Equipment data-based activity recognition of construction machinery

Maschinendatenbasierte Aktivitätserkennung von Baumaschinen

Bachelor's thesis

TUM Department of Mechanical Engineering
Technical University of Munich

Topic assigned by	Prof. Dr.-Ing. Johannes Fottner
	Chair of Materials Handling, Material Flow, Logistics
Supervisor	M.Sc. Anne Fischer
Submitted by:	Alexandre Beiderwellen Bedrikow Kohlbrennerstr. 16 81929, München
Submitted on:	15.06.2021 in Garching
Inventory no. fml:	2020/118

Preface

This thesis was written under the scientific and content guidance of **Anne Fischer**, M. Sc., research assistant at the Chair of Materials Handling Material Flow Logistics (fml) at the Technical University of Munich.

Copyright agreement

Garching, 15.06.2021

Contents

Table of abbreviations

Abbreviation	Meaning
AI	Artificial Intelligence
ANN	Artificial Neural Network
BiLSTM	Bidirectional long-short term memory
CNN	Convolutional Neural Network
CM	Casing machine
DES	Discrete event simulation
DNN	Deep Neural Network
FNOW	Fully non-overlapping window
HAR	Human Activity Recognition
IMU	Inertial measurement unit
IQR	Interquartile range
LoD	Level of Detail
LSTM	Long-short term memory
MLP	Multilayer perceptron
MTS	Multivariate time series
OW	Overlapping window
RNN	Recurrent neural network
SNOW	Semi non-overlapping window
TLU	Threshold logic unit

TS	Training set
TSC	Time Series Classification
TTS	Test set
TUM	Technische Universität München
VS	Validation set

Table of Symbols

Symbol	Unit	Description
$\mathbf{W}$	[-]	Weight matrix
$\mathbf{y}$	[-]	Model output
$\mathbf{x}$	[-]	Model input
$\mathbf{b}$	[-]	Bias vector
ϕ	[-]	Activation function
$\mathbf{i}$	[-]	LSTM Input gate
$\mathbf{o}$	[-]	LSTM Output gate
$\mathbf{f}$	[-]	LSTM Forget gate
$\mathbf{c}$	[-]	LSTM Long term state
$\mathbf{h}$	[-]	LSTM Short term state
p	[-]	Output probabilities of a softmax layer
TP	[-]	True positive
TN	[-]	True negative
FP	[-]	False positive
FN	[-]	False negative
P	[-]	Positive
N	[-]	Negative

Part I

Introduction

1 Motivation

1.1 Motivation

With more than 800 000 employees and an projected annual revenue of more than 330 billion Euro in 2021, the construction industry represents a large share of the economic output in Germany [Sta-2021, Sta-2020]. This phenomenon is not only restricted to Germany, but can be found all over the globe. Although the industry plays an essential role economically and socially, it faces a number of challenges. As is the case for the industry in general, according to a survey of leading companies in the German construction industry, staff shortages and increasing competitiveness and price pressure are the two biggest challenges facing the sector [PwC-2013]. But unlike the overall industry, the construction sector has seen only a low rate of productivity growth in recent years. The labor productivity growth rate of the German construction industry grew by an average of only 0.12 % between the years 1998 and 2015, which represents only about 10 % of the average development of the overall economy [Ber-2019].

Another challenge is the way in which the construction industry addresses the growing demands for environmentally friendly practices, as the emissions generated by the construction industry are not insignificant. The main source of these emissions is fuel consumption by construction equipment [U.S-2008]. These emissions are especially critical during downtimes, during which the machines are kept in an idle state. Despite the fact that no progress is being accomplished in the construction project, the construction equipment continues to emit pollutants [Lew-2011].

The two challenges described above give rise to the need to reduce the downtime of construction machinery as far as possible and to maximize operational efficiency. The starting point for this optimization is the precise assessment of the actual operating condition by recording the activities of the machines. Based on the activity data of the machines, various approaches can be pursued to achieve the mentioned goals. For instance, discrete event simulation (DES) can be used to predict the progress of the construction project in order to formulate recommendations for the next steps and ultimately increase operational efficiency [Fis-2020]. In addition to that, an estimation of the emission can be provided

based on the detailed knowledge of the performed activities [Ahn-2015, She-2020].

Although different methods are available to increase productivity, the main problem lies in obtaining the activity data. In practice, the activities of the machines are still mostly recorded manually, which leads to a number of implications and problems [Gol-2013]. The first problem is the time and manpower required for this task, which makes manual labeling very time consuming and costly. Furthermore, manual recording is also inaccurate, prone to errors and poorly reproducible, as it partly depends on the perception of the operator [Lan-2021, Gol-2013]. In order to circumvent these difficulties, automatic activity recognition presents itself as a suitable solution. With the help of different methods, the activities should be detected as far as possible without the need for human labeling.

In order to be able to overcome the explained challenges, possibilities for increasing the efficiency and productivity of construction processes through networking and communication of mobile machines are being investigated as part of the research project "Building 4.0" of the Chair of Materials Handling, Material Flow and Logistics (TUM). Among other topics, the tracking and tracing of objects within the construction site, the simulation of construction processes and modeling using Building Information Modeling (BIM) are being investigated. [Tec-2021]

Another examined topic within the project is the possibility to use available data of the construction equipment to infer about the performed activities. The aim of this work is to extend these investigations and to gain further knowledge of the extent to which conclusions about activity can be drawn from the machine data.

1.2 Structure

This thesis is basically divided into 3 parts. The first part includes Chapters 1 to 3. In Chapter 1, the motivation for the thesis has been outlined. In Chapter 2, the current state of the art and the existing approaches for automatic activity recognition of construction machines are presented. Hereby, the limitations and issues of the approaches or the available room for improvement are discussed. Based on these findings, the research objectives for this

thesis are formulated in Sec. 2.2. Chapter 3 covers the theoretical background. In particular, the drilling process with the Kelly drilling method, which is used as a use case in the following, is explained. Furthermore, the foundations of artificial intelligence and deep learning are outlined.

In the second part, which includes Chapters 4 to 6, the research methodology is described. The structure of this part of the thesis follows the common approach to machine learning projects according to CRISP-DM [Stu-2021, Cha-2000]. At the beginning of each chapter, an excerpt from [Stu-2021] is provided within a gray box, which briefly explains each step. However, this is not the main focus of the thesis and serves only as an orientation. Chapters 4 and 5 deal with the acquisition, analysis and preprocessing of the data. In Chapter 6, the models examined in the further work are introduced and explained.

In Part 3, the results of the investigations are presented and discussed. In Chapter 8 possible approach points for further future investigations are outlined. Chapter 9 provides a summary of the thesis.

2 Previous work

2.1 Activity recognition of construction equipment

Automatic activity detection of construction machines can be essentially divided into three groups according to the nature of the data: vision-based methods (Sec. 2.1.1), audio-based methods (Sec. 2.1.2) and kinematics- or motion-based methods (Sec. 2.1.3) [She-2020].

2.1.1 Vision-based methods

With the proliferation of low-cost video cameras with high-resolution and the simplification of storing and transmitting large data sets, the possibility of using vision-based methods for activity detection is also becoming increasingly popular [Gol-2013]. Cameras are placed at the construction site and record the movements of construction equipment. The methods used to analyze the recordings range from pure image processing methods to complex deep learning architectures.

On the side of simpler methods, Zou and Kim use the hue,saturation and value of recordings to determine the downtime of an excavator [Zou-2007]. Furthermore, Golparvar-Fard et al. explored techniques to detect five different activities of excavators and 3 activities of dump trucks [Gol-2013]. For this purpose, recordings from 10 cameras at different orientations from five different construction projects during six months were used as data to train a Support Vector Machine. Accuracies of up to 86.33 % were achieved for the excavator and 98.33 % for the dump truck. Furthermore, the models proved to be robust against small movements of the camera. Gong et al. employ a bag-of-video-features approach to distinguish between the swing, excavating, and relocating of a backhoe [Gon-2011]. The features are extracted using a 3D Harris detector and then represented using a local histogram. The activity detection is performed using a Bayersian network model. The results of the model range from 73.6 % to 79 %. A further approach using Deep Learning methods was developed by Kim and Chi and mainly consists of three steps [Kim-2019]. The first two steps are responsible for identifying and tracking the excavators in the recordings of the construction site. In the third step, the actual activity detection takes place. The models

are based on a combination of convolutional and recurrent neural networks and achieve a precision of up to 90.9 % for the six examined activities.

2.1.2 Audio-based methods

Audio-based methods use sound recordings of the machine at the construction site to obtain inferences about the activity performed. The whole approach is based on the assumption that the machines emit different sound patterns and levels when performing different activities [Lan-2021]. Since these approaches are the least similar with the goals pursued here, these methods will not be discussed further in detail.

2.1.3 Motion-based methods

Motion-based methods rely on sensor data from accelerometers, gyroscopes or IMUs [Lan-2021]. As is the case with cameras, motion sensors have become popular in recent years and are already present in most smartphones, for example. In contrast to vision and audio based methods, where the cameras or microphones are often located outside the area of operation, the sensors for motion based methods have to be attached directly to the machines under investigation, e.g. to the excavator arm or the cabin of an excavator.

Akhavian and Behzadan [Akh-2015] placed smartphones with accelerometers and gyroscopes inside the cabin of a front-end loader to measure the vibrations of the machine. The approach is based on the assumption that different activities also induce different vibrations. Statistical (e.g. mean, variance, interquartile range, etc.) as well as frequency domain (signal energy) features are extracted from the measured vibrations. A total of 42 features were available for analysis. Logistic Regression, k-Nearest Neighbour (KNN), Decision Tree (DT), Support Vector Machines (SVM) and Neural Networks (ANN) were investigated as methods for the classification of the activities on 3 different levels of detail (LoD). The most detailed LoD consisted of the activities "Engine Off", "Idle", "Scooping", "Moving" and "Dumping". Among the best results, accuracies of up to 86.09 % were obtained using neural networks for the five activities.

Rashid and Louis [Ras-2020] use data from an IMU to predict the activities of an excavator. However, instead of relying on the vibrations of the machine, they investigated the possibility

of using activity-specific motions of the machine to detect its activity. The reason for this, according to the authors, is the fact that vibrations can be highly machine-dependent and may also be influenced by external forces. The sensors were attached to the bucket, stick and boom of the excavator. As was the case with Akhavian and Behzadan [Akh-2015], different LoD with up to nine activities were analyzed. The maximum accuracies range from 100 % for two activities to 92.1 % for 9 activities among the four machine learning methods investigated (DT, SVM, ANN, and KNN).

Since most machine learning methods require a large amount of data to deliver reliable results, obtaining the data to train the models is a major problem. This problem only increases when the application of deep learning methods is considered. To address these issues, Rashid and Louis [Ras-2019] also investigated the effects of using time-series data augmentation methods (jittering, scaling, rotation, and time-warping). The main deep learning methods used were ANN and recurrent neural networks for the activity detection of an excavator and a front-end loader. The advantage of using deep learning methods lies in the fact that high-level features can be extracted automatically from the data. For the excavator, accuracies of 62.2 % (ANN) and 63.3 % without data augmentation were achieved, which increased up to 97.9 % (LSTM) when using the augmented data set. For the front loader, the results increased from 59.6 % (LSTM) to up to 96.7 % (LSTM).

Slaton et al. [Sla-2020] use a hybrid approach consisting of convolutional layers and recurrent layers for the activity detection of a roller compactor (six activities) and an excavator (seven activities). The validation accuracy for the compactor amounts to 77.1 % for all six activities and 96.2 % when only the direction is considered. For the excavator, the accuracy is as high as 90.7 %.

