estimations regarding the complexity of the task to be solved by the respective model. The hyperparameters of this model were used as a starting point for the hyperparameter optimization. All of the experiments conducted to find the optimal hyperparameter values consisted of training for 10 epochs while tracking loss and accuracy on the training and validation data.

The first hyperparameter to be optimized was the learning rate μ of the model, for which six experiments were done for each network. During this search, the learning rate μ was varied across orders of magnitude in four experiments before the last two experiments were concerned with finding the best learning rate within the range of tried learning rates.

Using the best learning rate the architecture of the network, i.e. the amount, composition and width of the hidden layers, was optimized in four experiments. Additional hidden layers were added or the number of hidden layers was reduced while increasing the width of the remaining hidden layers. Experiments were performed, in which the depth of the network was kept constant, while the width of the hidden layers was increased.

Afterward, the effects of regularization via dropout were explored by three experiments, using the dropout rates of 0.5, 0.25 and 0.1.

Afterward, the minibatch size was optimized in two additional experiments, where deviations from the default size of 64 were tried. These minibatch sizes were 256 and 512. It was chosen to explore the effects of higher minibatch sizes to test the effects of a less noisy approximation of the information within the dataset. The process of hyperparameter optimization is illustrated in Figure 13.

The final model was the model with the highest accuracy on the validation data from all the models produced throughout this hyperparameter optimization process. The validation data was used only for finding the correct hyperparameters of the model, not for optimizing the parameters of the final model.

The exact number of experiments for each hyperparameter was chosen based on the relevance of the respective hyperparameter. The best resulting model was used as the final model for this architecture and used for the classification of the test data. The optimal hyperparameter values were found by choosing the model, which resulted in the highest accuracy on the validation data during training. For this an naive early stopping algorithm was used, which saved the model with the current parameter values, when the performance of the model on the entire validation dataset was

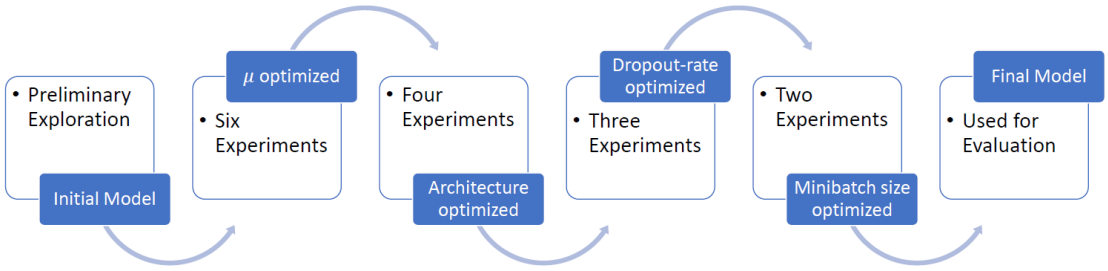

Figure 13: Process of hyperparameter optimization

better than all previously measured performances. This algorithm enabled training each model on the same amount of epochs, while not letting overfitting caused by training for too long influence the final model performance.

To prevent data leakage from the test data into the training process, the test data was used after the training of all models was finished. The classification of the test data happened five times throughout this study, i.e once for every model type mentioned throughout Chapters 4 and 5.

Due to the scope of this study, the effects of all hyperparameters could not be evaluated. It was chosen to limit the experiments to the set of hyperparameters that was assumed to have the biggest influence on the performance of the model. Some hyperparameter values were used for all networks, such as the loss function. For all networks, the Cross-Entropy Loss was used, which is explained in Subsection 2.3.1. As an optimization algorithm, adam was chosen. When not stated otherwise, the activation function used for the hidden units was the ReLU. The window size and resampling rate for all networks and the baseline method were constant, as stated in Section 3.4. Batch normalization was applied after every fully-connected hidden feedforward layer. These hyperparameters were chosen based on experience within the literature (Hammerla et al., 2016).

The training was done in Python, version 3.8.10, and the training of the neural networks was done via the PyTorch framework, more specifically the CPU-variant of version 1.13.1. When not stated otherwise, the default values for the respective parameters and functions of this library were used (Paszke et al., 2019).

The baseline method was processed by using the SVC class of the Python package Scikit learn (Pedregosa et al., 2011). Version 1.1.0 of this package was used.

To improve reproducibility and enable further research, the code used in this study is freely available under the MIT license, see appendix A.1.

The training and validation were performed on an Intel(R) Core(TM) i5-1135G7 CPU of the 11th generation with 2.40GHz.

5.3 Feedforward Network applied to features

The first model trained explicitly for this study was built to answer the question: Does the baseline method use the available information in an optimal way? Answering this is essential for the comparison of later models with the baseline method. To test this a neural network was built, which is in structure and format of training data exactly equal to the baseline method. It classifies hierarchically based on a normalized set of features, with first classifying on an upper level if the seen features belong to the classes lifting, walking, or resting. Depending on the first classification the lifting or walking classifiers might get called, analog to the baseline method, explained in Chapter 4. This classification process is depicted in Figure 12. In the following, this model is referred to as the ShallowFFNet model.

The preprocessing of the data for this model strictly follows the preprocessing explained in Section 4.2, except for exchanging the final classifier to a fully-connected feedforward neural network. Each classifier had an individual set of features as input. These sets of features were identical to the sets of features utilized by the baseline method, i.e. the top-level classifier of the ShallowFFNet model had the same set of features as input as the top-level classifier of the baseline method. For the normalization of the features, the same parameters as for the baseline method were used. These parameters were calculated on the training data and used for the validation and test data as well, see Subsection 4.2.3. The datasets, consisting of standardized sets of features, were balanced via oversampling as explained in Subsection 3.4.5.

Based on the hierarchical structure of this classifier, three networks were trained: The top-level classifier, as well as a lifting classifier and a walking classifier.

Training the three classifiers in sequential order with 15 experiments performed for each model to find the optimal hyperparameter values, with each experiment consisting of training over 10 epochs, resulted in the final hyperparameter values. Table 1 displays the final hyperparameter values for each classifier of the ShallowFFNet model.

Table 1: Hyperparameters for Final ShallowFFNet Model

Classifier	Learning rate	Nr. HL	Width of HL	Dropout rate	Minibatch size
Top	0.000003	1	8	0.0	64
Lifting	0.000003	1	8	0.1	64
Walking	0.00003	1	8	0.0	64

After the hyperparameter optimization process, the final top-level model contains 307 parameters, the lifting classifier 91 parameters and the walking classifier 99 parameters to be optimized during training. This amount of parameters refers to the number of weights and biases within the network. An explanation of these terms is given in Subsection 2.3.1.

5.4 Feedforward Network applied to raw data

The next trained model was built to answer the question if enabling the model to learn relevant features as a part of the training process enables better performance when compared to using handcrafted features as input for the models. For this purpose a model was created, which got the training data in a raw format as input. In the first step, the performance of a deep fully-connected feedforward network is evaluated, before more specialized network variants are trained in the following two subsections. As this model learns useful representations of the data within multiple hidden feedforward layers, this network is referred to as the DeepFFNet model.

The processing of the datasets was as follows: The general preprocessing as explained in Section 3.4 was applied. The labeled and resampled data was cut into windows. This data was z-normalized, as explained in Subsection 4.2.3. To avoid differences in the resulting performance based on different standardization for the various models, the data for the neural networks had to be standardized as well. Similar to the baseline method the parameters used for the standardization of the input data were computed based on the training data and then used for the training, validation and test data. In contrast to the previously mentioned models, the raw data was standardized. This was done as this model receives the raw data as input, whereas the ShallowFFNet model and the baseline method receive features as input. Standardization therefore had to be applied to the raw data. This was done for each axis separately. As a result, the $n \times m$ matrix for one window with m denoting the number of measured axis contained $m = 9$ vectors with an expected mean of 0 and an expected variance of 1. $n = 100$ is the number of resampled datapoints within a window, see Section 3.4.

After standardizing the data was flattened, i.e. the data for one window was brought from the shape $n \times m$ to a vector of length $n \cdot m$. With $m = 9$ and $n = 100$, this yields a length of the resulting vector of 900. Based on this process the input layer of the DeepFFNet model had a width of 900 units.

The resulting data was balanced via oversampling so that each of the seven activities has the same number of available data points. Analog to the model trained in Section 5.3, the chosen oversampling algorithm drew samples from the existing samples of the minority class and added them unchanged to the dataset.

The training regime for this model was as follows:

The preliminary exploration resulted in an architecture with three hidden fully-connected layers with 70, 40 and 20 ReLU units respectively after trying three model structures.

After 15 experiments, which followed strictly the training protocol explained in Section 5.2, the hyperparameters for the final DeepFFNet model were found.

Table 2 displays the final learning rate, number and width of hidden layers (HL), dropout rate and minibatch size. These hyperparameters are introduced in Subsection 2.3.1.

Table 2: Hyperparameters for Final DeepFFNet Model

Model	Learning rate	Nr. HL	Width of HL	Dropout rate	Minibatch size
DeepFFNet	0.00001	3	70,40,20	0.1	64

The final DeepFFNet model contained 66877 weights and biases which were optimized during the training process.

5.5 Convolutional Neural Network applied to raw data

This model was constructed and trained to test, whether allowing the model to take advantage of the ordering of the data along the time axis improves performance. This ordering is leveraged by models utilizing the convolution operation. An introduction of these models is given in Subsection 2.3.2. Based on this convolution operation the final network of this architecture is referred to as the ConvNet model. It is assumed that utilizing this ordering of the data is possible when the neural network receives the raw data, therefore the data used for this model is not brought into the format of features. Using several hidden layers further enables learning useful representations of the data.

The data used for the training, validation and testing of the model was preprocessed as laid out in Subsection 3.4. Analog to the DeepFFNet model explained in Section 5.4, the resampled and windowed datapoints were z-normalized among each of the $m = 9$ axes. The normalization parameters from the DeepFFNet model were used. These were calculated on the training data and used additionally on the validation and test data. Normalization was done, as the input for the baseline method was standardized. When evaluating the effects of using different architectures for solving a given task it is important to keep external factors constant. For this reason the input for all neural networks was standardized.

Analog to the other models, the data was balanced by oversampling, so that each activity has the same frequency within the resulting dataset. After balancing the total amount of occurrences depends on the number of windows of the activity with the previously highest number of windows. Analog to all the other models throughout this study, the oversampling was performed by repeatedly adding samples from the minority class to the dataset, until the dataset was balanced.

As the convolution operation leverages the spatial structure of the data, the shape of the data was not converted into a vector. The matrix for one window had a shape of $m \times n$. $n = 100$ denotes the number of resampled datapoints within one window and $m = 9$ denotes the amount of measured axis

Analog to the process in Section 5.4, a preliminary exploration was performed, to find an appropriate layer structure before optimizing the hyperparameters. Five models were evaluated before a model with an overall accuracy of over 70% on the validation data was found. The first layer of this model was a convolution layer, which applied 30 receptive fields with a length of 15 and a stride of five to the data. The convolution layer performed 1D convolution along the time axis, as suggested by the literature, as well as experience within the preliminary exploration, where 1D convolution outperformed 2D convolution (Murad and Pyun, 2017).

Afterward, these feature maps were aggregated via max pooling, as this is a widely used pooling technique (Wang et al., 2019a). This pooling is done with a stride of two and a length of the area to pool of two. After the pooling, the data was flattened, before being inserted in a fully connected ReLU layer with 50 units, which was followed by another ReLU layer with 30 units.