2.2 Research gaps and objectives

Each of the explained approaches presents its own advantages and disadvantages. Although very good results have been achieved with the vision based methods and its relatively simple setup, the generalization of the results to new cases can be challenging. Since the methods are based on visual characteristics, a large data set with different machine types and perspectives is required. Furthermore, poor lighting conditions, coverage

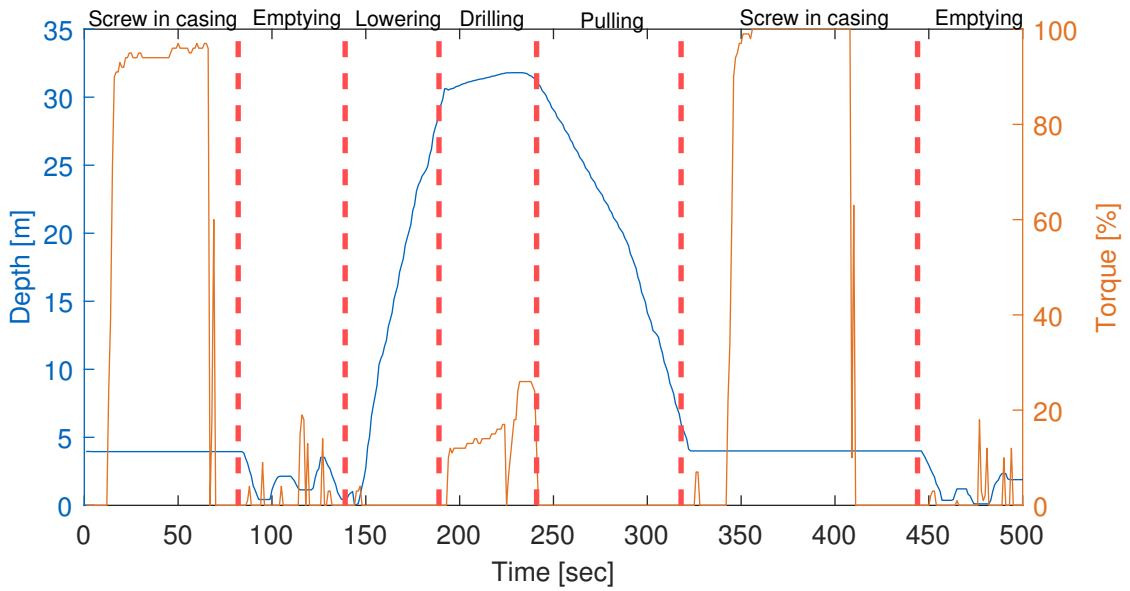


Figure 2-1: Measured depth and torque during drilling with a Kelly drilling rig

of the camera by other machines or weather conditions can also negatively influence the results. Regarding the audio based methods, mainly the difficulty to detect multiple machines simultaneously and the less detailed LoD compared to other methods can be mentioned. [Lan-2021]

Furthermore, it is important to highlight that although a large amount of construction equipment exists and is employed, the existing approaches primarily focus on only a small set of these machines, mainly front-end loaders [Akh-2015, Ras-2019], hydraulic excavators [Gol-2013, Ras-2020, Ras-2019, Sla-2020], and compactors [Sla-2020]. Industry-specific aspects of other sectors such as special civil engineering are not considered. Many machines used in special civil engineering do not allow the attachment of the required sensors (accelerometers, gyroscopes, IMUs, etc.) due to the high forces involved or the poorly accessible parts of the machine, which makes the application of the mentioned motion-based approaches difficult [Fis-2021]. In addition, another issue arises from the fact that a major portion of the processes in special civil engineering takes place below surface and is not visible, thus not allowing the use of vision based methods.

These requirements of special civil engineering demand a different approach to automatic activity recognition. A possible solution for addressing this was developed by Fischer et al. as part of the Building 4.0 project [Fis-2021]. The underlying principle is the machine data already available from the machine. Many machines used in special civil engineering, such

as Kelly drilling rigs, already include a series of sensors which already measure several signals such as depth, pump pressure, torque or forces, eliminating the necessity of attaching additional sensors [Fis-2020]. However, the data from these sensors are rarely used in practice, and may mask great potential as a basis for further knowledge about the process flow. In the remainder of this thesis, these data are referred to as "telematics data". The measured telematics data during a drilling operation are shown in Figure 2-1.

First investigations of the extent to which inferences about the performed activities can be drawn from these available machine data have been conducted by Fischer et al. [Fis-2021]. For the detection of the activities "Emptying", "Lowering", "Drilling", "Pulling" and "Drilling in casing" the use of Decision Trees, Logistic Regression, Support Vector Machine, Naive Bayes and Artificial Neural Networks was explored. The models achieved high accuracies of up to 95 %. Although good results were achieved, a possible generalization problem can be identified, since strong outlier filtering was performed during the preprocessing of the data and a large number of data points (up to one fifth of the data set) were removed. Since it cannot be excluded that the removed outliers are to be expected during a deployment in practice, it is not known how well the models would respond to a deployment in practice.

Based on the outlined aspects, the following points can be formulated as research objectives:

1. Extend investigations of activity recognition using telematics data.
2. Explore the effectiveness of existing motion-based methods for activity recognition based on telematics data.
3. Investigate models which require little preprocessing, are robust to outliers and sensor noise, and can detect many activities.
4. Determine influence of adopted labels on activity recognition.

3 Theoretical background

3.1 Kelly drilling method

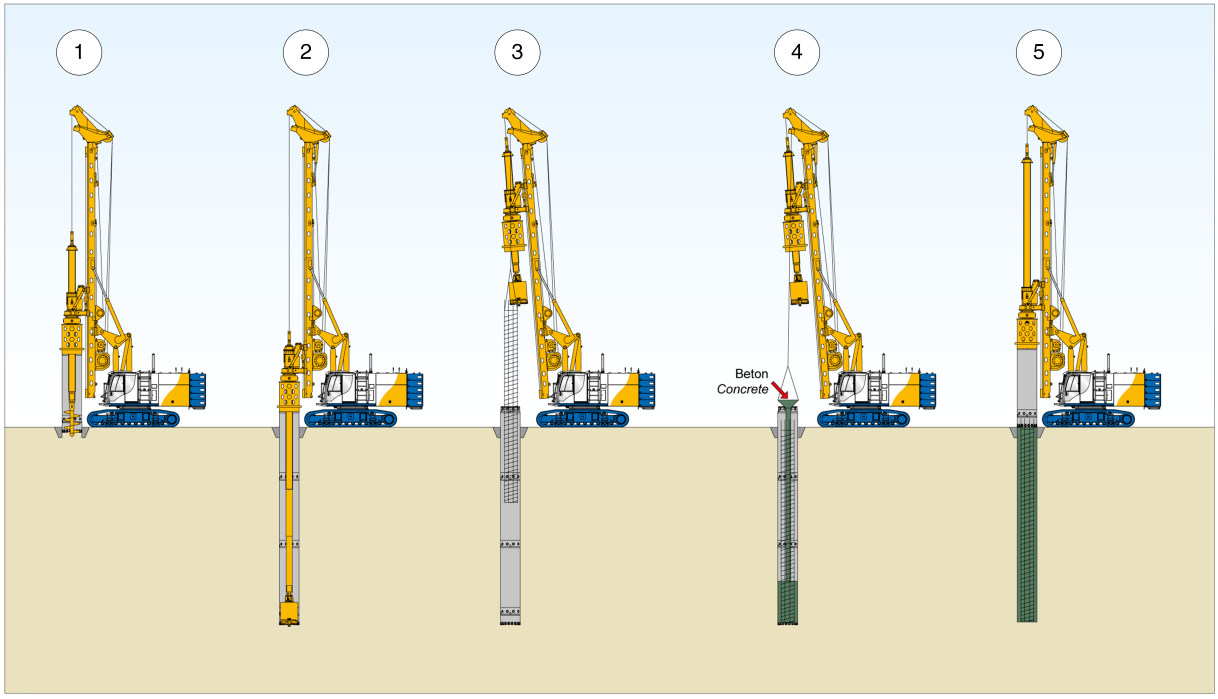


Figure 3-1: Process steps of the Kelly drilling method: (1) Screw in casing; (2) Drilling; (3) Reinforcing; (4) Concreting; (5) Remove casing [Bau-2021]

In this section, the construction of a pile using the Kelly drilling method is briefly explained. Since the focus of the work is on the activity detection itself, the section serves only as an orientation and looks at the procedure only superficially. The production process of a bored pile by means of the Kelly drilling method can be fundamentally divided into 5 steps. These are shown graphically in Figure 3-1. The informations of the following paragraphs are taken from [May-2011].

The first step consists of **screwing in the casing (1)**. A standpipe is placed over the planned borehole and the casing is screwed into the ground. If the drilling machine cannot provide the required torque or force, then a casing machine can be employed. In the second step, the actual **drilling (2)** takes place. This step can further be divided into four steps. First, the drilling tool is lowered until the bottom of the hole is reached. Then the actual drilling takes place until the drill box is entirely filled. Subsequently, the drilling tool is pulled up to the surface. Before the cycle is repeated, the drilling tool is emptied. When the

desired hole depth is reached, the drilling step ends. The next step involves **installing the reinforcement (3)**. This operation can be performed either directly by the drilling machine or optionally by an additional crane. In the fourth step, the **concreting (4)** stage takes place. For this purpose, pouring tubes are inserted into the drilled hole and the concrete is poured in. When the desired amount of concrete has been added, the pouring tube is removed again. In the fifth and last step, the **casing and the standpipe are removed (5)** from the ground.

In addition to the above-mentioned process steps with their respective sub-processes, other activities can also be performed. However, these do not belong directly to the process and are therefore considered either as secondary processes or downtime processes. Examples of such activities are, for example, refilling water, changing tools, refueling, break times or waiting for concrete.

3.2 Deep Learning

The term Artificial Intelligence (AI), formulated in the 1950s, refers to the "effort to automate intellectual tasks normally performed by humans" [Cho-2018]. According to Chollet, these approaches do not necessarily involve a so-called "learning" process and consisted at the beginning mainly of explicitly programmed rules (also called symbolic AI). For simpler problems an explicit formulation of the rules is still possible, but for more complex problems the formulation of the rules can represent a major challenge. To solve these problems, a change of paradigm in problem solving occured. Instead of the classical approach of explicitly programming the rules, the goal of the new paradigm is to determine these rules and learn a specific task. This new paradigm is referred to as Machine Learning. [Cho-2018]

Machine learning methods can be essentially divided into four main groups: supervised, unsupervised, semisupervised and reinforcement learning. The group of reinforcement learning methods is not relevant in the context of this study and will not be considered in detail. The fundamental principles of the first three variants are shown graphically in Figure 3-2. In supervised learning, the data set already contains the desired labels and the mapping of the inputs to the outputs is then searched. In unsupervised learning, the labels are not part of the data set and an attempt is made to extract knowledge, e.g. by clustering data

points into groups, from the data set based only on the inputs. The intersection between supervised and unsupervised learning is represented by semisupervised learning when the existing dataset only partially contains labels. [Gér-2019]

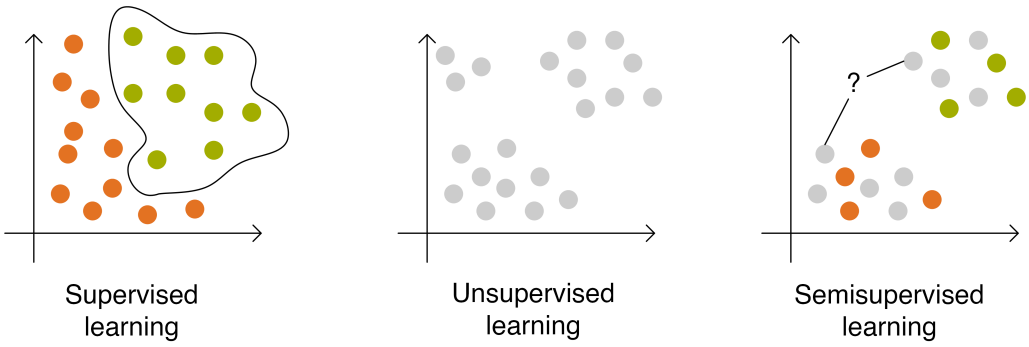


Figure 3-2: Types of Machine Learning problems, based on [Gér-2019]

As the use case under consideration focuses on a connection between sensor data and activity and uses real-world machine data to train a model, supervised learning methods will be discussed in more detail below.

The term supervised learning includes a series of methods, such as probabilistic modeling (e.g. Naive Bayes), which is based on statistical methods, kernel methods (e.g. Support Vector Machines), or decision trees. [Cho-2018]

Another subgroup of machine learning methods are deep learning approaches, which aim at formulate models with high levels of abstraction, which are characterized by their high complexity or their "depth" [Cho-2018, LeC-2015]. Deep learning also became well known because of the very good results achieved in the field of computer vision or machine translation. In the following, some specific types of Deep Learning methods are further explained.

3.2.1 Artificial Neural Networks

Perceptrons as a recreation of biological neurons

Artificial neural networks are machine learning models based on the structure of biological networks of neurons [Shr-2019, Gér-2019]. Biological neurons consist of a cell body, which aggregates several branches that are connected to other neurons. Connected to the cell body is also the axon, from which further branches then bind to several subsequent

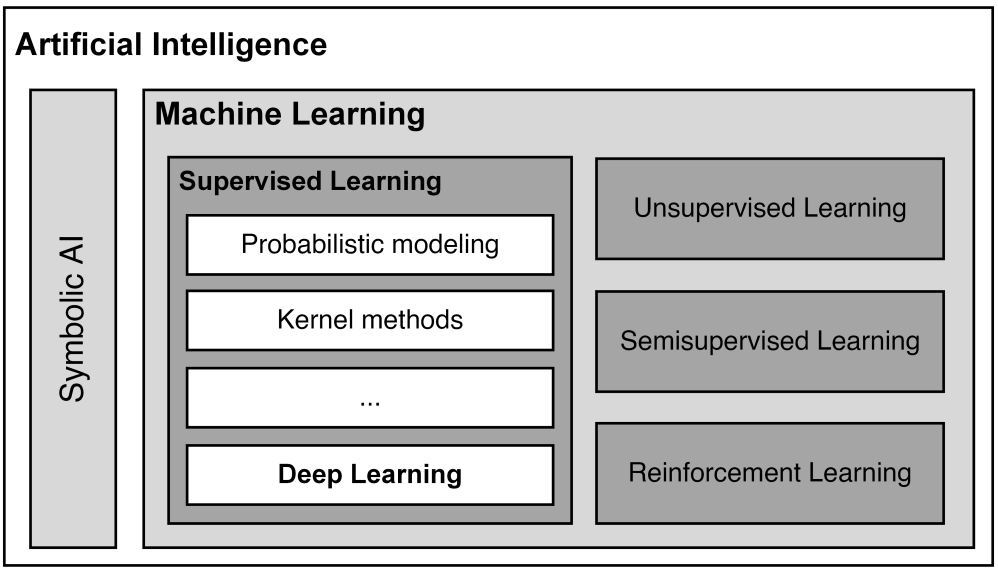


Figure 3-3: Overview of Artificial Intelligence and Machine Learning

neurons. The communication and transmission of neuroimpulses takes place through neurotransmitters. If sufficient neurotransmitters are provided within a short time by previous neurons, then the own neuron is "activated", so that further neurotransmitters are released at the synapses and the impulse is passed on. [Gér-2019]

Based on the structure of a single biological neuron, the threshold logic units (TLU) were developed (Fig. 3-4). The different inputs x_i of the TLU are multiplied with corresponding weights w_i and summed up to z (Eq. 3-1). Analogous to the biological variant, the unit is activated if z exceeds a certain threshold. For this purpose, the summed inputs are passed to the Heaviside function (see definition in Eq. 3-2) and the output h is potentially

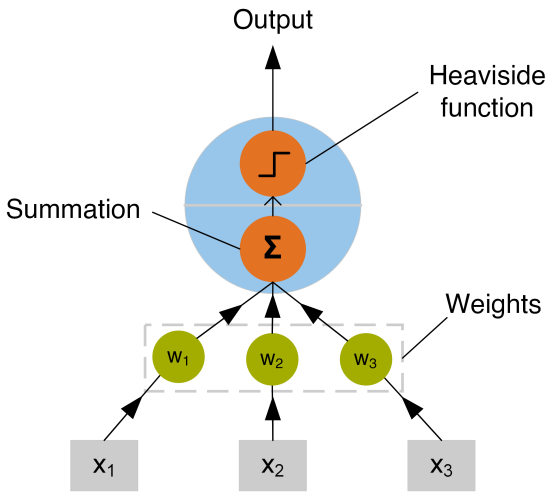


Figure 3-4: Schematic representation of a TLU

activated.

$$\begin{aligned} z &= \mathbf{x}^{\mathbf{T}}\mathbf{w} \\ h &= heaviside(z) \end{aligned} \tag{3-1}$$

$$heaviside(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases} \tag{3-2}$$

Since the Heaviside function can only take one of two possible values, TLUs are only useful for simple binary classification problems. An extension of the TLUs are the perceptrons, which consist of several juxtaposed TLUs. An additional bias unit, which invariably assumes the value 1, is added to the existing inputs. The relationships applicable to each individual TLU from Eq. 3-1 can be combined for a perceptron with m TLUs and n inputs to Eq. 3-3:

$$\mathbf{h}_{(\mathbf{W},\mathbf{b})}(\mathbf{X}) = heaviside(\mathbf{W}^{\mathbf{T}}\mathbf{x} + \mathbf{b}) \tag{3-3}$$

where $\mathbf{x} \in \mathbf{R}^n$ is the input vector, $\mathbf{W} \in \mathbf{R}^{n \times m}$ is the weight matrix, and $\mathbf{b} \in \mathbf{R}^m$ is the vector with weights for the bias terms

Perceptrons are also suitable for solving multiclass classification problems, since several outputs are available. The weights of the individual TLUs must be adjusted so that the desired output is produced for a given set of inputs. This process of adjusting the weights is called **training** and will be discussed in more detail later on. Although perceptrons can be used to differentiate between several classes, good results are limited to linearly separable problems. This is the case since the output of a perceptron consists only of a linear combination of the inputs and the application of the Heaviside function.

Dealing with complex problems using multilayer perceptrons

In order to solve more complex and nonlinear problems, it is advantageous to stack multiple perceptron layers. Such an architecture is called multilayer perceptron [Gér-2019]. The architecture can fundamentally be divided into three main components [Shr-2019]. The

input layer consists of the inputs of the model plus a bias term. The second to the previous layer are called hidden layers and consist of one or more perceptron layers. Finally, the last layer is called output layer. The number of hidden layers is also referred to as the "depth" of a model [Goo-2018]. If a model comprises multiple hidden layers, then it is called a Deep Neural Network (DNN). The architecture of a multilayer perceptron is shown schematically in Figure 3-5. As can be seen in the figure, the information flow takes place in one direction only. Due to this property, such networks are also called **Feedforward Neural Networks** [Goo-2018].

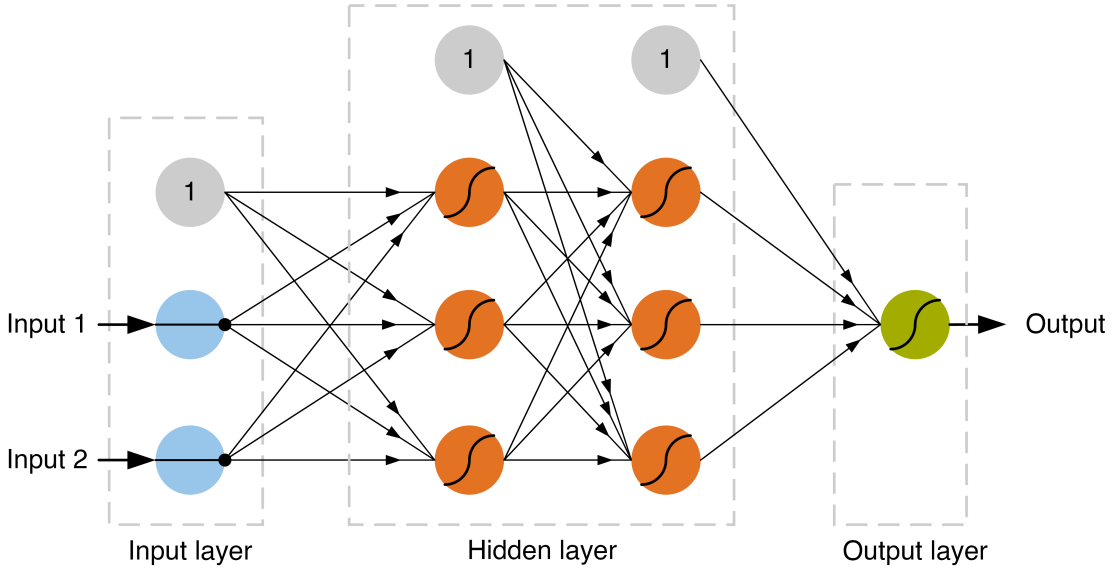


Figure 3-5: Exemplary architecture of a multilayer perceptron

Given that multilayer perceptrons are just the stacking of individual perceptrons on top of each other, then Eq. 3-3 describes the behavior of each individual layer, where for hidden layers no longer the inputs of the model but the outputs of the previous layer are used. Furthermore, the Heaviside function can be replaced by other nonlinear functions Φ. These functions are called **activation functions** and play an essential role, since without them the MLP would merely be a linear transformation of the inputs. The most commonly used

activation functions are listed in Eq. 3-4 [Goo-2018].

$$\begin{aligned} \text{Sigmoid function} \quad & \sigma(z) = \frac{1}{1+e^{-z}} \\ \text{ReLU} \quad & ReLU(z) = max(0,z) \\ \text{Hyperbolic tangent} \quad & tanh(z) = \frac{2}{1+e^{-2z}} - 1 \end{aligned} \tag{3-4}$$

Based on this, the mathematical description derived from Eq. 3-3 can be adapted and generalized for an MLP layer and states [Goo-2018, Gér-2019]:

$$\mathbf{h}^{(i)}_{(\mathbf{W}_i,\mathbf{b}_i)}(\mathbf{h}^{(i-1)}) = \Phi(\mathbf{W}_i^{\mathbf{T}}\mathbf{h}^{(i)} + \mathbf{b}_i) \tag{3-5}$$

where the Φ represents a general activation function and i is the index of the layer.

For the MLP shown in Fig. 3-5, the mapping from input $\mathbf{x}$ to output $\mathbf{y}$ using the two hidden layers $\mathbf{h}^{(1)}$ and $\mathbf{h}^{(2)}$ is given by:

$$\begin{aligned} \mathbf{h}^{(1)}_{(\mathbf{W}_1,\mathbf{b}_1)}(\mathbf{x}) &= \Phi(\mathbf{W}_1^{\mathbf{T}}\mathbf{x} + \mathbf{b}_1) \\ \mathbf{h}^{(2)}_{(\mathbf{W}_2,\mathbf{b}_2)}(\mathbf{h}^{(1)}) &= \Phi(\mathbf{W}_2^{\mathbf{T}}\mathbf{h}^{(1)} + \mathbf{b}_2) \\ \mathbf{y}_{(\mathbf{W}_3,\mathbf{b}_3)}(\mathbf{h}^{(2)}) &= \Phi(\mathbf{W}_3^{\mathbf{T}}\mathbf{h}^{(2)} + \mathbf{b}_3) \end{aligned} \tag{3-6}$$

In summary, the architecture of MLPs can be described as follows [Shr-2019]:

- The neurons of one layer are connected to all neurons of the previous and the next layer. Each connection has a weight.
- The connections of a neuron with the previous one are summed up weighted and passed to a nonlinear activation function.
- The output of the activation function is connected to the neurons of the next layer.