The hyperparameter optimization was done based on the architecture of the preliminary exploration, as well as experience from the literature. Using the best learning rate from six experiments, four different layer structures were explored. The effects of adding another convolution and pooling layer were tested within this optimization step. The best network architecture was found to be a model utilizing a convolution layer with 30 receptive fields, each having a length of five, applied with a stride of five. Afterward, max-pooling was performed with a length and stride of two before the resulting condensed feature maps were flattened. this vector was processed by two feedforward layers, using 50 and 30 ReLU units respectively, before the results were passed into the output layer. Table 3 provides an overview of the final hyperparameters of the Convnet Model. This table includes the learning rate, the number and width of the hidden layers (HL), as well as the dropout rate and minibatch size. These hyperparameters are introduced in Subsection 2.3.1.

Table 3: Hyperparameters for Final ConvNet Model

Model	Learning rate	Nr. HL	Width of HL	Dropout rate	Minibatch size
ConvNet	0.00001	5	C,P,300,50,30	0.1	64

In the Table 3 the width of the hidden layers is laid out only for the last three hidden layers. The Convolution Layer (C) and the pooling layer (P) do not have a defined width. The flattening of the pooled feature maps resulted in 300 units as input for the two remaining hidden fully-connected feedforward layers.

The final model was trained based on the optimal hyperparameters found by this process of hyperparameter optimization and had 18177 parameters to be optimized throughout the training process. These parameters include all the weights and biases of the ConvNet model.

5.6 Long Short-term memory Network applied to raw data

Additional models were trained, which specialized in handling time-related data and long-term dependencies. These were trained to test the hypothesis, that handling this time-related structure of the data enables better performance. As these recurrent models make use of a unit called a long short-term memory (LSTM) cell, the final network utilizing this architecture is referred to as the LSTMNet model.

The preprocessing of the datasets for this model follows exactly the one described in Section 5.5, where the processing of the data for convolutional neural networks is laid out. As convolutional and recurrent neural networks leverage the information inherent in the ordering of the sensor readings, both networks receive the data in the same format.

The networks trained within this study make use of the standard version of the LSTM cell, which does not contain any peephole connections. This cell and the general architecture of recurrent and LSTM networks are introduced in Subsection 2.3.3.

The preliminary exploration resulted in a network utilizing a single hidden layer of 150 LSTM cells. Before arriving at this network four combinations of learning rates and amount and width of LSTM layers were tried.

The process of hyperparameter optimization deviates from the description given in Section 5.2. After six experiments regarding the optimal learning rate the architecture was explored in four experiments. Throughout the optimization of the network architecture, neither adding another layer of LSTM cells nor adding additional feed-forward layers improved performance. As a result, the effects of dropout could not be researched, as the final model consisted only of two layers, the input or LSTM layer, as well as the output layer. To enable comparison with the other network architectures by keeping the amount of performed experiments equal, instead of exploring the effects of different dropout rates, three additional experiments regarding the optimal architecture were performed. Analog to the other networks the final two experiments explored the effects of varying minibatch sizes. The number of experiments was 15, analog to the other neural networks.

Table 4 contains the hyperparameter values for the final LSTMNet model after the hyperparameter optimization process was finished. The hyperparameters within this table are explained in Subsection 2.3.1.

Table 4: Hyperparameters for Final LSTMNet Model

Model	Learning rate	Nr. HL	Width of HL	Dropout rate	Minibatch size
LSTMNet	0.00004	1 LSTM	200	0.0	64

The final LSTMNet model contains 170207 parameters which were optimized during the training. This number is the amount of weights and biases within the network.

6 Evaluation

6.1 Setup

6.1.1 Procedure

All the results displayed in Section 6.2 refer to the final models, i.e. the models with the best hyperparameters and parameters found throughout the hyperparameter process. This process is explained in the Chapter 5. There the model-specific processing of the datasets is explained in the respective subsections. The origin of all datasets, as well as the preprocessing applied to all datasets is explained in Chapter 3.

For the hierarchically classifying models, i.e. the baseline method and the ShallowFFNet model, the final confusion matrix was created in the following manner: The top-level classifier separated the classes lifting and walking from the activity resting. The classes lifting and walking contain each three activities. This induces the problem, that when a sample is misclassified by the top-level classifier, it is ambiguous which concrete activity got misclassified as which concrete other activity. To allocate these misclassifications to a specific field within the joined confusion matrix, it was assumed that these errors occur independently. As an example: When 9% of all samples are samples of the class lifting, that got misclassified as belonging to the class walking, each of the fields within the three by three submatrix of the confusion matrix displaying the mislabeling of these classes, received an entry of 1% of all samples. By introducing this assumption a confusion matrix with all the $k = 7$ activities as ticks for the x- and y axes could be generated to enable comparison with the confusion matrices of the other models. This procedure may have inserted slight errors in the activity-specific calculation of precision and recall but does not have any effect on scores calculated based on the entire confusion matrix.

6.1.2 Scores

There are several scores available to measure the performance of a machine learning model. The following aims at explaining all the scores used throughout Section 6.2 for the comparison of the models described in Chapters 4 and 5.

The results of classification are typically aggregated in a matrix, called the confusion matrix. The steps along the x-axis typically denote the various classes classified by the model, whereas the steps along the y-axis denote the ground truth, i.e. the true class of an example. The k classes are ordered along the axis in such a way that a correct classification is denoted by the addition of one to the main diagonal at the correct position. Especially when this $k \times k$ matrix is color-coded to mark fields with high numbers, confusion matrices pose an intuitive way of visually understanding the classification behavior of the model, given k is reasonably small. This visualization enables the practitioner to detect classes often mislabeled by the classifier, as these

are visible as fields outside the main diagonal with a relatively large number in them (Grandini et al., 2020). Figure 14 illustrates a confusion matrix.

Figure 14: Example of a Confusion Matrix

Accuracy is the most popular score for the evaluation of a classification model. It describes the proportion of correct predictions on all predictions. As a proportion, it may take values from 0 to 1 with higher values denoting a better model. As a score, it has come under some critique, especially when the data is highly unbalanced. In this case, a classifier that always predicts the majority class might achieve high accuracy, even though not learning any patterns within the given data. For a two-class classification problem, the accuracy is calculated by the formula given in equation 20.

$$accuracy = \frac{tp + tn}{tp + tn + fp + fn} \tag{20}$$

In this equation for binary classification, tp denotes the number of true positives, i.e. the number of samples for which the classifier predicted class 1 and the ground truth is class 1. tn denotes the number of correct predictions of class 0, i.e. the true negatives. fp denotes the number of samples that have the true label 0 attached to them, whereas the model classified them as class 1, meaning the model falsely classified them as positive. fn is the number of labels misclassified as class 0 (Gu et al., 2021). The multi-class extension of this calculation is straightforward when using the concept of the confusion matrix: Accuracy is the proportion of the correctly classified samples, i.e. the sum of the main diagonal, on all samples, i.e. the sum of all fields within the confusion matrix.

This score was chosen due to its widespread popularity and to enable comparison with other models within the literature. Furthermore, the data processed by the models for training, validation and testing was balanced, therefore some of the critique directed at this score does not apply to this study. The used balancing technique is described in Subsection 3.4.5.

The precision of a classifier refers to the proportion of correctly predicted positive cases, tp among the total number of positive predicted cases, i.e. $tp + fp$. As a proportion, it is defined within $[0, 1]$. A classifier with high precision is reliable in the sense, that if it classifies an example as positive, it has a high probability of being positive as the ground truth. When inspecting a confusion matrix for a multi-class classification problem, the precision may be calculated by dividing the number of true positives of this class, tp, by the amount of positive predicted cases, i.e. the sum on this column. Equation 21 contains the formula for the calculation of the precision of a two-class classifier.

$$precision = \frac{tp}{tp + fp} \tag{21}$$

However using the precision alone is not suited for the evaluation of a classifier, because a classifier that only predicts positive, when it has overwhelming evidence at hand might have a high precision, but an overall poor performance, because it produces a high amount of false negatives, fn. The recall of a classifier refers to the proportion of correctly predicted positive cases, tp among the total number of positive cases, i.e. $tp + fn$. It is also defined within $[0, 1]$. A classifier that always predicts class 0 in a two-class problem will have a high amount of true negatives tn, but no correctly classified true positives, leading to a recall of 0. A classifier with a high recall is reliable in the sense that it has a high probability of predicting a positive sample as positive (Grandini et al., 2020). The formula for calculating the recall is given in equation 22. For multi-class classification the recall may be calculate by dividing the amount of true positives of this class, tp by the sum of values within the respective row.

$$recall = \frac{tp}{tp + fn} \tag{22}$$

Based upon the critique of the accuracy score, the F-measure became a popular score for the evaluation of classifiers. It is the harmonic mean of precision and recall and enables evaluating a model in a single score. Similar to the other above-mentioned scores it is bounded between $(0, 1)$. For multi-class classification the F-measure of a label may not be calculable when this class was never correctly predicted. Equation 23 contains the formula for the calculation of the F-measure.

$$F - measure = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{23}$$

Another score used for assessment of the performance of a classifier is Matthews correlation coefficient (MCC). This score is bounded between $[-1, 1]$. It returns a high score only if the classifier was able to correctly predict the majority of positive instances, as well as the majority of negative instances. A value of 1 indicates perfect classification, whereas a value of -1 indicates perfect misclassification. The expected

value of the MCC for a randomly guessing classifier is 0. The mathematical properties of this score result in a more robust estimation of the overall performance of the classifier when compared to the accuracy or the F-measure (Chicco and Jurman, 2020). The formula for the calculation of the MCC is given in equation 24.

$$MCC = \frac{tp \cdot tn - fp \cdot fn}{\sqrt{(tp + fp) \cdot (tp + fn) \cdot (tn + fp) \cdot (tn + fn)}} \tag{24}$$

The given definitions were formulated for two-class classification problems but may be applied to multi-class classification as well. This may be done by converting the problem to k class vs. rest classification problems. Doing so enables calculating these scores for each class individually.

As discussed in Subsection 2.2.2, real-time prediction of the current activity is an important requirement. Therefore the average prediction time in milliseconds (ms) $\hat{\mu}_{ms}$ of the models is reported. The prediction time is defined as the time needed for the inference step, i.e. passing the data through the model. The data for this inference step was already within the needed format, i.e. the time needed to bring the raw data into the needed format is not included in the calculation of the average prediction time.

For the baseline method and the ShallowFFNet model, the overall prediction time was calculated based on the prediction time of the respective classifiers. The exact formula for the prediction time of hierarchically classifying models is given in equation 25.

$$\hat{\mu}_{ms} = \hat{\mu}_{top} + \frac{6}{7} \cdot \frac{\hat{\mu}_{lifting} + \hat{\mu}_{walking}}{2} \tag{25}$$

This was done to correctly estimate the needed time, under the assumption that all $k = 7$ activities appear equally often and the lifting or walking classifier gets called whenever the top-level classifier does not classify the activity as resting, see Figure 12.

6.2 Experimental Results

This section displays the results of training and testing various models. The baseline method, a hierarchically classifying Support Vector Machine applied to handcrafted features, is introduced in Chapter 4. The ShallowFFNet model, a shallow feedforward neural network applied to the same features as the baseline method, is explained in Section 5.3. The DeepFFNet model is a deeper feedforward network directly applied to the raw data, see Section 5.4. The ConvNet model is a convolutional neural network applied to the raw data, see Section 5.5 and the LSTMNet utilizes specialized recurrent units, see Section 5.6. Model-specific preprocessing of the data, as well as the regimen utilized to find the best-suited hyperparameters, are explained in these locations.

The origin of the training, test and validation data is explained in Chapter 3, and the scores utilized for the evaluation are defined in Subsection 6.1.2. All datsets used for the tables and figures within this section were balanced as explained in 3.4.5.

Table 5 summarizes the performance of the final models, i.e. the models trained with the best hyperparameters, on the unseen test data.

Table 5: Performance of final models on test data

Model	Input	Classifier	Accuracy	F-Measure	MCC	$\hat{\mu}_{ms}$
Baseline	Features	SVM	60.87%	55.43%	56.24%	0.64
ShallowFFNet	Features	Feedforward NN	71.46%	71.37%	66.81%	0.13
DeepFFNet	Raw	Feedforward NN	76.79%	76.69%	73.48%	0.22
ConvNet	Raw	Convolutional NN	70.91%	71.28%	66.46%	0.19
LSTMNet	Raw	LSTM NN	70.94%	71.24%	66.25%	0.75

For practitioners, it may be important to estimate the amount of occurred overfitting. To do this, the performance of the final models on the validation data is reported in Table 6.

Table 6: Performance of final models on validation data

Model	Input	Classifier	Accuracy	F-Measure	MCC	$\hat{\mu}_{ms}$
Baseline	Features	SVM	64.84%	60.24%	60.44%	0.63
ShallowFFNet	Features	Feedforward NN	82.25%	82.08%	79.43%	0.13
DeepFFNet	Raw	Feedforward NN	84.21%	84.11%	81.96%	0.22
ConvNet	Raw	Convolutional NN	80.42%	80.27%	77.39%	0.20
LSTMNet	Raw	LSTM NN	73.68%	73.55%	69.57%	0.75

Figure 15 contains a color-coded confusion matrix of the baseline method, explained in Chapter 4, on the unseen test data.

Figure 15: Confusion Matrix of the Baseline Method on test data

Table 7 contains the precision, recall and F-measure of the baseline method on the unseen test data. The F-measure for the activity lifting was set to N/A, short for not available, as calculating the F-measure would require dividing by zero, see equation 23.

Table 7: Precision, Recall and F-measure of the Baseline Method

Activity	Precision	Recall	F-Measure
Lifting	0.0	0.0	N/A
Dropping	49.29%	82.26%	61.64%
Holding	92.60%	84.36%	88.29%
Walking normal	46.55%	84.99%	60.15%
Walking up stairs	56.09%	44.07%	49.36%
Walking down stairs	95.74%	32.83%	48.89%
Resting	67.31%	97.56%	79.66%

Figure 16 contains the confusion matrix of the ShallowFFNet Model on unseen test data. An explanation of this model, as well as hyperparameter values for this model are given in Section 5.3.

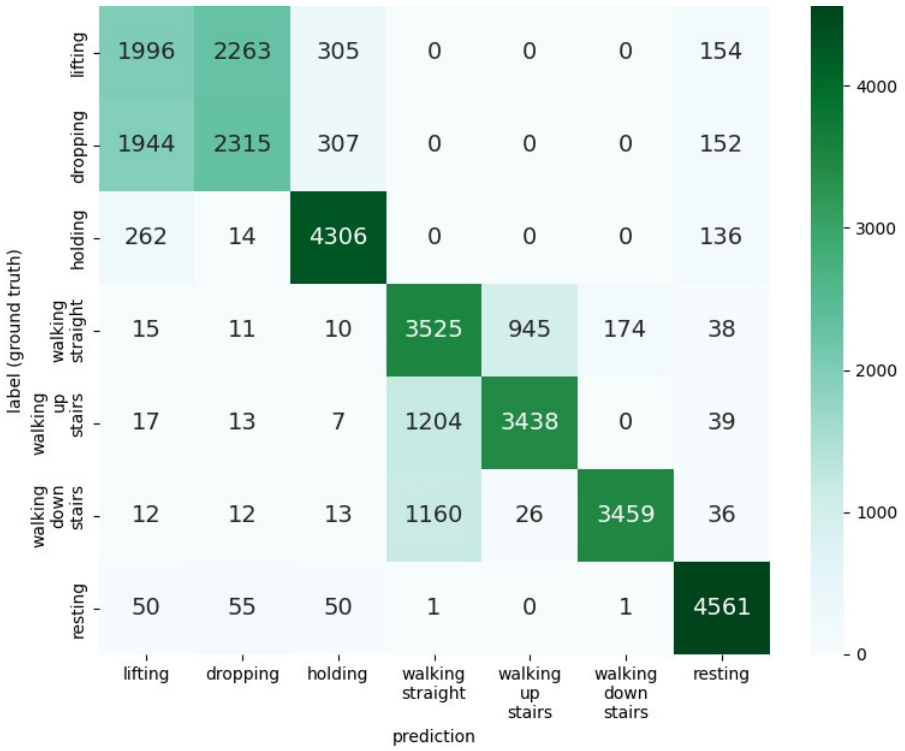

Figure 16: Confusion Matrix of the ShallowFFNet Model on test data

Table 8 contains the precision, recall and F-measure for each activity after converting the confusion matrix given in Figure 16 to seven activity versus rest classification problems.

Table 8: Precision, Recall and F-measure of the ShallowFFNet Model

Activity	Precision	Recall	F-Measure
Lifting	46.46%	42.31%	44.29%
Dropping	49.43%	49.07%	49.25%
Holding	86.15%	91.27%	88.64%
Walking normal	59.85%	74.71%	66.46%
Walking up stairs	77.98%	72.87%	75.34%
Walking down stairs	95.18%	73.31%	82.83%
Resting	89.15%	96.67%	92.72%

The confusion matrix for a feedforward network applied on the raw data, i.e. the DeepFFNet model, on the unseen test data is given in Figure 17. This model is introduced in Subsection 5.4.

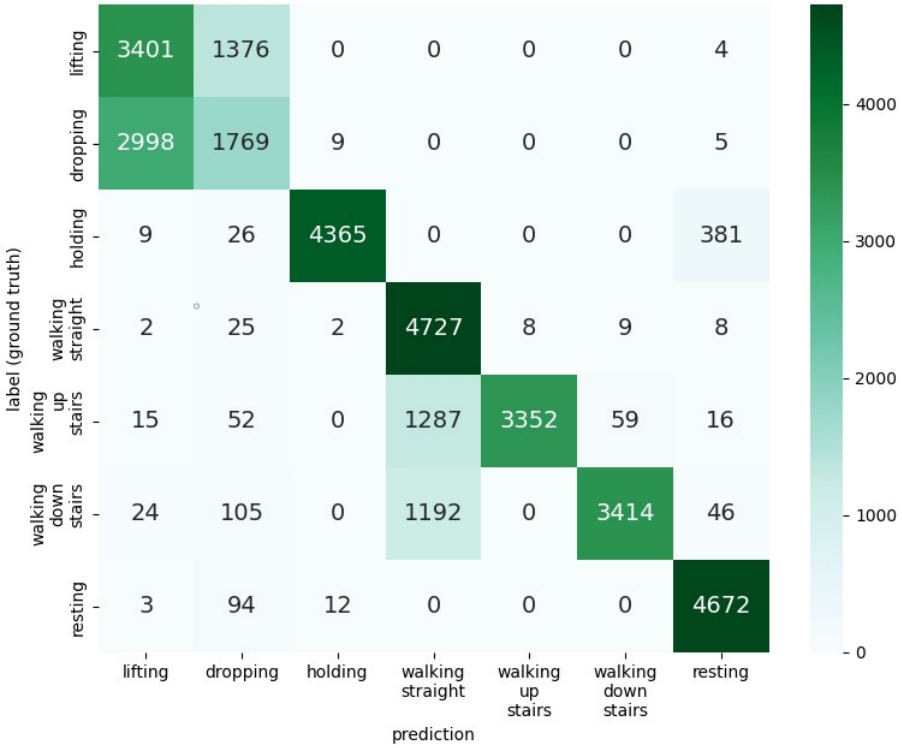

Figure 17: Confusion Matrix of the DeepFFNet Model on test data

The precision, recall and F-measure of the DeepFFNet model, when applied to the unseen test data, is given in Table 9.

Table 9: Precision, Recall and F-measure of the DeepFFNet Model

Activity	Precision	Recall	F-Measure
Lifting	52.71%	71.13%	60.55%
Dropping	51.31%	37.00%	42.99%
Holding	99.48%	91.29%	95.21%
Walking normal	65.59%	98.87%	78.69%
Walking up stairs	99.76%	70.11%	82.34%
Walking down stairs	98.05%	71.41%	82.63%
Resting	91.04%	97.72%	94.26%

Figure 18 contains the confusion matrix of the ConvNet model on unseen test data. The ConvNet model utilizes the convolution operation. The background of this architecture is laid out in Subsection 2.3.2, whereas the concrete hyperparameters of the ConvNet are laid out in Section 5.5.

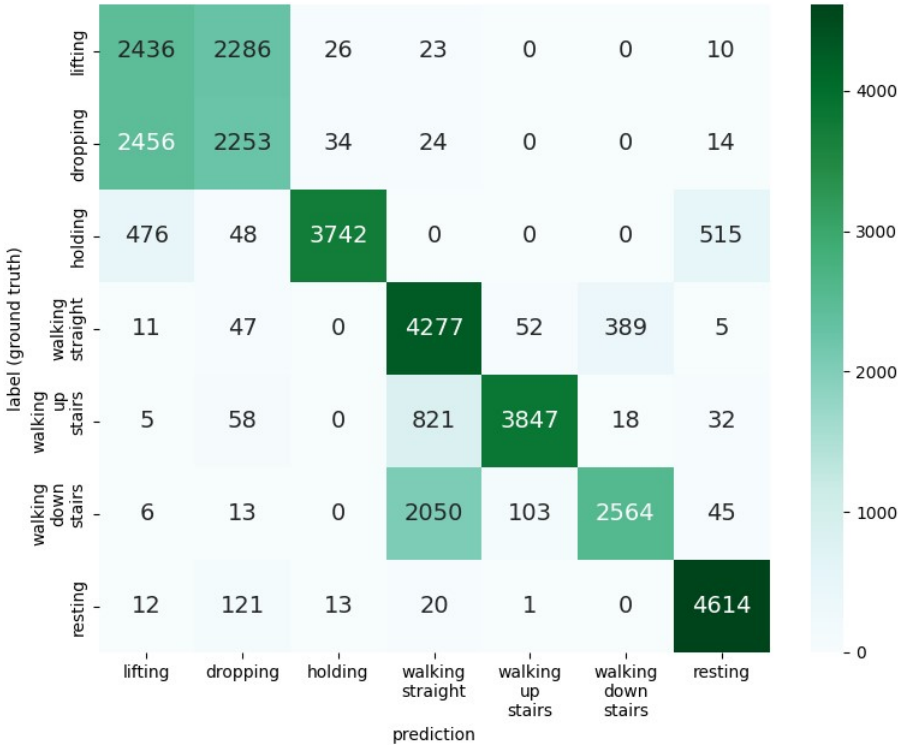

Figure 18: Confusion Matrix of the ConvNet Model on test data

The precision, recall and F-measure of the ConvNet model on unseen test data is given in Table 10.