Different problems require different solutions

The supervised learning problems can essentially be divided into two subgroups. Regression problems are problems where a continuous numerical value has to be predicted. In the case of classification problems, the instance must be assigned to one of a finite set of classes. The group of classification problems can be further divided into binary problems (only two possible classes), multilabel binary problems (multiple, non-exclusive classes) and multiclass problems (multiple exclusive classes). Each type of problem also requires an appropriate modification of the output layer, so that the output has the desired structure. [Gér-2019]

For binary classification problems, it is possible to simply use the sigmoid function as the activation function [Gér-2019, Goo-2018]. This activation function returns a value between 0 and 1, indicating the probability that the analyzed instance belongs to the class under consideration [Gér-2019]. Multilabel binary problems can be treated as multiple single binary classification problems, where the output layer has as many neurons with a sigmoid function as activation function as there are different classes [Gér-2019]. In the case of exclusive classes, it must be ensured that the sum of the probabilities of the individual classes does not exceed 100 %. To ensure this property, a so-called softmax layer (see Eq. 3-7) is used [Goo-2018]. For each class i out of K possible classes, the probability that the instance belongs to this class is output.

$$softmax_i(x) = \frac{e^{x_i}}{\sum_j^K e^{x_j}} \tag{3-7}$$

Training

The points presented up to this point in Sec. 3.2.1 address only the pure architecture of the models. Thus, the question arises how a model which can predict solutions for a specific problem is created from an ordinary and common architecture. The training already introduced in the previous segment represents exactly this tuning of a model with a defined architecture for a specific problem. This adjustment is done by modifying the weight matrix W and the bias vector b in such a way that the desired outputs are obtained.

Prerequisite for this tuning to be done correctly is the ability to quantify and measure the quality of a model. For this purpose, a cost or loss function L is introduced, which characterizes the difference between the predictions of the model and the desired results [LeC-2015]. The goal of training is to minimize this error term and therefore training a neural network can be regarded as a minimization problem and is performed using the **backpropagation** algorithm, which consists of three steps.

In the first step, also called forward pass, the model outputs y are calculated according to Eq. 3-6 based on the inputs x. Then, using the selected loss function, the deviation between the current predictions of the model and the correct outputs is calculated [Gér-2019, LeC-2015]. The second step, also called reverse pass, applies the chain rule for derivatives to determine the influence of each connection weight on the loss function [Gér-2019, LeC-2015]. The last step consists of applying an optimization algorithm (e.g. Gradient Descent, Momentum optimization or Adam) to update the weights so that the error is minimized [Gér-2019].

Over- and underfitting

The quality of the models is basically assessed on the two criteria. The first is the loss during training introduced above. The better and more complex the models, the smaller the training loss and higher the achieved accuracy. However, if the model is too complex, then not only the underlying behavior is modeled, but the data set itself. If one would attempt to classify new instances, then only poor results would be achieved, since the real underlying behavior was not correctly modeled. If this scenario occurs, where a low loss on the training set can be detected, but a high loss on an additional validation set, then this is referred to as **overfitting**. If, on the other hand, no high accuracy is achieved on the training set, then **underfitting** occures. The reason for this is usually a too low complexity of the model. [Gér-2019]

3.2.2 Convolutional Neural Networks

Although MLPs can achieve very good results for certain problems, such models may encounter limitations or be inefficient for other problems. If a model is supposed to have a large number of inputs, as is the case in image recognition, for example, the set of parameters of MLPs can get excessively large. This is due to the fact that each input is connected to all neurons of the next layer. At a resolution of only 400x400 pixels and only one hidden layer with 100 neurons, 16 million weights have to be tuned, which represents a substantial training effort.

A common feature of these types of machine learning problems is the structured form (often in the form of arrays) in which the data is presented. Convolutional Neural Networks (CNN) make use of this structure to reduce the number of required weights. Instead of the matrix vector multiplication which is performed within the MLP neurons, the mathematical operation called **convolution** is used. The mathematical expression for the 1D convolution of a signal X with a kernel (or filter) K states:

$$(K * X)(\tau) = \sum_{i=1}^{W} K(i)X(\tau - i) \tag{3-8}$$

Figure 3-6 exemplifies the convolution of a 1D signal with a kernel of size 3. The elements of the kernel are multiplied by the matching entries of the signal and summed up. Then the kernel is shifted in the corresponding direction and the process is repeated.

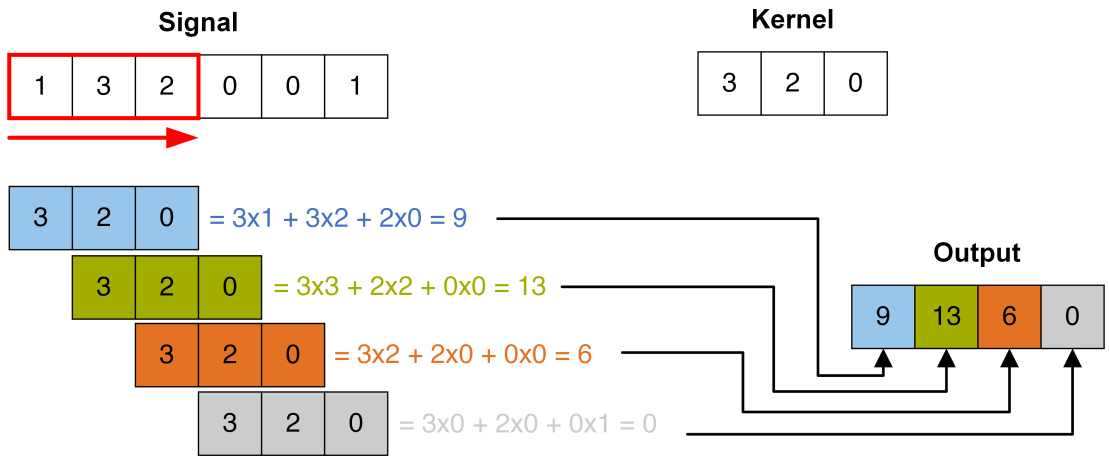


Figure 3-6: Exemplary convolution of a 1D signal

Replacing matrix-vector operations by convolution operations with a kernel yields a number of advantages [Goo-2018]. Since the selected kernel is smaller than the considered signal, a neuron of a deeper layer is connected only to a subset of the signal (also called receptive field of a unit), unlike MLPs in which a unit is connected to all previous units [Goo-2018, LeC-2015, Shr-2019]. This characteristic is called sparse connectivity. In addition to that, the same weights are used for all units (weight sharing). The combination of these two concepts significantly reduces the amount of adjustable parameters [Goo-2018, Shr-2019]. The architectural differences between MLPs and CNNs are illustrated schematically in Figure 3-7.

Main characteristic of CNN layers is the ability to identify and extract local features [LeC-2015]. Instead of adjusting the weights of the connections as in MLPs, the training of a CNN involves learning the filters that detect the most useful features which help to correctly classify the instance. Shallow layers are responsible for extracting low-level features of the signal, while filters from deeper layers can construct high-level abstractions [Gér-2019]. This ability to extract features from raw data is also called **representation learning** [LeC-2015]. Training a CNN follows the same principle of an MLP, but requires much less resources, since there are considerably fewer parameters [Shr-2019].

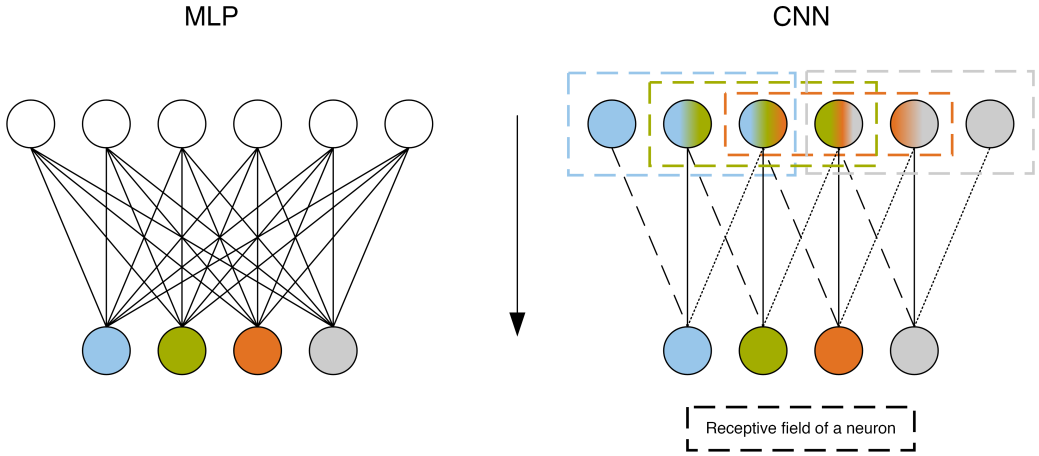


Figure 3-7: Schematic comparison of MLPs and CNNs, based on [Goo-2018].
Left: A neuron of a MLP consists of a combination of all neurons of the previous layer. Each connection has its own weight, which is stored separately.
Right: A CNN neuron consists of a convolution operation involving the neurons located in the receptive field with a filter. All neurons of a layer share the same filters. The affiliation of each neuron to each receptive field is color coded. Connections of the same line type represent connections with the same weights.

3.2.3 Recurrent Neural Networks

The previous two sections introduced the common MLPs and explained the CNNs. MLPs can be commonly employed for most types of problems, but may be inefficient for certain types of problems or may require a large number of layers to correctly model the problem. CNNs constitute a class of more specialized models that exploit the vector or matrix-like structure of the data to increase the efficiency of the model. In addition to the possible grid-like structure, data can also present itself in the form of a sequence, as for instance with sensor measurements. Just like CNNs, **Recurrent Neural Networks** (RNN) form a class of methods, which, however, is specialized for the processing of sequences [Goo-2018].

Unlike feedforward networks such as MLPs and CNNs, in which information flows in only one direction, RNNs feature information flow in both directions [Gér-2019]. The underlying reason for this is that RNNs use output loopback to store information from the past [Yu-2019]. These feedbacks can be viewed as hidden units and are read by each neuron in addition to the inputs from the previous layer at each point in time [Yu-2019, LeC-2015]. Hidden units offer the advantage of allowing the model to make decisions based not only on the current data, but also to consider the temporal context [LeC-2015, Yu-2019, LeC-2015, Goo-2018]. The basic architecture of an RNN neuron is shown in Figure 3-8.

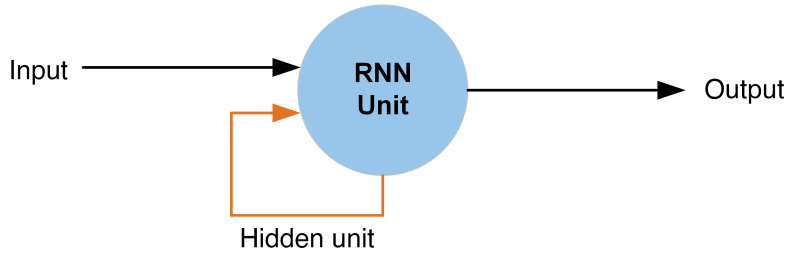


Figure 3-8: Schematic representation of a RNN unit

Dealing with long sequences

Simple RNNs with one or more layers can achieve very good results for relatively short sequences. The main limitation however, is when dealing with longer sequences [Gér-2019]. If the distance between the current time step and the beginning of the sequence is too large, then the context no longer contains the possibly relevant information of the beginning [Hoc-1997]. In addition, it was found that RNN can rapidly suffer from instabilities

during parameter tuning, which has a negative impact on the training process [Gér-2019, Shr-2019].

To deal with these problems, Hochreiter and Schmidhuber developed in 1997 an alternative to the simple RNN units, the so-called **Long Short-Term Memory** (LSTM) method [Hoc-1997]. The important modification is the addition of two additional states: the long term state $\mathbf{c}_{(t)}$ and the short term state $\mathbf{h}_{(t)}$. With the help of this additional long-term state, even temporally distant information can be remembered and considered in context. The architecture of the interior of an LSTM cell is shown in Figure 3-9 and is considered in more detail below.

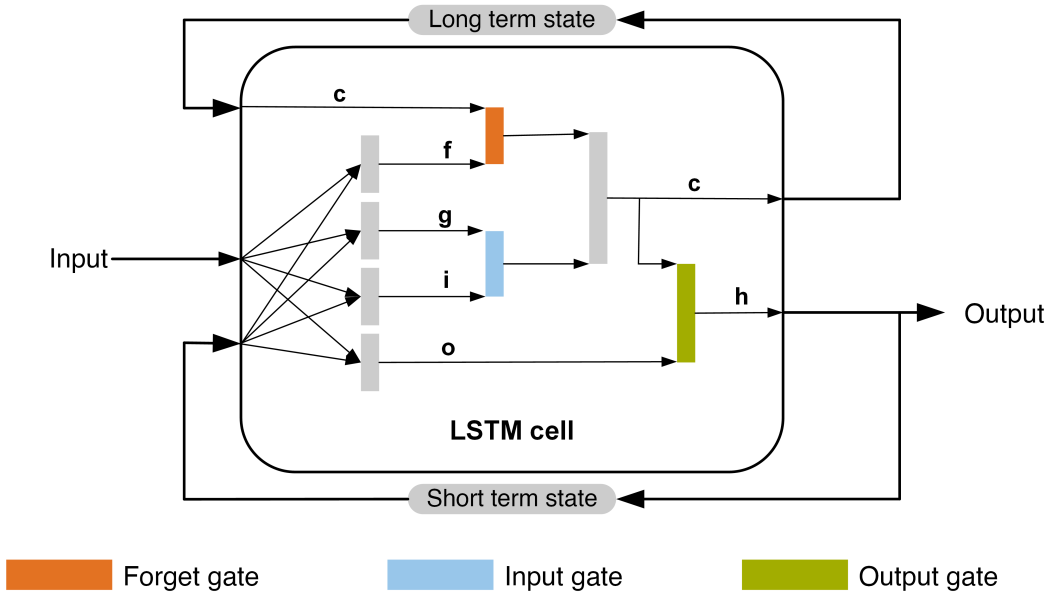


Figure 3-9: Architecture of the interior of an LSTM cell, based on [Gér-2019]

Within the LSTM cell a series of operations are performed which are used to determine and update these states. Essentially, the cell can be divided into three main parts. The following equations have been taken from [Gér-2019].

In the first part, the variable $\mathbf{g}_{(t)}$ is calculated based on the inputs and the information stored in the short term state:

$$\mathbf{g}_{(t)} = tanh(\mathbf{W}_{xg}^T \mathbf{x}_{(t)} + \mathbf{W}_{hg}^T \mathbf{h}_{(t-1)} + \mathbf{b}_g) \tag{3-9}$$

Meanwhile, in the second subsection, the vectors $\mathbf{f}_{(t)}$, $\mathbf{i}_{(t)}$ and $\mathbf{o}_{(t)}$ are calculated (see Eq. 3-10 , which control the so-called **forget gate**, **input gate** and **output gate**.

$$\begin{aligned}\mathbf{i}_{(t)} &= \sigma(\mathbf{W}_{xi}^T\mathbf{x}_{(t)} + \mathbf{W}_{hi}^T\mathbf{h}_{(t-1)} + \mathbf{b}_i)\\ \mathbf{f}_{(t)} &= \sigma(\mathbf{W}_{xf}^T\mathbf{x}_{(t)} + \mathbf{W}_{hf}^T\mathbf{h}_{(t-1)} + \mathbf{b}_f)\\ \mathbf{o}_{(t)} &= \sigma(\mathbf{W}_{xo}^T\mathbf{x}_{(t)} + \mathbf{W}_{ho}^T\mathbf{h}_{(t-1)} + \mathbf{b}_o)\end{aligned} \tag{3-10}$$

The forget gate is used to delete any no longer relevant information from the long-term state (first term of Eq. 3-11). The function of the input gate is to control which information of the currently active inputs should be stored in the long-term state (second term of Eq. 3-11).

$$\mathbf{c}_{(t)} = \mathbf{f}_{(t)} \otimes \mathbf{c}_{(t-1)} + \mathbf{i}_{(t)} \otimes \mathbf{g}_{(t)} \tag{3-11}$$

Finally, the output gate extracts the important information from the long-term state and stores it in the short term state (Eq. 3-12). The information of the short term state is also the output $\mathbf{y}_{(t)}$ of the LSTM cell.

$$\mathbf{y}_{(t)} = \mathbf{h}_{(t)} = \mathbf{o}_{(t)} \otimes tanh(\mathbf{c}_{(t)}) \tag{3-12}$$

For all equations introduced above $\mathbf{W}_{ab}$ denotes the weight matrix. The index a indicates the variable to which the weights refer. The second index b indicates which variable is calculated.

Bidirectional LSTMs

LSTM networks have a causal structure, i.e., outputs at a given time depend only on inputs and states from past moments in time [Goo-2018, Gra-2005, Gér-2019]. Schuster

and Paliwal [Sch-1997] developed an extension to RNNs which considers not only inputs from the past, but also future ones. While in a normal RNN the processing of a sequence starts at the beginning of the sequence and then moves forward, in **bidirectional** RNN the sequence is additionally considered backwards. To make this possible, an additional layer is added. The first layer processes the inputs in the forward direction in time, while the second layer processes the inputs backward [Gér-2019, Goo-2018]. Thus, both time directions can be processed simultaneously [Sch-1997]. Graves and Schmidhuber applied the concept of bidirectionality in conjunction with LSTM networks. In general, it is possible to say that LSTM networks give better results than simple RNN. The use of bidirectional LSTMs showed a further improvement of the results [Gra-2005]. A schematic representation of a bidirectional LSTM layer is shown in Figure 3-10.

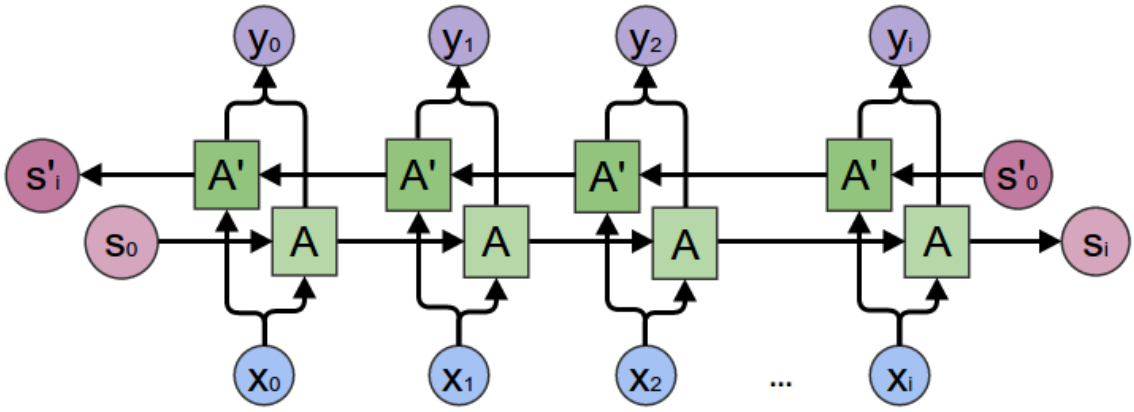


Figure 3-10: Schematic representation of a bidirectional LSTM [Ola-2015]

3.3 Time series classification

One particular format in which the data of a problem may be available is in the form of a Time Series. This is the case for many problems in engineering applications. The main distinguishing characteristic is the existence of a chronological order of the data [Fu-2011]. According to Ismail Fawaz et al. [Ism-2019], univariate and multivariate (MTS) time series, respectively, can be defined as follows:

> "A univariate time series $X = [x_1, x_2, ..., x_T]$ is an ordered set of real values. The length of X is equal to the number of real values T."

> "An M-dimensional MTS, $X = [X^1, X^2, ..., X^M]$ consists of M different univariate time series with $X^i \in \mathbf{R}^T$."

If for each univariate or multivariate time series instance $\mathbf{X}_i$ an additional label $\mathbf{Y}_i$ assigning the instance to a class is provided, then the problem is a **Time Series Classification** (TSC) problem [Ism-2019]. The goal is to correctly classify a sample into one of K classes.

Time series are characterized by their continuous nature of data [Fu-2011]. For instance, sensor measurements are continuous, i.e., they are taken over a longer period of time with a high measurement frequency. To convert a time series dataset into a time series classification problem, the continuous data must first be divided into samples [Ban-2014]. This process is called segementation.

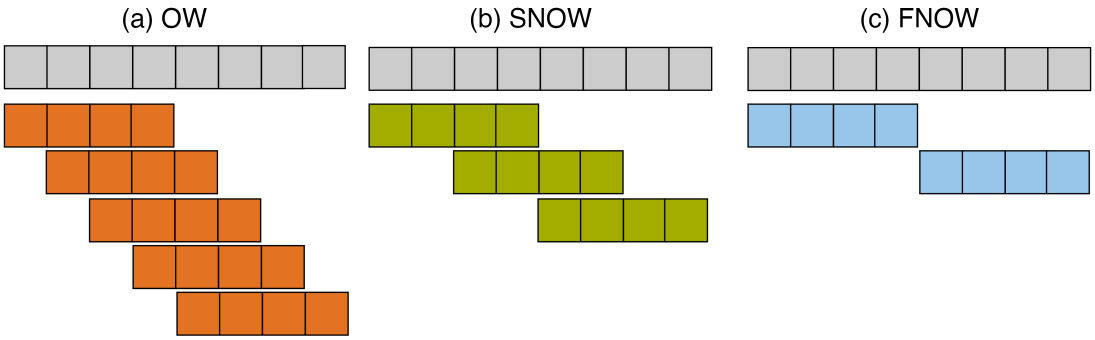


Figure 3-11: Exemplary representation of the segmentation methods

The most common method for segmentation is the **sliding window** method [Fu-2011, Ban-2014]. It involves sliding a window with a fixed window width along the time axis. This is exemplified in Figure 3-11. The two properties of the method that affect the segmentation the most are the window width and the overlap of two neighboring samples. In the overlapping temporal window (OW) all but one time step overlap. For the semi-non-overlapping temporal window (SNOW), 50 % of the time steps overlap [Sin-2021] . These first two methods suffer from a high bias, since a large part of the samples coincide [Sin-2021]. With the fully non-overlapping temporal window (FNOW) the samples do not overlap at all. The disadvantage is that the number of generated samples is significantly reduced [Sin-2021].

One of the most well-known applications of time series classification problems is activity recognition, especially **human activity recognition** (HAR). The goal is to assess the activity performed by a person based on sensor measurements of various types [Min-2020,

Mur-2017]. With the recent developments in the field of smartphones and wearables, the number of available data also increased, which further boosted the progress in the field. In the field of HAR, a wide range of research has been carried out in recent years. Activity recognition of construction machines is highly similar to HAR in many aspects. Selected relevant studies are discussed in Chapter 6 in the context of the proposed models.