Table 10: Precision, Recall and F-measure of the ConvNet Model

Activity	Precision	Recall	F-Measure
Lifting	45.09%	50.95%	47.84%
Dropping	46.68%	47.12%	46.90%
Holding	98.09%	78.27%	87.06%
Walking normal	59.28%	89.46%	71.31%
Walking up stairs	96.10%	80.45%	87.89%
Walking down stairs	86.30%	53.63%	66.15%
Resting	88.14%	96.51%	92.13%

Figure 19 contains the confusion matrix of the LSTMNet Model on unseen test data. The origin of the test data is laid out in Subsection 3.3.3. The hyperparameters of the LSTMNet model are laid out in Section 5.6.

Figure 19: Confusion Matrix of the LSTMNet Model on test data

Based on the evaluation visualized in Figure 19, Table 11 contains the respective precision, recall and F-measures for each activity. The values were calculated based on the performance of the LSTMNet model on unseen test data.

Table 11: Precision, Recall and F-measure of the LSTMNet Model

Activity	Precision	Recall	F-Measure
Lifting	43.80%	48.55%	46.05%
Dropping	44.90%	44.49%	44.69%
Holding	91.24%	74.47%	81.99%
Walking normal	65.99%	83.33%	73.66%
Walking up stairs	88.17%	83.39%	85.71%
Walking down stairs	91.73%	68.39%	78.36%
Resting	83.15%	93.99%	88.24%

Figure 20 displays the highest accuracy in % of the models trained throughout the hyperparameter optimization process on the validation data. As the best LSTMNet model did not contain any hidden layers, no experiments regarding dropout were performed for this model. To keep the number of experiments equal for the network types, three additional experiments regarding the architecture were performed for the LSTMNet model, see Section 5.6.

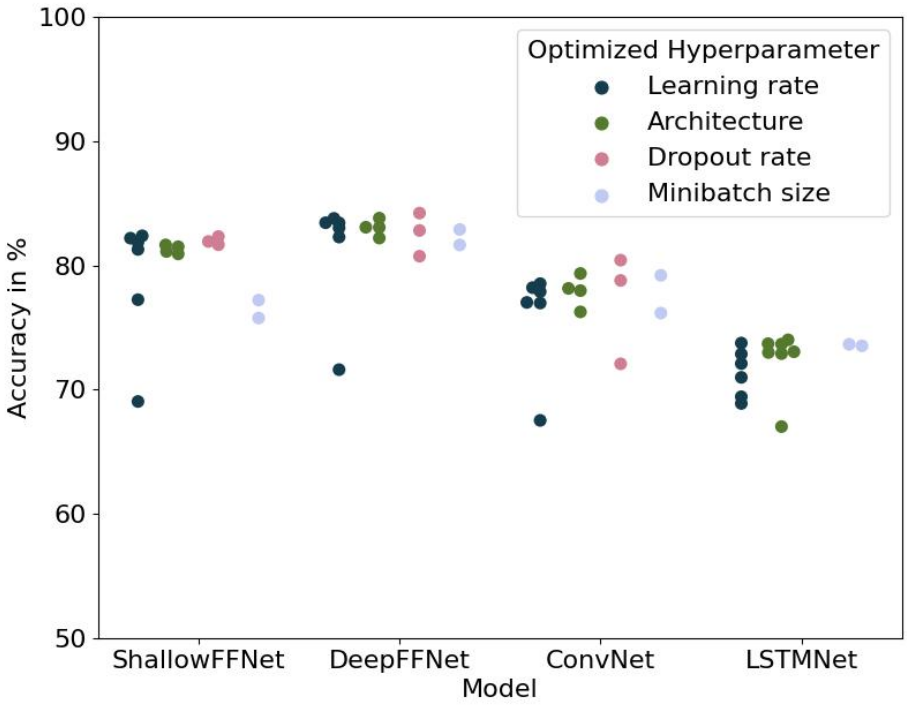

Figure 20: Accuracy of the models on validation data throughout Hyperparameter Optimization Process

7 Discussion

The following entails a discussion of the results seen in Section 6.2.

7.1 Approriatness of Deep Learning Models

The first research question is if the approach of deep learning is appropriate for human activity recognition (HAR) on active exoskeleton data. To answer this the performance of the baseline method is compared to the performance of neural networks, both applied to unseen data. Table 5 visualizes the performance of the trained neural networks and the baseline method on the unseen test data. For the evaluation of the performance the accuracy, F-measure and Matthews correlation coefficient (MCC) are used. These scores have an upper limit of one with a higher score denoting a better model. An introduction of these scores is given in Subsection 6.1.2. The baseline model achieved an accuracy of 60.87% on unseen test data, in comparison, the worst performing neural network achieved an accuracy of 70.91%. When using the F-measure and the MCC a similar picture emerges: The performance measures consistently indicate a better performance of the neural networks. This is interpreted as evidence supporting the claim that deep learning models are appropriate for the task of human activity recognition based on active exoskeleton data.

7.2 Best-performing neural network

Different architectures of neural networks were trained, as each can process the data in a different way. This was done to estimate, what information is most important for the successful classification of human activity. The following entails a discussion of the performance of the various network types.

A hierarchical classifier, utilizing the same information as the baseline method, however using neural networks as the classification algorithm outperforms the baseline method, see Tables 5, 7 and 8. While the baseline method achieved an F-measure of 55.43% on unseen test data, the ShallowFFNet model, which was trained on the same training data, achieved an F-measure on the same unseen test data of 71.37%. This indicates that the difference in the resulting scores between the baseline method and the neural networks is not due to a difference in the format of the available data, but due to the suboptimal usage of the available information by the baseline method. A plausible explanation for this difference in performance is that the baseline method might have overfitted on the training data. Based on the way this margin-based classifier finds the respective hyperplane, it is susceptible to mislabeling of the data, leading to unnecessarily complex decision boundaries, which do not reliably separate real-world data (Qiao and Zhang, 2015). The explanation of an overfitted baseline method is supported by the fact, that the hyperparameters of the baseline method were optimized on randomly chosen subsets of the training

data. It is plausible that this differing process of hyperparameter optimization may have lead to a greater extend of overfitting in the baseline method. The utilized neural networks of the ShallowFFNet model were rather small, i.e. no classifier of this model had more than 310 parameters to be optimized throughout the training, see Section 5.3. As a small number of parameters translates to smaller representational capacity, this smaller capacity may have prevented them from learning overly complex decision functions.

The approach of using hand-crafted features as input for the final classifier resulted in acceptable performance comparable to other networks trained on the raw data, as the ConvNet model achieved an F-measure of 71.28% on unseen test data and the LSTMNet model an F-measure of 71.24%. These scores were calculated on unseen test data, see Table 5. Using features as input for the final classification system seems to be an acceptable choice, given these features contain enough information. Domain knowledge may be required for the process of generating and selecting features. Choosing the correct classification algorithm which classifies based on these features has a considerable effect on the performance of the overall activity recognition system.

The DeepFFNet model was trained to observe if providing the classifier with the means to learn useful representations directly from the raw data does enable the model to achieve a better performance compared to models receiving the data in the format of features. As the resulting model got the highest accuracy, F-measure and MCC, see Table 5, this is interpreted as evidence supporting the claim that representation learning provides an advantage for this task. When compared against the performance of the ShallowFFNet model, the DeepFFNet model performs reliably better. This is either due to having more information available, i.e. the raw data instead of features or the ability to better extract useful representations within the hidden layers. An interesting finding is that the difference in performance on the validation and test set is bigger on the ShallowFFNet model when compared to the DeepFFNet model, see Tables 5 and 6. This indicates that relying on handcrafted features in combination with small neural networks is not a useful strategy for the prevention of overfitting. A bigger feedforward model, directly applied to the raw data generalized better than the set of hierarchically classifying simpler networks in the ShallowFFNet model. As the DeepFFNet model reliably achieved good performance during training, see Figure 20, practitioners may be well advised to invest resources directly in training such models rather than spending time on feature engineering. The research reported well-performing standard feedforward models, even when compared to convolutional or recurrent networks (Hammerla et al., 2016).

The ConvNet model was built to test if providing the network with the means of detecting local patterns within the time structure of the data enables the model to perform better when compared to a classical feedforward network. As the performance of the ConvNet is similar to the ShallowFFNet and LSTMNet model, this indicates that this ability does not seem to bring any relevant advantage, see Tables 5, 10 and 11. To the author this result came surprisingly, considering that the state of the art of HAR, makes use of convolutional models. There are several

possible explanations for this behavior, the most plausible one being that the literature made use of a more rigorous exploration of the hyperparameter space, which enabled finding a better-suited constellation. During training the ConvNet model achieved consistent performance, i.e. the performance for various constellations of hyperparameters was similar, see Figure 20. This finding is in line with the results of large-scale studies in which this robustness has also been observed (Hammerla et al., 2016).

A similar research question was addressed by the training of the LSTMNet model, which had a greater capacity for processing long-term relationships within the data. Analog to the ConvNet model, this did not seem to bring any advantages and the performance of the model is comparable to the ShallowFFNet model. Utilizing this capacity of the model to better process the flow of information through the model did not seem to yield an advantage over a standard feedforward network, as represented by the DeepFFNet model.

As recurrent models are frequently used within the current state of the art of HAR, the most plausible explanation for their lack of performance within this study is the limited search for well-suited hyperparameters (Pesenti et al., 2023). Analog to convolutional models, the training of recurrent models resulted in models with constant performance, see Figure 20.

7.3 Comparatively low level of performance

Another important observation is that the overall performance of the models is substantially lower than reported in the current state of the art, see Subsection 2.2.3. There are three main reasons for this:

7.3.1 Models unable to distinguish Lifting from Dropping

The main reason for this is the definition of the activities to be classified. As laid out in Section 3.2, the activities lifting and dropping were defined as activities not only involving the moving of a weight into a different height but also including the process of returning to the original position to grab the next weight. As a result, similar sensor values had differing labels attached, i.e. moving the torso downwards while holding a weight was classified as dropping, whereas moving the torso downwards without holding a weight was labeled as lifting. None of the trained models was able to reliably classify if this moving of the torso was being done with or without a weight, resulting in being unable to distinguish lifting from dropping. The models were randomly guessing between lifting and dropping, whenever one of them was detected. This effect can be seen when observing the distribution of the values within the upper left corner of the Figures 15, 16, 17, 18 and 19. This effect numerically expressed in the precision, recall and F-measure achieved on these activities, see Tables 7, 8, 9, 10 and 11. The phenomenon of kinetically similar activities being difficult to differentiate is observed within the literature (Kuschan et al., 2021).

Based on the inability of the models to discern two of the $k = 7$ classes the maximum expected accuracy was $100\% - \frac{1}{7} = 85.72\%$. As it was assumed that the models would be able to distinguish lifting from dropping based on this definition, the classes were defined as such at the beginning of the study. As a result, the models seem to perform significantly worse than the current state of the art, but considering that two classes could not be separated based on the available data the resulting models seem acceptable. The best model achieved an accuracy of 76.79% on the test data, with a theoretical maximum accuracy of 85.72%. Setting these values into relationship yields

$$\frac{76.79\%}{85.72\%} \approx 93.08\% \tag{26}$$

This performance is indeed comparable to the current state of the art.

7.3.2 Training Data Class Imbalance

As laid out in Subsection 3.4.2, the training data was affected by class imbalance. This was especially severe regarding the walking activities. The activities of walking up and walking down stairs were observed for less than four minutes each, see Figure 10. This lack of information made the detection of patterns distinguishing walking activities difficult. Even on more balanced data sets, discerning these classes was shown to be susceptible to error (Valarezo et al., 2017).