Part II

Methodology

4 Data understanding

"The data understanding phase starts with an initial data collection and proceeds with activities in order to get familiar with the data, to identify data quality problems, to discover first insights into the data, or to detect interesting subsets to form hypotheses for hidden information." [Stu-2021]

4.1 Data acquisition

The investigations into the automatic activity recognition of construction machines in special civil engineering will be conducted based on a use case. For this purpose, data from a real construction site will be used. The data used in this thesis originate from the construction project "*Westtangente Rosenheim*". Within the scope of this project, the construction company *Bauer Maschinen GmbH* was responsible for the production of bored piles for the foundation of the planned bridge near Rosenheim, Bavaria. Several bored piles with lengths of up to 45 m and diameters of 1200 mm were produced. [Fis-2020]

During the drilling process, several machine data are recorded with the help of different sensors. All available sensors with the corresponding units of the measurement data are shown in Table 4-1. In total, measurements from 20 sensors are available. The readings are taken at a frequency of 1 Hz. The data measured during the working time is then transferred to a proprietary platform from Bauer and is available for analysis.

In addition to the machine data, the performed activities of the drill rig were recorded on 16 days. As already described in the motivation section (see Section 1.1), this recording is performed manually. For this purpose, Bauer's B-Tronic software is used to record the start and end times of the different activities. In total, approximately 102 hours of activities were recorded between 10/14/2019 and 12/04/2019.

Before any analysis of the data is done, a single data set must be created. This is the case since the machine data and the activity data are collected separately. Each measurement of the machine data is assigned a timestamp, each of which is then assigned to a specific

Table 4-1: Overview of available sensors

Sensor	Unit	Sensor	Unit
Depth	m	Torque steps	-
Torque of rotary drive	kNm	Torque of Kelly bar	%
Speed of rotary drive	rpm	Aux. winch rope force	t
Main winch rope force	t	Crowd depth	m
Crowd-force	t	Casing length	m
Main winch rope speed	cm/min	Status rig	-
Pressure pump 1	bar	Main winch gear mode	-
Pressure pump 2	bar	Inclination X	deg
Pressure pump 3	bar	Inclination Y	deg
Pressure pump 4	bar	Boring threshold	m

label using the start and end times of the respective activities from the activity data. This was automated using a Python code (see appendix A.1). As a result, one dataset per day is obtained (except for the 20.11, 25.11, 27.11 and 02.12, which were split into two different datasets), which contains the sensor data and the corresponding label of the activity.

4.2 Data exploration

In the following section, initial investigations of the data set are performed. In Table 4-2, the number of observations of the different activities are broken down by day. For the labeling, labels based on the process steps from Section 3.1 were used. A total of 27 activities are considered. It is recognizable that on different days also various activities take place with different frequencies. For example, the activities that belong to the process step **Concreting** in general are not performed on certain days (e.g. 14.10, 06.11 or 27.11). However, the activities which belong to the process step **Drilling** are present in every data set. Other activities, which do not directly belong to the drilling process, such as **Relocate**, **Pause**, **Wait**, **Failure** or **Maintenance**, are only occasionally present. The reason for this is the fact that the data originates from a real operation and was not generated artificially. This leads to the dataset being highly imbalanced. For instance, **Empty** represents 8.97 % and **Drilling** represents 5.28 % of the data, while **Drilling with bucket** and **Wait for concrete** are only 0.23 % and 0.15 % of the dataset, respectively.

Table 4-2: Breakdown of collected data per day and label

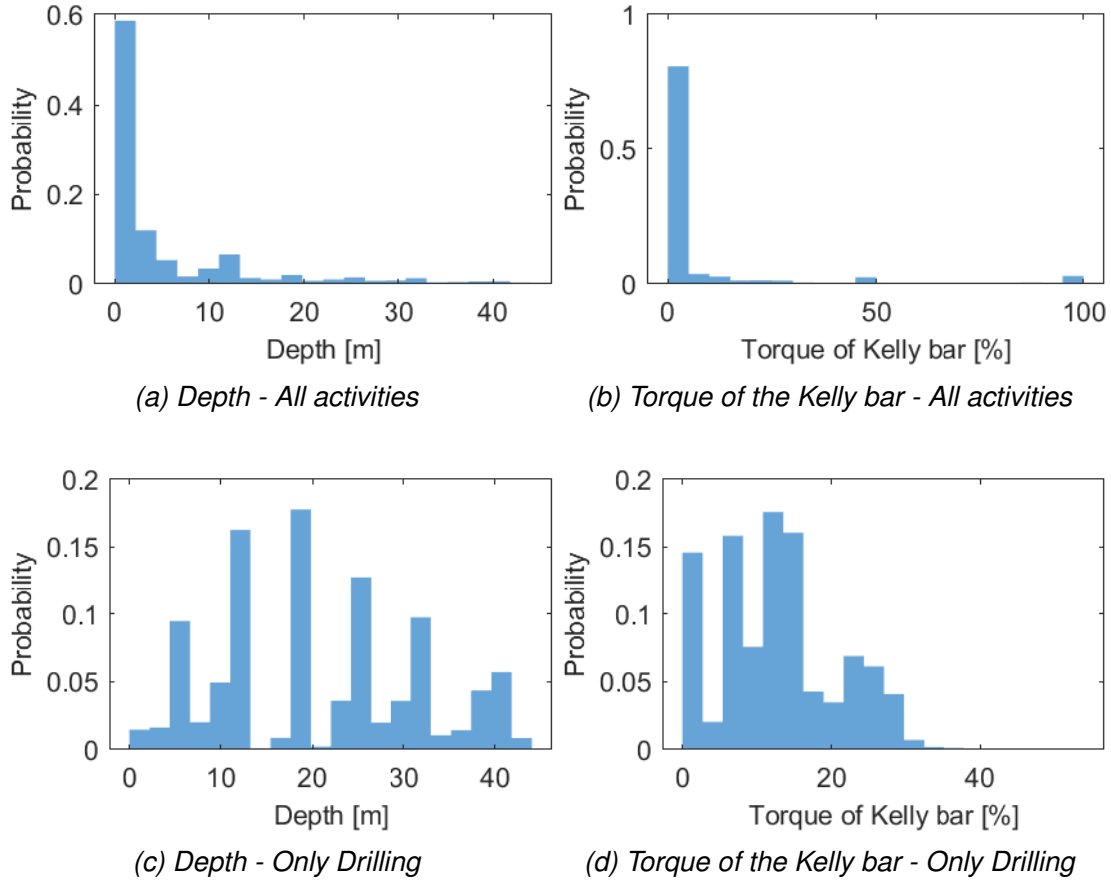
Activity	14.10	16.10	21.10	23.10	25.10	28.10	30.10	06.11	11.11	13.11	18.11	20.11	20.11	25.11	25.11	27.11	27.11	02.12	02.12	04.12	Total	%
Concreting	-	-	3009	1905	2356	2831	2600	-	4118	4838	-	-	-	-	-	-	-	1	1	-	21659	5.85 %
Place pouring pipe	-	-	1549	1194	1044	1164	1217	-	1628	1555	1658	-	-	-	-	-	-	-	-	-	11009	2.97 %
Remove pouring pipe	-	-	797	-	-	499	476	-	1139	1696	1	4	1	-	-	-	-	-	-	-	4613	1.25 %
Remove casing	-	-	1697	762	896	1206	981	-	1574	1944	480	-	2	-	-	-	-	-	-	-	9542	2.58 %
Pull standpipe	-	-	1791	4200	1295	614	677	-	3699	784	-	-	1	-	-	-	-	3	1	-	13065	3.53 %
Installation cushion	-	1885	2488	2987	3068	2088	2279	-	133	1182	2341	-	537	-	482	-	-	-	-	-	19470	5.26 %
Installation rebar cage	-	-	859	805	1032	1015	735	-	1115	1404	1464	-	1272	-	255	-	-	-	-	-	9956	2.69 %
Place standpipe	1256	1776	2093	5162	2484	2529	2545	464	3747	3079	2167	1531	1812	2149	1118	1303	2680	2095	1067	4344	45401	12.26 %
Screw in casing	1149	3718	3507	2858	2742	3329	2617	796	5468	2731	76	1933	873	2417	538	1435	1333	2373	31	2412	42336	11.44 %
Lowering	267	892	956	655	802	848	797	200	1388	1128	848	506	989	698	715	268	974	1286	3319	1269	18805	5.08 %
Drilling with drill bucket	675	61	-	-	-	25	-	-	-	-	-	-	-	-	38	-	-	-	-	46	845	0.23 %
Pulling	552	2051	2329	1461	2028	2057	2027	603	2928	2465	1227	1367	1431	1470	1197	871	1901	1987	526	2739	33217	8.97 %
Drilling		1275	1371	1174	1337	1259	1274	305	1823	1226	710	879	661	1204	505	593	908	1185	341	1522	19552	5.28 %
Emptying	743	1749	2160	1695	1763	2121	2091	471	2274	1655	683	1271	824	1731	672	755	1169	1713	266	2155	27961	7.55 %
Screw in casing (CM)	-	-	-	-	-	-	-	-	2073	2379	3284	-	4209	-	2924	-	2805	186	1519	2898	0	0.00 %
Pull standpipe (CM)	-	-	-	-	-	-	-	-	-	1976	-	-	-	-	-	-	-	-	-	-	0	0.00 %
Other	446	1610	669	463	403	992	358	18	991	391	856	-	-	52	16	410	255	31	1	54	8016	2.17 %
Refill water	-	850	892	832	1006	976	996	460	1166	1753	710	1115	582	1434	562	1189	955	1210	105	1536	18329	4.95 %
Tool exchange	-	-	367	387	50	-	175	-	674	284	181	-	231	-	299	-	153	-	197	196	3194	0.86 %
Refueling	-	503	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	171	674	0.18 %
Depth sensing	-	-	240	205	211	221	336	-	710	789	97	-	168	-	43	-	100	-	-	28	3148	0.85 %
Relocate	-	-	-	1044	-	432	-	-	-	541	737	-	-	208	-	-	-	-	-	-	2962	0.80 %
Waiting for other	-	-	-	-	-	-	277	-	302	-	-	-	-	-	-	-	41	2948	1	-	3569	0.96 %
Break	-	-	5337	4293	4583	4603	7024	-	4204	-	255	-	4321	-	3864	-	1220	-	3701	3224	46629	12.60 %
Waiting for concrete	-	-	-	-	-	533	6	-	-	-	-	-	-	-	-	-	-	-	-	-	539	0.15 %
Maintenance	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	0.00 %
Failure	-	-	-	-	-	-	-	-	-	5683	-	-	-	-	-	-	-	-	-	-	5683	1.54 %
Total	5088	16370	32111	32082	27100	29342	29488	3317	39081	35128	14491	8606	13705	11363	10304	6824	11689	14832	9557	19696	**370174**	
% of samples	1.37%	4.42%	8.67%	8.67%	7.32%	7.93%	7.97%	0.90%	10.56%	9.49%	3.91%	2.32%	3.70%	3.07%	2.78%	1.84%	3.16%	4.01%	2.58%	5.32%		

Furthermore, different statistical metrics (mean, standard deviation, minimum, maximum, and the .25, .50, and .75 quantiles) of the available sensors data are examined. These are shown in Table 4-3. One can easily recognize the fact that three of the sensors do not provide any meaningful measurements and only assume a constant value. The data from the **Casing Length**, **Boring threshold**, and **Status rig** sensors have no information content, so these can be deleted from the dataset and do not serve as inputs to the models.

Table 4-3: Statistical properties of the data set

Sensor	mean	std	min	25 %	50 %	75 %	max
Depth	5.36	8.84	0	0	1.13	6.08	43.08
Torque of the Kelly bar	7.64	21.44	0	0	0	0	100
Rope Force Main Winch	13.58	3.43	0.40	12.80	14.30	14.90	49
Rope Force Aux. Winch	0.09	0.67	0	0	0	0	14.10
Speed of Rotary Drive	2.60	7.40	0	0	0	0	52
Crowd Depth	0.01	0.09	0	0	0	0	1.27
Pressure Pump 1	47.49	92.25	0	1	2	42	376
Pressure Pump 2	53.58	96.37	0	1	1	78	378
Pressure Pump 3	34.75	61.78	0	0	0	56	414
Pressure Pump 4	27.34	61.46	0	1	2	5	338
Torque Steps	2.85	0.50	0	3	3	3	3
Casing Length	0	0	0	0	0	0	0
Boring threshold	0	0	0	0	0	0	0
Status Rig	0	0	0	0	0	0	0
Inclination X	-0.59	1.29	-6.40	-0.70	0	0	4.90
Inclination Y	0.69	1.69	-5.5	-0.10	0	0.30	8
Crowd-Force	0.59	9.61	-82.80	0	0	0	78.10
Torque of the rotary drive	30.33	84.08	0	0	0	0	390
Main Winch Rope Speed	5.28	12.06	0	0	0	2.39	191.94
Gear Mode Main Winch	0.71	0.90	0	0	0	2	2

A further issue which stands out is the very small or at times even non-existent interquartile range (IQR, distance between the .25 and the .75 quantile). For example, when considering the feature **Torque of the Kelly bar**, both the .25 and the .75 quantile are zero, although the maximum measured value is 100. Similarly, for the feature **Depth**, the IQR is significantly smaller than the difference between the minimum and maximum values. Such behavior is evident for a large number of features. To further examine this phenomenon, Figures 4-1a



(a) Depth - All activities

(b) Torque of the Kelly bar - All activities

(c) Depth - Only Drilling

(d) Torque of the Kelly bar - Only Drilling

Figure 4-1: Histograms of selected sensor data

and 4-1b show exemplary plots of the distributions of the measured data.

Both for the feature **Depth** as well as for the feature **Torque of the Kelly bar** a clear right skewed distribution is noticeable. In both cases, the most frequent value is zero. This also makes sense, since for a large part of the activities the drilling tool is at the surface and no torque is transmitted. It would seem logical to consider the extreme values on the right as outliers and remove them from the data set. Yet, if only isolated activities are examined in more detail instead of the entire data set (Fig. 4-1c and 4-1d), it can be observed that the values which appear as extreme values in Fig. 4-1a and 4-1b are not actually such extreme values. Since, however, a large number of classes are present and the individual classes do not usually exceed 10 % of the data set, measurements which do not constitute outliers may stand out as extreme values when considered as a whole, since the classes amount to only a small portion of the entire data set. Removing such data would only manipulate the data set by eventually removing individual classes. The histograms for the remaining sensors are shown in Figure B-1 in the Appendix.

An additional reason, which would oppose the removal of outliers, is the sheer nature of the process. The occurrence of extreme values is normal for some processes. A pure formal removal of these cannot be performed, rather they must be considered in the context of the actual process characteristics. This can be exemplified by considering the depth of the drilling tool during a drilling operation. When a drilling cycle is started, the drilling tool moves downward from the surface (0 m) until the desired depth is reached. If such an procedure is repeated multiple times and each time the drilling is deeper, then the maximum depth reached (e.g. 20 m) is measured only once, while the remaining segments (between 0 m and 20 m) are measured multiple times. It is evident in this case that the measurement of the maximum depth reached cannot be considered as an outlier.

Furthermore, the correlation between the individual sensors was also investigated. For this purpose the function *corr()* of the Python library *Pandas* was employed. The correlation matrix is shown in Figure 4-2. Overall, the dataset is characterized by weak correlations. Notable exceptions are the correlations between Torque of the Kelly bar, Torque of the rotary drive, and the pressures of pumps 1 and 2.

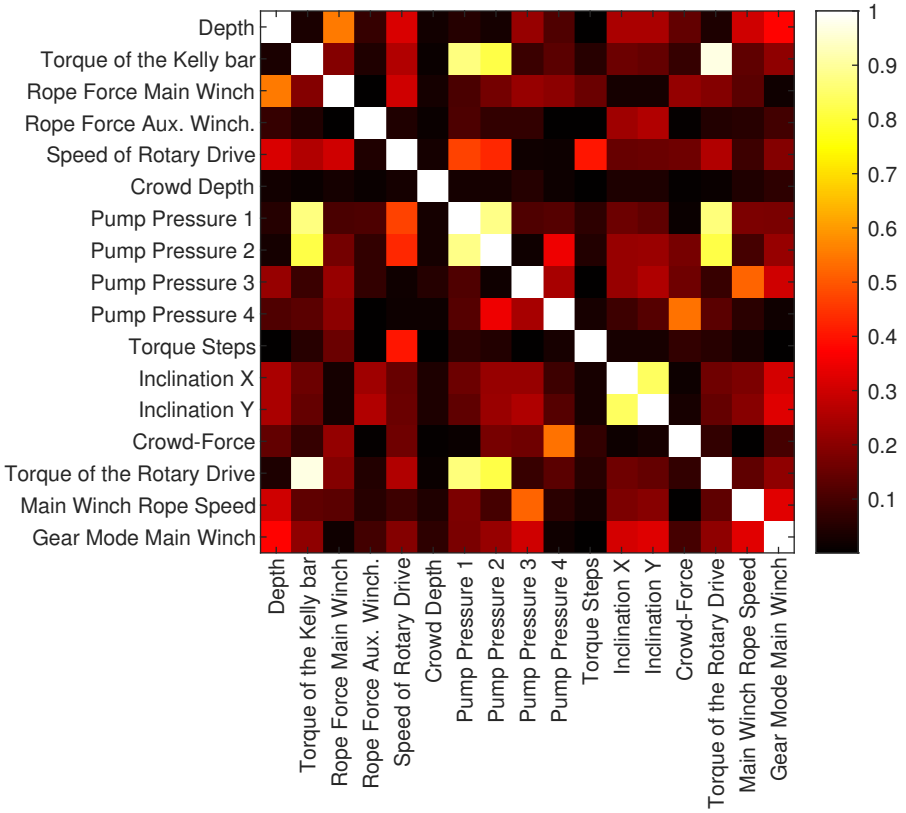


Figure 4-2: Absolute values of the correlation coefficients of the sensors

5 Data preparation

"The data preparation phase covers all activities to construct the final dataset (data that will be fed into the modeling tool(s)) from the initial raw data. Data preparation tasks are likely to be performed multiple times, and not in any prescribed order. Tasks include table, record, and attribute selection, data cleaning, construction of new attributes, and transformation of data for modeling tools." [Stu-2021]

5.1 Data segmentation

As mentioned in Section 3.3, continuous time series measurements need to be preprocessed so that they can be regarded as a supervised learning problem. The first aspect is the segmentation of the continuous measurement into discrete samples, which serve as inputs to the models. Since different model architectures and types also require inputs with different data structures, the segmentation is also partially model dependent. Furthermore, the corresponding sample must be determined for each sample.

Models which do not consider temporal context require only the measured values at a given time point. Therefore, the inputs must have the shape of a matrix $\mathbf{X} \in R^{m \times n}$, where m denotes the number of sensors and n the number of samples. Each time point corresponds to a sample whose label is the activity at that exact time point.

On the other hand, for models which also consider the temporal context, samples have to be generated from the continuous data set. For this, the sliding window approach explained in section 3.3 with a fixed window width is used. The window width corresponds to the amount of time considered retrospectively. If the window width is given by T and if one is located at time $t = t_k$, then a sample consists of the measurements from time $t_k - T$ to time t_k. The larger the window width, the larger the temporal context passed to the models. In this case, the input matrix of the models is three-dimensional and has the form $\mathbf{X} \in R^{m \times T \times n}$. In the context of this work, different window widths and the influence of these are investigated. Starting point is a window width of $T = 16$, which corresponds to a context of **16 seconds**,

since the sensors measure with a frequency of 1 Hz. Furthermore, window widths of **8**, **32** and **48 seconds** are examined.

Another possible parameter in segmentation is the overlap of two neighboring samples. Again, several approaches (**OW**, **SNOW**, and **FNOW**) are examined. The detailed description of the three methods is given in Section 3.3. Finally, a label must be assigned to each sample. In contrast to the baseline MLP model, which considers only one time point and the associated label is unambiguous, different approaches can be followed in the case of sliding windows. The first possibility investigated is, for a sample between time points $t_k - T$ and t_k, to select the activity at the **last** time point t_k as the associated label of the sample. The second possibility investigated uses the **most frequent** activity occurring between time points $t_k - T$ and t_k as the label.

For the automatic generation of the samples, the procedure was automated through a self-written Python function. The source code is given in Appendix A.2. The function takes, among others, the size of the window, the overlap and the labeling method as inputs and outputs the input matrix of the models $\mathbf{X}$ and the label vector $\mathbf{y}$. In the context of this work, only the first option is employed.

5.2 Scaling

Since each feature represents the measurements of a single sensor, they often possess different orders of magnitude. For example, the torque of the Kelly bar can assume values between 0 and 100, while the values of the inclination in X direction usually range between -10 and 10.

These differences in scale usually have a negative impact on the performance of the models [Gér-2019]. For this reason, the data of all sensors are scaled. The two most common methods are *min-max scaling* and *standardization*. However, as it was recognized in Chapter 4.2 that the sensor measurements are not normally distributed, scaling by means of *Standardization* is not advantageous. Therefore, all data are transformed so that they lie

in the range between 0 and 1. The transformation algorithm for a feature $\mathbf{x}$ is given by [Gér-2019]:

$$x_i' = \frac{x_i - min(\mathbf{x})}{max(\mathbf{x}) - min(\mathbf{x})} \tag{5-1}$$

where $\mathbf{x}'$ is the scaled feature.

5.3 Data splitting

The complete data set is divided into a training set (TS), validation set (VS) and test set (TTS) according to the aspects explained in Section 3.2.1. The test set amounts to approximately 30 % of the entire data set and the validation set constitutes another 20 % of the remaining 70 %. In total, approximately 56 % is used as training set, 14 % as validation set and 30 % as test set. The subdivision of the data can follow different approaches, which are described in more detail below.

The primary and also most widely used method is based on a random partitioning of the entire dataset. This approach was also used in previous student theses [Ors-2020, Bi-2020, Lia-2020]. The subdivision is mostly done automatically using the function *Scikit-Learn* included in the Python library *train_test_split()*, which randomly splits a given vector or matrix [Sci-2021]. One potential issue that could arise and affect the results is the similarity of the training and test data. Given that the data is recorded at a relatively low frequency and the process is generally characterized by a rather slow behavior, it is perfectly possible that two observable samples are extremely similar, but are located once in the training set and once in the test set. This would be equivalent to using one sample for both training and testing, which could potentially bias the results. Since the goal of the model is to model the underlying behavior and not to memorize the data, such a division of the data can have a negative impact.

One possible approach to address these potential issues is to split the data by days. In this case, the data is still divided into training, validation and test set, but the single data sets form the smallest possible unit for splitting. Samples from a single data set cannot be

present in both the training set and the test set. Furthermore, such a split can potentially provide further insight into the generalization capabilities of the models and allows testing of the models on a continuous time window rather than on single samples. The assignments of the individual data sets to the respective training, validation and test sets is shown in Table 5-1.

Table 5-1: Data splitting by days

Date	TS	VS	TTS	Date	TS	VS	TTS
14.10		X		18.11	X		
16.10			X	20.11 (1)	X		
21.10			X	20.11 (2)			X
23.10	X			25.11 (1)	X		
25.10			X	25.11 (2)	X		
28.10		X		27.11 (1)			X
30.10	X			27.11 (2)	X		
06.11			X	02.12 (1)			X
11.11	X			02.12 (2)	X		
13.11	X			04.12		X	

During the splitting, attention is paid to the fact that the ratios between TS, VS and TTS are still followed. However, since the smallest unit is a complete data set of one day, it is not possible to precisely match the percentages. Attention is also paid to avoid very strong imbalances of activities in the different data sets. The exact ratios are shown in Table 5-2.