7.3.3 No direct Comparison with State of the Art possible

Furthermore, there is no study on HAR using deep learning on the data from active exoskeletons to date, which utilizes a test set recorded at different locations and containing only subjects never seen in the training data, see Subsection 2.2.3. Therefore the results of this study cannot be compared without restriction, as there are valid reasons to assume, that at least some sort of information about the subjects or the shared environment could have been utilized by the models within the literature to correctly classify the test set.

Based on the three above-mentioned arguments, i.e. the inability to discern lifting from dropping, heavy class imbalance, as well as the application to a test set that contains subjects and environments not observed throughout the training process the conclusion is drawn, that the trained models do perform adequately. A purely numerical comparison with the state of the art reviewed in Subsection 2.2.3 is misleading.

7.4 Robustness of Deep Learning Models

Another research question aimed at the model's ability to generalize beyond the subjects and environments seen in the training data. Based on the results of this study

with $n = 9$ subjects observed in the training data, see Subsection 3.3.1, the neural networks trained seem to generalize rather well, as the best model achieved an accuracy of 84.21% on the validation data and 76.79% on the test data. These results are especially promising considering, that a singular environment was observed within the training and validation data. This might have led to some overfitting on the shared environment throughout the hyperparameter optimization process. Furthermore, the training data contained little data on stair-related activities. This data was oversampled, but the chosen oversampling technique simply adjusted the importance of these samples for the loss function and did not insert any new information about these classes into the dataset. Considering all this the models classified new data in a relatively robust manner.

For each model, the extent of overfitting can be estimated from the difference in performance on the validation and test data, see Tables 5 and 6. The biggest difference between the performance on the validation and test data occurred at the ConvNet model, followed by the ShallowFFNet model. The performance gap of the DeepFFNet model is slightly smaller. The smallest differences are recorded on the final LSTMNet and baseline method. The LSTMNet model has a difference regarding the accuracy of test and validation data of 2.74%, which may be indicative of underfitting. In combination with the successful prior usage of these recurrent models the conclusion is drawn, that recurrent models might be well-suited for the given task. These models might outperform standard feedforward models, albeit at the cost of making a diligent search through the hyperparameter space necessary. Training these networks takes more time than training a feedforward neural network, because of reasons explained in Subsection 2.3.3. Based on these two observations, the training of recurrent networks could be appropriate if sufficient resources are available for comprehensive training (Wang et al., 2019a).

This study was motivated by the goal of adjusting the support generated by the exoskeleton according to the current need of the user. In addition to finding initial evidence for appropriate network types, this study looked at the robustness of the trained models, i.e. their ability to infer reliably when applied to new users in unseen environments. The results indicate that the trained models are able to perform classification on unseen data quite well. This result must be taken with caution, as there are reasons indicating that these results may overestimate the performance when applied to real-world data. The first is that the training, validation and test data were recorded in rather artificial environments, i.e. the subjects knew what was being recorded and may therefore have changed their behavior to perform the activities in a particularly clean or ergonomically healthy way. This effect may have been stronger when the subjects were under direct observation. This was the case in the validation and test data, see Section 3.3. The process of labeling the data encouraged removing ambiguous activities or any activities, which did not match the expectations of the labeling human regarding the specific movement patterns. By doing so all the datasets might be subject to a process that leads to clearer differences between the activity patterns, than may be observed within the real world. Another important consideration is that none of the datasets used

in this study are representative. The testers and subjects used were not randomly selected from a clearly defined larger population. Some groups that might use the exoskeletons were never observed in any of the datasets, e.g. none of the datasets included a person over the age of 53. Given these limitations, the results of this study should be interpreted as preliminary evidence that neural networks appear to generalise quite well for this application. Good generalization is not the only requirement for a model to be used in the application for the control of the support by an exoskeleton. Several other requirements, listed in Subsection 2.2.2, would also need diligent exploration and examination.

7.5 Evaluation time

In terms of inference time, it is interesting to note that the number of parameters in the neural networks seems to have little effect. Standard feedforward passes appear to have almost identical computational requirements, regardless of the number of parameters involved. This is inferred from the difference in inference time between the ShallowFFNet and DeepFFNet model, see Tables 5 and 6. The convolution operation adds little additional computational time. This may be due to the ability to parallelize this operation. The inherent sequential nature of the LSTMNet model did increase the time required. Comparisons between the baseline method and the neural networks cannot be made, as they are based on different libraries and are therefore optimized differently. In general, these results should be taken cautiously, as these inference times are not gathered from edge devices and the used hardware greatly affects the resulting times.

7.6 Results from the Hyperparameter Optimization Process

Another goal of this study was to communicate experience with training neural networks for this application. Therefore the following discusses the effects of hyperparameter optimization on the performance on the validation data. This discussion refers to results illustrated in Figure 20.

For each trained network, the learning rate was varied six times at the beginning of the hyperparameter optimization process, initially changing the learning rate for orders of magnitude. These six experiments yielded an accuracy improvement on the validation data of 0.5% to 3.5%, with models having a smaller initial performance experiencing bigger improvements. Optimizing this hyperparameter yielded the biggest increase in performance. The literature reports similar results for studies of larger scale (Hammerla et al., 2016).

Afterward, the structure of the model was varied four times, as described in Section 5.2. This optimization resulted in improvements regarding accuracy on the validation data between 0.1% and 1.2%. If the initial model performed well, adding additional layers never improved performance on the validation data, and adding

units to existing layers often yielded only minimal bonus in performance. The complexity of the task to be solved by the network set clear limits on the structure and complexity of the model: Classifying the activity based on selected features is an easier task than learning and constructing these features based on the raw data, so the ShallowFFNet model was able to perform well using three networks, each containing a single hidden layer of eight units, whereas the DeepFFNet model had to use three hidden layers. Adding additional layers did not increase the performance of the DeepFFNet model. These findings are in accordance with prior studies, stating that utilizing networks with more than three hidden layers for human activity recognition seldom leads to improved performance (Hammerla et al., 2016).

Changing the used dropout rate either did not improve accuracy on the validation data or by small margins, i.e. $\approx 1\%$. Improving the model via dropout was usually achieved with a dropout rate of 10%. It remains unclear why higher dropout rates did not lead to increased performance, however, there are two plausible explanations: Either the relatively large amount of training data regularized the neural networks enough so that including dropout did not increase performance or the networks were too small to profit from dropout. As explained in Subsection 2.3.1, using dropout might make utilizing larger network sizes necessary. Because of the training schedule, the network size was treated as fixed when optimizing this hyperparameter, so the positive effects of using a higher dropout rate in combination with a larger network could not be explored.

Regarding the minibatch size, the results of the experiments are as expected from the literature: Using a smaller minibatch size increases the performance on the validation data, albeit at the cost of longer training time (Masters and Luschi, 2018). Throughout the hyperparameter optimization phase, no network with a larger minibatch size was able to outperform its counterpart trained with a smaller minibatch size. Ideally, minibatch size and learning rate would be optimized together, as smaller minibatches contain less information and a more noisy approximation of the information within the overall training data, therefore a lower learning rate may be appropriate for them.

8 Future Work

The following entails a discussion of possibilities for future research.

Current research neglects the importance of unbiased estimation of the model's ability to generalize. To the best knowledge of the author, this is the first study regarding HAR on low-back exoskeletons via deep learning, which utilizes unseen environments combined with unseen subjects for the test data. Recording representative datasets, which include a wider variety of subjects and locations, is necessary to prevent overestimating the model's ability to generalize (Kuschan et al., 2021). Using these overfitted models might be dangerous, as the classification of the models might be used to influence the behavior of the active exoskeletons and lead to inappropriate support, endangering the user. Therefore similar work, which extends the number of subjects and locations for the test data via a representative sample of a clear-defined population is necessary to correctly estimate the ability of deep learning models to generalize. The existing research evaluates the performance of the models via the holdout method, see subsections 2.1.2 and 2.2.3. Utilizing larger and more diverse datasets enables a more stable estimation of the robustness of the models via k-fold cross-validation, see Subsection 2.1.2

As creating large-scale representative datasets with an adequate variety might prove difficult for practitioners, data-generating techniques might be leveraged to increase the variety of the available data. Training of generative models to produce additional training data may improve the generalization of said models, however there is little experience available regarding the usage of these models for HAR (Gu et al., 2021). Comparing the robustness of models trained using data augmentation techniques may provide insight into the appropriateness of this approach. Investigating this topic may provide a low-cost way to increase the robustness of models.

Closely related to the effects of generating additional samples are the effects of balancing on the ability of the model to generalize. This paper has used a basic oversampling technique to balance the activities, but future work could evaluate the effects of more sophisticated techniques on the generalization of the model. These more sophisticated algorithms could generate additional minority class samples by interpolating between observed minority class samples, as is done by the Synthetic Minority Oversampling Technique (Chawla et al., 2002).

As a combination of the different approaches mentioned above, HAR with little training data is another interesting area of research. Answering the question of how many different subjects should be included in a dataset for what minimum amount of time to achieve a given level of performance could enable the use of HAR on exoskeleton data in low data scenarios. Experience could be gained by successively adding testers to the available data pool and gradually increasing the time each tester is observed. For each step, models are trained and evaluated. This could provide insight into the minimum amount of data required for a given level of performance.

As explained in Subsection 2.2.3, the current research mainly focuses on training the models after finishing recording the data and afterward using the final models without further adjustment to them (Wang et al., 2019a). As the use of active exoskeletons in the field provides more unlabelled data, integrating this additional data into the model could improve robustness. Online and incremental learning, possibly even on the edge devices themselves, may prove essential for classifying more complex activities. To the best of the author's knowledge, there is no study to date that evaluates the possibility of online learning based on exoskeleton data.

Another topic in HAR based on active exoskeleton data requiring further research is the possibility of training individual models per user. Doing so might increase the performance of said models by reducing the problem of intraclass variety, as explained in Subsection 2.2.2. This may be implemented by starting with an initial pre-trained model which gets more adapted to the specific user over time, as more data is collected from this user. Some authors assume that using individual models is necessary for real-world application to remove the effects of inter-person variety within the activity classes (Pesenti et al., 2023).

As multiple activities may be performed at the same time, models which allow for the simultaneous classification of activities may be explored. These activities may either be hierarchically ordered or specific to certain limbs, e.g. the leg-related activity may be walking, while the arm-related activity is holding, meaning that the user is carrying an object. This more fine-grained classification may allow for more precise control of the support generated by the active exoskeleton (Gu et al., 2021).

Regarding the model architecture, there is a wide range of experiences gathered within the field of HAR, however when applied to exoskeleton data, there is little data available. Large-scale explorations of different model architectures, as well as the effects of hyperparameter optimization on these model types, remain to be done. Several architectures, such as hybrid models combining convolution operations with recurrent units, remain unexplored within the existing literature (Wang et al., 2019a). Throughout this thesis, some hyperparameters such as the window size and the resampling rate were taken as fixed to enable comparison between the models, albeit at the cost of not being able to estimate the effects of varying these parameters. In Chapter 7 evidence regarding underfitting of the recurrent model was gathered, therefore a more in-depth exploration of hyperparameters for this model type seems promising. Within these large-scale studies, another interesting research topic is the evaluation of transformer models. These have achieved better performance than convolutional and recurrent networks when applied to activity recognition tasks, however they have not been studied in the context of HAR on exoskeleton data (Shavit and Klein, 2021).

Research into computationally cheap and real-time classification for HAR on exoskeleton data is another little explored area. Reducing the computational burden has several benefits for the usability of the model, such as reduced energy consumption and enabling smaller hardware to perform the classification on edge devices. The importance of this is recognized in the field, as one of the first studies measured

and reported the time required for classification on edge devices (Jaramillo et al., 2022).

In addition, no research to date investigates the possibility of HAR for multiple exoskeleton users. As the exoskeleton users could be working in the same environment and interacting with each other, the sensor readings from multiple exoskeletons could be aggregated for a joint classification of activities involving multiple subjects, such as carrying a heavy object together.

Adding additional data sources to the data available to the exoskeleton might improve prediction and enable more precise identification of the user's interaction with the environment. As active exoskeletons might be utilized within modern factories, which are highly connected and equipped with a multitude of sensors to identify and locate products, the data from these object-bound sensors may be combined with data stemming from the exoskeleton (Hozdic, 2015). This application has potential regarding injury prevention, as the data from the products might contain information regarding the weight, enabling the exoskeleton to provide optimal support for handling said product.

As the classification of human activity could be used to influence the behavior of the force-generating components of active exoskeletons, predicting future activity could provide the means to control the relevant motors in a smoother, more adaptive way. By using activity prediction, the amount of time needed to apply supporting forces could be reduced. This topic has been explored for other applications of HAR, while it remains an unexplored topic for exoskeletons (Du et al., 2019).

If the models developed are to be used in practice, security issues need further research. The impact of misclassification on the user varies: Similar tasks require similar assistance, so confusing an activity with a similar one, e.g. walking normal with stair climbing, has little impact on the user. If the classified activity, e.g. lifting, is different from the real activity, e.g. resting, the exoskeleton could provide unwanted support and move the worker unexpectedly. Current research does not take into account that the cost of misclassification varies. To provide safe and reliable support, this information would need to be incorporated into the training process of neural networks. A general framework for evaluating the safety of exoskeleton control by human activity recognition systems would need to be developed and tested.

9 Conclusion

The present study tested the performance of various types of neural networks for the task of human activity recognition based on active exoskeleton data. A test set containing $n = 10$ new subjects at five previously unseen locations was recorded. The comparison with a model representing the conventional, feature-based approach to human activity recognition indicates that deep learning models are appropriate for this task.

Representation learning via neural networks applied to raw sensor data can outperform the conventional approach to human activity recognition for this use case. A fully-connected feedforward neural network applied directly to the raw and flattened sensor data was able to classify unseen data with acceptable performance.

Specialized neural network architectures making usage of recurrency or the convolution operation underperformed within this study, indicating that their successful usage requires a more thorough investigation of the hyperparameter space. As the model, which utilized long short-term memory units, showed signs of underfitting it seems plausible that this class of neural networks may be able to perform better, given the correct hyperparameter constellation.

The hyperparameter optimization process confirmed knowledge from existing literature, that optimizing the learning rate does have a greater effect on the performance of the model when compared to the effects of other hyperparameters. The second-most important hyperparameter is the architecture of the utilized network. As the utilized networks for this study were rather small and the architecture of the network was treated as fixed when optimizing the influence of regularization via dropout, all possible effects of this regularization method could not be investigated. Positive effects were found when using small dropout rates, such as 0.1. A negative correlation between the minibatch size and the performance of the model was detected.

Regarding the robustness of said models, evidence was gathered that deep learning models prove to be robust when applied to new locations or new subjects. These results apply with various limitations, therefore further research is needed to confirm these findings.

A call for a more thorough exploration of possible hyperparameter constellations for the various model types is made. There are various important research questions to be answered: These include exploring the effects of utilizing datasets of greater variety on the robustness of the resulting models, testing the effects of data generating techniques and various oversampling techniques. Online and incremental learning, e.g. for adapting to individual workers, is another currently unexplored topic.

Before the human activity recognition system can safely be used to control the behavior of the force-generating elements of exoskeletons, diverse research is needed to prove that deep learning models are adequate for this task. This includes proving that the safety, stability, robustness and overall performance of the models are appropriate.

Bibliography

Athar Ali, Vigilio Fontanari, Werner Schmoelz, and Sunil K. Agrawal. Systematic Review of Back-Support Exoskeletons and Soft Robotic Suits. *Frontiers in Bioengineering and Biotechnology*, 9, November 2021. ISSN 2296-4185. doi: https://doi.org/10.3389/fbioe.2021.765257. URL `https://www.frontiersin.org/articles/10.3389/fbioe.2021.765257`.

Manfredo Atzori, Arjan Gijsberts, Claudio Castellini, Barbara Caputo, Anne-Gabrielle Mittaz Hager, Simone Elsig, Giorgio Giatsidis, Franco Bassetto, and Henning Müller. Electromyography data for non-invasive naturally-controlled robotic hand prostheses. *Scientific Data*, 1(1):140053, December 2014. ISSN 2052-4463. doi: 10.1038/sdata.2014.53. URL `https://www.nature.com/articles/sdata201453`. Number: 1 Publisher: Nature Publishing Group.

Akram Bayat, Marc Pomplun, and Duc A. Tran. A Study on Human Activity Recognition Using Accelerometer Data from Smartphones. *Procedia Computer Science*, 34:450–457, August 2014. ISSN 18770509. doi: 10.1016/j.procs.2014.07.009. URL `https://linkinghub.elsevier.com/retrieve/pii/S1877050914008643`.

Valentina Bianchi, Marco Bassoli, Gianfranco Lombardo, Paolo Fornacciari, Monica Mordonini, and Ilaria De Munari. IoT Wearable Sensor and Deep Learning: An Integrated Approach for Personalized Human Activity Recognition in a Smart Home Environment. *IEEE Internet of Things Journal*, 6(5):8553–8562, May 2019. ISSN 2327-4662, 2372-2541. doi: 10.1109/JIOT.2019.2920283. URL `https://ieeexplore.ieee.org/document/8727452/`.

Luigi Bibbò, Riccardo Carotenuto, and Francesco Della Corte. An Overview of Indoor Localization System for Human Activity Recognition (HAR) in Healthcare. *Sensors*, 22(21):8119, October 2022. ISSN 1424-8220. doi: 10.3390/s22218119. URL `https://www.mdpi.com/1424-8220/22/21/8119`. Number: 21 Publisher: Multidisciplinary Digital Publishing Institute.

Christopher M. Bishop. *Pattern recognition and machine learning*. Information science and statistics. Springer, New York, August 2006. ISBN 978-0-387-31073-2.

Rich Caruana, Steve Lawrence, and C Lee Giles. Overfitting in Neural Nets: Backpropagation, Conjugate Gradient, and Early Stopping. In *Proceedings of the 13th International Conference on Neural Information Processing Systems*, NIPS, pages 381–387. MIT Press, Cambridge, MA, USA, January 2000.

Girish Chandrashekar and Ferat Sahin. A survey on feature selection methods. *Computers & Electrical Engineering*, 40(1):16–28, January 2014. ISSN 00457906. doi: 10.1016/j.compeleceng.2013.11.024. URL `https://linkinghub.elsevier.com/retrieve/pii/S0045790613003066`.

N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research*, 16:321–357, June 2002. ISSN 1076-9757. doi: 10.1613/jair.953. URL https://www.jair.org/index.php/jair/article/view/10302.

Jingcheng Chen, Yining Sun, and Shaoming Sun. Improving Human Activity Recognition Performance by Data Fusion and Feature Engineering. *Sensors*, 21(3):692, January 2021. ISSN 1424-8220. doi: 10.3390/s21030692. URL https://www.mdpi.com/1424-8220/21/3/692. Number: 3 Publisher: Multidisciplinary Digital Publishing Institute.

Davide Chicco and Giuseppe Jurman. The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation. *BMC Genomics*, 21(1):6, January 2020. ISSN 1471-2164. doi: 10.1186/s12864-019-6413-7. URL https://doi.org/10.1186/s12864-019-6413-7.

Waltenegus Dargie. Analysis of Time and Frequency Domain Features of Accelerometer Measurements. In *2009 Proceedings of 18th International Conference on Computer Communications and Networks*, pages 1–6, San Francisco, CA, USA, August 2009. IEEE. doi: 10.1109/ICCCN.2009.5235366. URL http://ieeexplore.ieee.org/document/5235366/.

Larissa Brentini de Almeida, Edgar Ramos Vieira, José Eduardo Zaia, Branca Maria de Oliveira Santos, Américo Riccardi Vaccari Lourenço, and Paulo Roberto Veiga Quemelo. Musculoskeletal disorders and stress among footwear industry workers. *Work*, 56(1):67–73, February 2017. ISSN 10519815, 18759270. doi: 10.3233/WOR-162463. URL https://www.medra.org/servlet/aliasResolver?alias=iospress&doi=10.3233/WOR-162463.

Michiel P. de Looze, Tim Bosch, Frank Krause, Konrad S. Stadler, and Leonard W. O'Sullivan. Exoskeletons for industrial application and their potential effects on physical work load. *Ergonomics*, 59(5):671–681, May 2016. ISSN 0014-0139, 1366-5847. doi: 10.1080/00140139.2015.1081988. URL https://www.tandfonline.com/doi/full/10.1080/00140139.2015.1081988.

Yegang Du, Yuto Lim, and Yasuo Tan. A Novel Human Activity Recognition and Prediction in Smart Home Based on Interaction. *Sensors*, 19(20):4474, January 2019. ISSN 1424-8220. doi: 10.3390/s19204474. URL https://www.mdpi.com/1424-8220/19/20/4474. Number: 20 Publisher: Multidisciplinary Digital Publishing Institute.

Alam Fakhri, ZhaiHe Zhou, and JiaJia Hu. A Comparative Analysis of Orientation Estimation Filters using MEMS based IMU. In *2nd International Conference on Research in Science, Engineering and Technology (ICRSET'2014), March 21-22, 2014 Dubai (UAE)*. International Institute of Engineers, March 2014. ISBN 978-93-82242-81-9. doi: 10.15242/IIE.E0314552. URL http://iieng.org/siteadmin/upload/7697E0314552.pdf.

Alberto Fernandez, Salvador Garcia, Francisco Herrera, and Nitesh V. Chawla. SMOTE for Learning from Imbalanced Data: Progress and Challenges, Marking the 15-year Anniversary. *Journal of Artificial Intelligence Research*, 61:863–905, April 2018. ISSN 1076-9757. doi: 10.1613/jair.1.11192. URL `https://www.jair.org/index.php/jair/article/view/11192`.

Vaishali Ganganwar. An overview of classification algorithms for imbalanced datasets. *International Journal of Emerging Technology and Advanced Engineering*, 2(4), 2012. ISSN 2250-2459. URL `https://www.researchgate.net/profile/Vaishali-Ganganwar/publication/292018027_An_overview_of_classification_algorithms_for_imbalanced_datasets/links/58c7707a458515478dc4c68b/An-overview-of-classification-algorithms-for-imbalanced-datasets.pdf`.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, November 2016. ISBN 978-0-262-33737-3. Google-Books-ID: omivDQAAQBAJ.

Margherita Grandini, Enrico Bagli, and Giorgio Visani. Metrics for Multi-Class Classification: an Overview, August 2020. URL `http://arxiv.org/abs/2008.05756`. arXiv:2008.05756 [cs, stat].

Lorenzo Grazi, Emilio Trigili, Giulio Proface, Francesco Giovacchini, Simona Crea, and Nicola Vitiello. Design and Experimental Evaluation of a Semi-Passive Upper-Limb Exoskeleton for Workers With Motorized Tuning of Assistance. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 28(10):2276–2285, October 2020. ISSN 1534-4320, 1558-0210. doi: 10.1109/TNSRE.2020.3014408. URL `https://ieeexplore.ieee.org/document/9159674/`.

Lorenzo Grazi, Emilio Trigili, Noemi Caloi, Giulia Ramella, Francesco Giovacchini, Nicola Vitiello, and Simona Crea. Kinematics-Based Adaptive Assistance of a Semi-Passive Upper-Limb Exoskeleton for Workers in Static and Dynamic Tasks. *IEEE Robotics and Automation Letters*, 7(4):8675–8682, October 2022. ISSN 2377-3766, 2377-3774. doi: 10.1109/LRA.2022.3188402. URL `https://ieeexplore.ieee.org/document/9815517/`.

Rene Grzeszick, Jan Marius Lenk, Fernando Moya Rueda, Gernot A. Fink, Sascha Feldhorst, and Michael ten Hompel. Deep Neural Network based Human Activity Recognition for the Order Picking Process. In *Proceedings of the 4th International Workshop on Sensor-based Activity Recognition and Interaction*, pages 1–6, Rostock Germany, September 2017. ACM. ISBN 978-1-4503-5223-9. doi: 10.1145/3134230.3134231. URL `https://dl.acm.org/doi/10.1145/3134230.3134231`.

Fuqiang Gu, Mu-Huan Chung, Mark Chignell, Shahrokh Valaee, Baoding Zhou, and Xue Liu. A Survey on Deep Learning for Human Activity Recognition. *ACM Computing Surveys*, 54, August 2021. doi: 10.1145/3472290.

Saurabh Gupta. Deep learning based human activity recognition (HAR) using wearable sensor data. *International Journal of Information Management Data*

Insights, 1(2):100046, November 2021. ISSN 2667-0968. doi: 10.1016/j.jjimei.2021.100046. URL https://www.sciencedirect.com/science/article/pii/S2667096821000392.

Ye Haibo, Gu Tao, Tao Xianping, and Lu Jian. Scalable floor localization using barometer on smartphone. *Wireless Communications and Mobile Computing*, 16(16):2557–2571, October 2016. ISSN 1530-8669. doi: 10.1002/wcm.2706. URL https://onlinelibrary.wiley.com/doi/10.1002/wcm.2706?casa_token=WMWSkWvjSAsAAAAA%3AkSqph3VxTTUZe60Sw4l8Ud-0nLcmF18sdD5OBaat1xrRMUGr59PUD5JM0jTnYy1DJZZ1-McSzvPPbUe9.

Nils Y. Hammerla, Shane Halloran, and Thomas Ploetz. Deep, Convolutional, and Recurrent Models for Human Activity Recognition using Wearables, April 2016. URL http://arxiv.org/abs/1604.08880. arXiv:1604.08880 [cs, stat].

Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors, July 2012. URL http://arxiv.org/abs/1207.0580. arXiv:1207.0580 [cs].

Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. Conference Name: Neural Computation.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, January 1989. ISSN 08936080. doi: 10.1016/0893-6080(89)90020-8. URL https://linkinghub.elsevier.com/retrieve/pii/0893608089900208.

Elvis Hozdic. Smart Factory for industry 4.0. *International Journal of Modern Manufacturing Technologies*, 2015. ISSN 20673604, 20673604.

Md Milon Islam, Sheikh Nooruddin, Fakhri Karray, and Ghulam Muhammad. Human Activity Recognition Using Tools of Convolutional Neural Networks: A State of the Art Review, Data Sets, Challenges and Future Prospects. *Computers in Biology and Medicine*, 149:106060, October 2022. ISSN 00104825. doi: 10.1016/j.compbiomed.2022.106060. URL http://arxiv.org/abs/2202.03274. arXiv:2202.03274 [cs, eess].

Ismael Espinoza Jaramillo, Jin Gyun Jeong, Patricio Rivera Lopez, Choong-Ho Lee, Do-Yeon Kang, Tae-Jun Ha, Ji-Heon Oh, Hwanseok Jung, Jin Hyuk Lee, Won Hee Lee, and Tae-Seong Kim. Real-Time Human Activity Recognition with IMU and Encoder Sensors in Wearable Exoskeleton Robot via Deep Learning Networks. *Sensors*, 22(24):9690, January 2022. ISSN 1424-8220. doi: 10.3390/s22249690. URL https://www.mdpi.com/1424-8220/22/24/9690. Number: 24 Publisher: Multidisciplinary Digital Publishing Institute.

Charmi Jobanputra, Jatna Bavishi, and Nishant Doshi. Human Activity Recognition: A Survey. *Procedia Computer Science*, 155:698–703, January 2019. ISSN 1877-0509. doi: 10.1016/j.procs.2019.08.100. URL https://www.sciencedirect.com/science/article/pii/S1877050919310166.

A. Jovic, K. Brkic, and N. Bogunovic. A review of feature selection methods with applications. In *2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 1200–1205, Opatija, Croatia, May 2015. IEEE. ISBN 978-953-233-082-3. doi: 10.1109/MIPRO.2015.7160458. URL http://ieeexplore.ieee.org/document/7160458/.

Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization, January 2017. URL http://arxiv.org/abs/1412.6980. arXiv:1412.6980 [cs].

Jan Kuschan, Moritz Burgdorff, Hristo Filaretov, and Jörg Krüger. Inertial Measurement Unit based Human Action Recognition for Soft-Robotic Exoskeleton. *IOP Conference Series: Materials Science and Engineering*, 1140(1):012020, May 2021. ISSN 1757-899X. doi: 10.1088/1757-899X/1140/1/012020. URL https://dx.doi.org/10.1088/1757-899X/1140/1/012020. Publisher: IOP Publishing.

Alex Labach, Hojjat Salehinejad, and Shahrokh Valaee. Survey of Dropout Methods for Deep Neural Networks, October 2019. URL http://arxiv.org/abs/1904.13310. arXiv:1904.13310 [cs].

Yann LeCun, Yoshua Bengio, and T Bell Laboratories. Convolutional Networks for Images, Speech, and Time-Series. In *The Handbook of Brain Theory and Neural Networks*, pages 255–258. MIT Press, Cambridge, MA, USA, October 1998. ISBN 0-262-51102-9.

Moshe Leshno. Multilayer Feedforward Networks with a Non-Polynomial Activation Function Can Approximate Any Function. *Neural Networks*, 6 (6):861–867, 1993. ISSN 0893-6080. doi: https://doi.org/10.1016/S0893-6080(05)80131-5. URL https://www.sciencedirect.com/science/article/pii/S0893608005801315.

Ashwin Shenoy M and N. Thillaiarasu. A Survey on Different Computer Vision Based Human Activity Recognition for Surveillance Applications. In *2022 6th International Conference on Computing Methodologies and Communication (ICCMC)*, pages 1372–1376, Erode, India, March 2022. IEEE. ISBN 978-1-66541-028-1. doi: 10.1109/ICCMC53470.2022.9753931. URL https://ieeexplore.ieee.org/document/9753931/.

Dominic Masters and Carlo Luschi. Revisiting Small Batch Training for Deep Neural Networks, April 2018. URL http://arxiv.org/abs/1804.07612. arXiv:1804.07612 [cs, stat].

L. Minh Dang, Kyungbok Min, Hanxiang Wang, Md. Jalil Piran, Cheol Hee Lee, and Hyeonjoon Moon. Sensor-based and vision-based human activity recognition: A comprehensive survey. *Pattern Recognition*, 108:107561, December 2020. ISSN 00313203. doi: 10.1016/j.patcog.2020.107561. URL `https://linkinghub.elsevier.com/retrieve/pii/S0031320320303642`.

Francesco Missiroli, Nicola Lotti, Enrica Tricomi, Casimir Bokranz, Ryan Alicea, Michele Xiloyannis, Jens Krzywinski, Simona Crea, Nicola Vitiello, and Lorenzo Masia. Rigid, Soft, Passive, and Active: A Hybrid Occupational Exoskeleton for Bimanual Multijoint Assistance. *IEEE Robotics and Automation Letters*, 7 (2):2557–2564, April 2022. ISSN 2377-3766, 2377-3774. doi: 10.1109/LRA.2022.3142447. URL `https://ieeexplore.ieee.org/document/9681209/`.

Tom M. Mitchell. *Machine Learning*. McGraw-Hill series in computer science. McGraw-Hill, New York, 1997. ISBN 978-0-07-042807-2.

Roweida Mohammed, Jumanah Rawashdeh, and Malak Abdullah. Machine Learning with Oversampling and Undersampling Techniques: Overview Study and Experimental Results. In *2020 11th International Conference on Information and Communication Systems (ICICS)*, pages 243–248, Irbid, Jordan, April 2020. IEEE. ISBN 978-1-72816-227-0. doi: 10.1109/ICICS49469.2020.239556. URL `https://ieeexplore.ieee.org/document/9078901/`.

Abdulmajid Murad and Jae-Young Pyun. Deep Recurrent Neural Networks for Human Activity Recognition. *Sensors*, 17(11):2556, November 2017. ISSN 1424-8220. doi: 10.3390/s17112556. URL `https://www.mdpi.com/1424-8220/17/11/2556`. Number: 11 Publisher: Multidisciplinary Digital Publishing Institute.

Najme Zehra Naqvi. Step Counting Using Smartphone-Based Accelerometer. *International Journal on Computer Science and Engineering*, 4(05), 2012.

Michael Nielsen. Neural Networks and Deep Learning, 2015.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. *CoRR*, 2019. doi: https://doi.org/10.48550/arXiv.1912.01703. URL `http://arxiv.org/abs/1912.01703`.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12(85):2825–2830, 2011. ISSN 1533-7928. URL `http://jmlr.org/papers/v12/pedregosa11a.html`.

Pei-Chun Lin, Jau-Ching Lu, Chia-Hung Tsai, and Chi-Wei Ho. Design and Implementation of a Nine-Axis Inertial Measurement Unit. *IEEE/ASME Transactions on Mechatronics*, 17(4):657–668, August 2012. ISSN 1083-4435, 1941-014X. doi: 10.1109/TMECH.2011.2111378. URL `http://ieeexplore.ieee.org/document/5734855/`.

Mattia Pesenti, Giovanni Invernizzi, Julie Mazzella, Marco Bocciolone, Alessandra Pedrocchi, and Marta Gandolla. IMU-based human activity recognition and payload classification for low-back exoskeletons. *Scientific Reports*, 13(1): 1184, January 2023. ISSN 2045-2322. doi: 10.1038/s41598-023-28195-x. URL `https://www.nature.com/articles/s41598-023-28195-x`. Number: 1 Publisher: Nature Publishing Group.

Tommaso Poliero, Stefano Toxiri, Sara Anastasi, Luigi Monica, Darwin G. Caldwell, and Jesús Ortiz. Assessment of an On-board Classifier for Activity Recognition on an Active Back-Support Exoskeleton. In *2019 IEEE 16th International Conference on Rehabilitation Robotics (ICORR)*, pages 559–564, June 2019. doi: 10.1109/ICORR.2019.8779519. ISSN: 1945-7901.

Xingye Qiao and Lingsong Zhang. Distance-weighted Support Vector Machine, October 2015. URL `http://arxiv.org/abs/1310.3003`. arXiv:1310.3003 [stat].

Suneth Ranasinghe, Fadi Al Machot, and Heinrich C Mayr. A review on applications of activity recognition systems with regard to performance and evaluation. *International Journal of Distributed Sensor Networks*, 12(8):1550147716665520, August 2016. ISSN 1550-1329. doi: 10.1177/1550147716665520. URL `https://doi.org/10.1177/1550147716665520`. Publisher: SAGE Publications.

Rui Zhang, Fabian Hoflinger, and Leonhard M. Reind. Calibration of an IMU Using 3-D Rotation Platform. *IEEE Sensors Journal*, 14(6):1778–1787, June 2014. ISSN 1530-437X, 1558-1748. doi: 10.1109/JSEN.2014.2303642. URL `http://ieeexplore.ieee.org/document/6728637/`.

Stuart J. Russell, Peter Norvig, Ernest Davis, and Douglas Edwards. *Artificial intelligence: a modern approach*. Prentice Hall series in artificial intelligence. Pearson, Boston Columbus Indianapolis New York San Francisco Upper Saddle River Amsterdam Cape Town Dubai London Madrid Milan Munich Paris Montreal Toronto Delhi Mexico City Sao Paulo Sydney Hong Kong Seoul Singapore Taipei Tokyo, third edition, global edition edition, 2016. ISBN 978-1-292-15396-4.

Armin Schmidt. German-Bionic-5th-Generation-1-scaled.jpg, 2021a. URL `https://germanbionic.com/wp-content/uploads/2021/12/German-Bionic-5th-Generation-1-scaled.jpg`.

Armin Schmidt. German Bionic launches 5th generation Cray X, smart AI-powered exoskeleton, December 2021b. URL `https://germanbionic.com/en/german-bionic-launches-5th-generation-cray-x-smart-ai-powered-exoskeleton/`.

Elke Schneider, Xabier Irastorza, and Sarah Copsey. *OSH in figures: work-related musculoskeletal disorders in the EU - Facts and figures*. European risk observatory report. Office for Official Publ. of the Europ. Communities, Luxembourg, April 2010. ISBN 978-92-9191-261-2. doi: 10.2802/10952.

Yoli Shavit and Itzik Klein. Boosting Inertial-Based Human Activity Recognition With Transformers. *IEEE Access*, 9:53540–53547, 2021. ISSN 2169-3536. doi: 10.1109/ACCESS.2021.3070646. Conference Name: IEEE Access.

Jozsef Suto, Stefan Oniga, and Petrica Pop Sitar. Feature Analysis to Human Activity Recognition. *INTERNATIONAL JOURNAL OF COMPUTERS COMMUNICATIONS & CONTROL*, 12(1):116–130, 2017. ISSN 1841-9844. URL `https://www.univagora.ro/jour/index.php/ijccc/article/view/2787`. Number: 1.

Noelia Sánchez-Maroño, Amparo Alonso-Betanzos, and María Tombilla-Sanromán. Filter Methods for Feature Selection – A Comparative Study. In Hujun Yin, Peter Tino, Emilio Corchado, Will Byrne, and Xin Yao, editors, *Intelligent Data Engineering and Automated Learning - IDEAL 2007*, volume 4881, pages 178–187. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. ISBN 978-3-540-77225-5. doi: 10.1007/978-3-540-77226-2_19. URL `http://link.springer.com/10.1007/978-3-540-77226-2_19`. Series Title: Lecture Notes in Computer Science.

Luca Tiseni, Michele Xiloyannis, Domenico Chiaradia, Nicola Lotti, Massimiliano Solazzi, Herman van der Kooij, Antonio Frisoli, and Lorenzo Masia. On the edge between soft and rigid: an assistive shoulder exoskeleton with hyper-redundant kinematics. In *2019 IEEE 16th International Conference on Rehabilitation Robotics (ICORR)*, pages 618–624, Toronto, ON, Canada, June 2019. IEEE. ISBN 978-1-72812-755-2. doi: 10.1109/ICORR.2019.8779546. URL `https://ieeexplore.ieee.org/document/8779546/`.

Stefano Toxiri, Matthias B. Näf, Maria Lazzaroni, Jorge Fernández, Matteo Sposito, Tommaso Poliero, Luigi Monica, Sara Anastasi, Darwin G. Caldwell, and Jesús Ortiz. Back-Support Exoskeletons for Occupational Use: An Overview of Technological Advances and Trends. *IISE Transactions on Occupational Ergonomics and Human Factors*, 7(3-4):237–249, October 2019. ISSN 2472-5838. doi: 10.1080/24725838.2019.1626303. URL `https://doi.org/10.1080/24725838.2019.1626303`. Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/24725838.2019.1626303.

E. Valarezo, P. Rivera, J. M. Park, G. Gi, T. Y. Kim, M. A. Al-Antari, M. Al-Masni, and T.-S. Kim. Human Activity Recognition Using a Single Wrist IMU Sensor via Deep Learning Convolutional and Recurrent Neural Nets. *Journal of ICT, Design, Engineering and Technological Science*, pages 1–5, June 2017. ISSN 2604-2673. doi: 10.33150/JITDETS-1.1.1. URL `https://jitdets.com/ojs/index.php/jitdets/article/view/30`.

Praneeth Vepakomma, Debraj De, Sajal K. Das, and Shekhar Bhansali. A-Wristocracy: Deep learning on wrist-worn sensing for recognition of user complex activities. In *2015 IEEE 12th International Conference on Wearable and Implantable Body Sensor Networks (BSN)*, pages 1–6, June 2015. doi: 10.1109/BSN.2015.7299406. ISSN: 2376-8894.

Michalis Vrigkas, Christophoros Nikou, and Ioannis A. Kakadiaris. A Review of Human Activity Recognition Methods. *Frontiers in Robotics and AI*, 2, 2015. ISSN 2296-9144. URL `https://www.frontiersin.org/articles/10.3389/frobt.2015.00028`.

Jindong Wang, Yiqiang Chen, Shuji Hao, Xiaohui Peng, and Lisha Hu. Deep Learning for Sensor-based Activity Recognition: A Survey. *Pattern Recognition Letters*, 119:3–11, March 2019a. ISSN 01678655. doi: 10.1016/j.patrec.2018.02.010. URL `http://arxiv.org/abs/1707.03502`. arXiv:1707.03502 [cs].

Yan Wang, Shuang Cang, and Hongnian Yu. Survey on wearable sensor modality centred human activity recognition in health care. *Expert Systems with Applications*, 137:167–190, 2019b. doi: https://doi.org/10.1016/j.eswa.2019.04.057. URL `https://www.sciencedirect.com/science/article/pii/S0957417419302878`.

Zeyu Yin, Jianbin Zheng, Liping Huang, Yifan Gao, Huihui Peng, and Linghan Yin. SA-SVM-Based Locomotion Pattern Recognition for Exoskeleton Robot. *Applied Sciences*, 11(12):5573, January 2021. ISSN 2076-3417. doi: 10.3390/app11125573. URL `https://www.mdpi.com/2076-3417/11/12/5573`. Number: 12 Publisher: Multidisciplinary Digital Publishing Institute.

Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. Dive into Deep Learning, February 2023. URL `http://arxiv.org/abs/2106.11342`. arXiv:2106.11342 [cs].

A Appendix

A.1 Code Availability

The code which produced the results for this study is made freely available under the MIT license. It is available for usage under the respective terms at `https://github.com/CZ438FE/HAR_DL_exoskeleton.git`.

This repository contains the functionalities to process labeled raw data, train models on this processed data and evaluate the resulting models. A detailed explanation of this repository is given in the README.md in the parent directory, as well as in the respective READMEs for the various functionalities of the repository.

Some functionalities within this repository expect the existence of an anonymization_file.json, which is not contained within this repository. This file contains company secrets owned by German Bionic and can therefore not be made publicly available. Care has been taken to reduce the dependency on this file as far as possible. The repository and included functions were constructed to enable quick understanding and modification of the code. Therefore most of this repository could be used for similar research after minor adjustments.

The code used for this study was written by the author of this study with the exception of the functions used for the creation of the features, i.e. all the functions within the feature_utils.py file and the respective tests in feature_utils_test.py.

The repository was created on the Linux/GNU distribution Ubuntu, therefore some functionalities are adapted to the folder structure of this operating system. When bugs appear under Windows-based operating systems, checking the correct handling of the file and folder structure might be a starting point. Making this repository robust to different operating systems was beyond the scope of this study.

A.2 Illustration of Resampling

The following entails a visualization of the process of resampling, as applied within this study. As the sensor readings are not evenly spaced and may occur with differing frequencies, resampling the data is needed to ensure that the data has the same format for all further processing. To illustrate this process, the following displays the performed steps on an example containing two features. In this example a period of five milliseconds (ms) is to be aggregated as a new resampled datapoint.

The raw data may contain gaps between sensor readings, illustrated in Table 12. As not every millisecond from zero to four contains sensor readings, there are gaps to be filled before resampling. This can be understood as solving a missing data problem, because the two features are unobserved for the times one and three.

Table 12: Illustration Resampling: Raw Sensor data

Time in ms	Feature 1	Feature 2
0	0.0	4.8
2	1.3	3.7
4	1.9	4.2

There are several possibilities to impute these missing values for unobserved points in time. For this study it was chosen to impute via forward filling, i.e. imputing by inserting the last observed value for all unobserved points in time. Table 13 contains the same data as Table 12 after imputing the unobserved values via forward filling. The period to be aggregated has a length of five milliseconds.

Table 13: Illustration Resampling: Raw Sensor data after forward filling

Time in ms	Feature 1	Feature 2
0	0.0	4.8
1	0.0	4.8
2	1.3	3.7
3	1.3	3.7
4	1.9	4.2

These filled sensor values are aggregated via averaging. Table 14 contains the aggregated values of the data seen in Table 13. The aggregated values were created by averaging column-wise.

Table 14: Illustration Resampling: Aggregated Sensor Data

Time in ms	Feature 1	Feature 2
2	0.9	4.24

A.3 Protocol of Recording a Testfile

A protocol, which was used for the recording of validation data is shown in Table 15. Before each task the subject rested for at least 10 seconds. The given duration was the intended duration, but as the subjects were allowed to perform the tasks in a manner that is natural to them, the time needed to perform the task deviated in practice, e.g. by walking faster than expected.

Table 15: Protocol for recording

Task	Manipulated weight	Duration
Lifting weights to the height of the hip	10 kg	30 seconds
Set weights down from the height of the hip	10 kg	30 seconds
Pulling a cart	25 kg	20 seconds
Pushing a cart	25 kg	20 seconds
Sitting on chair one	None	30 seconds
Set weights down from shelve to floor	5-20 kg	20 seconds
Lifting weights from floor into shelve	5-20 kg	20 seconds
Holding weight in front of the body	10 kg	40 seconds
Sitting on chair two	None	30 seconds
Walking on flat ground towards staircase	None	30 seconds
Going up a staircase two floors	None	40 seconds
Going down a staircase two floors	None	40 seconds

Declaration of Authorship

Ich erkläre hiermit gemäß § 9 Abs. 12 APO, dass ich die vorstehende Abschlussarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Des Weiteren erkläre ich, dass die digitale Fassung der gedruckten Ausfertigung der Abschlussarbeit ausnahmslos in Inhalt und Wortlaut entspricht und zur Kenntnis genommen wurde, dass diese digitale Fassung einer durch Software unterstützten, anonymisierten Prüfung auf Plagiate unterzogen werden kann.

Place, Date

Signature