Table 5-2: Percentage of samples by dataset type

Dataset	% of samples
Training (TS)	54.51 %
Validation (VS)	14.62 %
Test (TTS)	30.87 %

6 Modeling

"In this phase, various modeling techniques are selected and applied, and their parameters are calibrated to optimal values. Typically, there are several techniques for the same data mining problem type. Some techniques require specific data formats. There is a close link between Data Preparation and Modeling." [Stu-2021]

6.1 Requirements

Considering the motivation, the state of the art and the theoretical foundations of Deep Learning and time series classification, this chapter aims to formulate the requirements for the models to be investigated and to present the proposed approach. Based on the aspects discussed up to this point, it is possible to summarize the following requirements regarding the models:

1. The models must be capable of predicting the correct activity performed from existing telematics data.
2. The models should take into account the time dependencies of the telematics data.
3. The models must be capable of using non-strongly filtered data as inputs, since the presence of such data cannot be ruled out in practice.
4. The models should need as little human intervention as possible, as this can be time-consuming and costly.

6.2 Models

Within the scope of this work, different models are investigated. The focus is on the performance of hybrid models consisting of CNN and RNN, which have proven to perform well in other similar applications. To compare the performance, two additional baseline models are investigated. Section 6.2.1 explains the baseline models, while Section 6.2.2 deals with the hybrid models. An overview of all used models is given schematically in Figure 6-1.

6.2.1 Baseline models

The performance of a complex model can only be considered in comparison to other models. If simple models already achieve accuracies of 75 %, then an accuracy of 77 % of a very complex model does not represent a large increase in performance. If, on the other hand, a simple model only achieves 40 % accuracy, then the use of a more complex model, which is also significantly more resource-intensive, might be appropriate. The models which serve as a comparison and a starting point for the performance assessment of more complex approaches are called baseline models.

Multilayer perceptron

The first baseline model considered is based on one of the models investigated by Fischer et al. [Fis-2021]. Here, a simple ANN with 1 hidden layer containing 7 neurons was used to classify five different activities. The accuracies achieved ranged from 77 % to 96 % [Fis-2021].

The model based on these results and used in this work is an MLP, which consists of 4 layers with 128 neurons each (see Fig. 6-1a). The sensor data at a given time are used as inputs. This kind of MLP architecture does not take into account temporal dependencies, since no temporal context is provided to the model. To avoid possible overfitting, a dropout rate of 30 % is set, i.e., during training, 30 % of the neuron connections are randomly disabled at each iteration. Since this model is the only one which does not model the temporal dependencies, it is used as a comparison to investigate the influence of these temporal relationships in activity recognition.

Long-short term memory

The second model, which is considered as a baseline model, is a neural network consisting of LSTM layers. LSTM networks are well-known and frequently the most widely used models when considering sequences. As explained in Section 3.2.3, LSTMs distinguish themselves exactly by their ability to model temporal relationships. For this reason, a model consisting of 2 LSTM layers, each with 128 cells is investigated within the scope of this

work. The raw sensor data without any processing (filtering, feature extration, etc.) are used as inputs. As with the MLP baseline model, a dropout rate of 30 % is used.

6.2.2 Hybrid models

The focus of the investigations in this thesis is on two models, which are made of a combination of two types of neural networks and are therefore called **hybrid models**. In the following, the two variants of these hybrid models and their advantages and backgrounds are explained in detail.

DeepConvLSTM

The foundation of hybrid models is the combination of the advantages of **convolutional neural networks** and **recurrent neural networks**. This combination has already been studied by different authors dealing with different application areas. To address problems in human activity recognition, Ordóñez and Roggen [Ord-2016] developed an architecture called *DeepConvLSTM* in 2016. This architecture combines the ability of CNNs to extract features from pure signals (see Section 3.2.2) with the ability to model temporal dependence of RNN (see Section 3.2.3). The original architecture proposed by Ordóñez and Roggen [Ord-2016] consists of one input layer, four convolutional layers, two LSTM layers and one softmax activation layer. The four convolutional layers convolve the multivariate sensor data with different sensors and pass the convolved signals to the LSTM layers. The convolutional filters are set so that the outputs of the convolutional layers form high level representations of the sensor data. Thus, as already explained in Chapter 3, local and short term properties and features can be extracted. These high-level representations are passed to the LSTM layers, which are then responsible for modeling the long-term relationships. The advantages of this architecture were summarized by Ordóñez and Roggen:

> "[The framework] (i) is suitable for multimodal wearable sensors; (ii) can perform sensor fusion naturally; (iii) does not require expert knowledge in designing features; and (iv) explicitly models the temporal dynamics of feature activations.". [Ord-2016]

It is important to mention that these advantages match well with the requirements from Section 6.1. Especially the capabilities of CNN to automatically merge the data from different sensors and to automatically generate high-level features are very useful compared to simple RNN.

Based on this architecture, Slaton et al. [Sla-2020] examined a variation of the model concerning the activity recognition of construction machines. The model is adopted with the same number of layers and neurons. In addition, batch normalization layers between convolutional layers and a dropout layer between convolutional and recurrent layers are added. Slaton et al. achieved good results using this architecture and, since telematics data is most similar to acceleration data in terms of the nature of data, it is also conceivable that such an architecture would also yield good results for telematics data.

The model presented in this thesis is strongly based on these two aforementioned works and includes only minor adaptations (see Fig. 6-1c). The first adaptation is in terms of the number of layers. Instead of four convolutional layers, only three layers with 64 feature maps each are used, between which, as in Slaton et al. [Sla-2020], batch normalization layers are also added. Each LSTM layer consists of 128 LSTM cells. In addition, a dropout layer with a dropout rate of 30 % was added between each layer to further minimize the risk of overfitting, as there was a significant difference between training and validation loss in the experiments of Slaton et al.

DeepConvBiLSTM

The second approach involving hybrid models includes a replacement of the unidirectional LSTM layers by **bidirectional LSTM** layers. As explained in Section 3.2.3, the inputs of bidirectional RNN are processed in the two temporal directions. Replacing unidirectional with bidirectional RNN generally has the potential to improve the results. Xu et al. [Xu-2019] investigated the use of bidirectional LSTM in the context of the *DeepConvLSTM* architecture and human activity recognition. It was found that the use of bidirectional LSTM further improved the results. Therefore, the impact of using bidirectional LSTM is further investigated in this thesis.

The proposed bidirectional architecture (*DeepConvBiLSTM*) is a modification of the unidirectional architecture and further consists of one input layer, three convolutional layers, two recurrent layers and one softmax activation layer (see Fig. 6-1d). However, the two LSTM layers are replaced by two bidirectional LSTM layers. The number of cells within each layer is not modified.

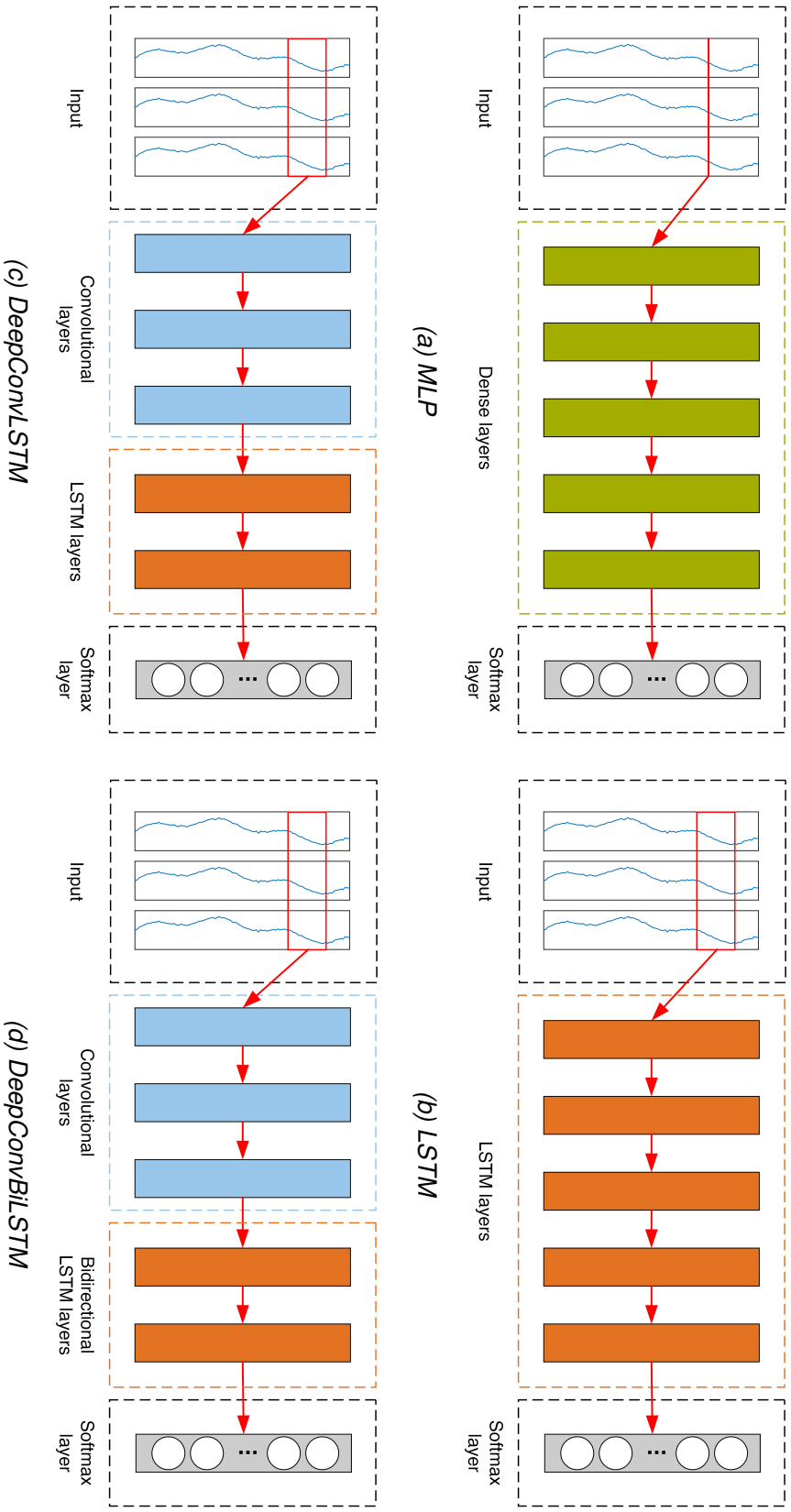


Figure 6-1: Overview of the selected models

6.3 Hierarchical classification

An alternative approach to classification is the hierarchical classification. This is especially useful in cases where the classes under investigation have a certain hierarchy or certain classes can be grouped into metaclasses. One of the most popular approaches to hierchical classification is the local classifier per parent node approach [Sil-2011]. For each subgroup a separate multi-class classifier is trained, which is only specialized to distinguish between child nodes within a group [Sil-2011]. For example, activities of a machine can be examined with different levels of detail, as was also done by Akhavian and Behzadan [Akh-2015].

In the case of a Kelly drilling machine, it is possible to identify three levels of detail (LoD). These are shown in Figure 6-2. In the first LoD (LoD 1), a distinction is only made between working and idle. The Working group contains all activities in which the drill is actively involved. The parent node Working contains several child nodes. These are the main process steps (see Section 3.1) of the production of a pile: Drilling, Reinforcing and Concreting. Within the idle node, a distinction can be made between downtime and secondary process time. These subgroups form the second LoD (LoD 2). The third LoD (LoD 3) subdivides the rough process steps into more detailed steps.

For each parent node (e.g. Working) a classifier is trained, which can classify among the child nodes (Concreting, Reinforcing and Drilling). As possible classifiers the above mentioned models are considered and investigated. The advantage of such approaches is the possibility to exploit the tree structure of the classes in order to reduce the number of classes considered by a classifier. Instead of directly classifying 27 activities (flat classification), which may require highly complex models which need to detect very subtle differences in the data, local classifiers need to distinguish a much smaller number of classes (only up to 7).

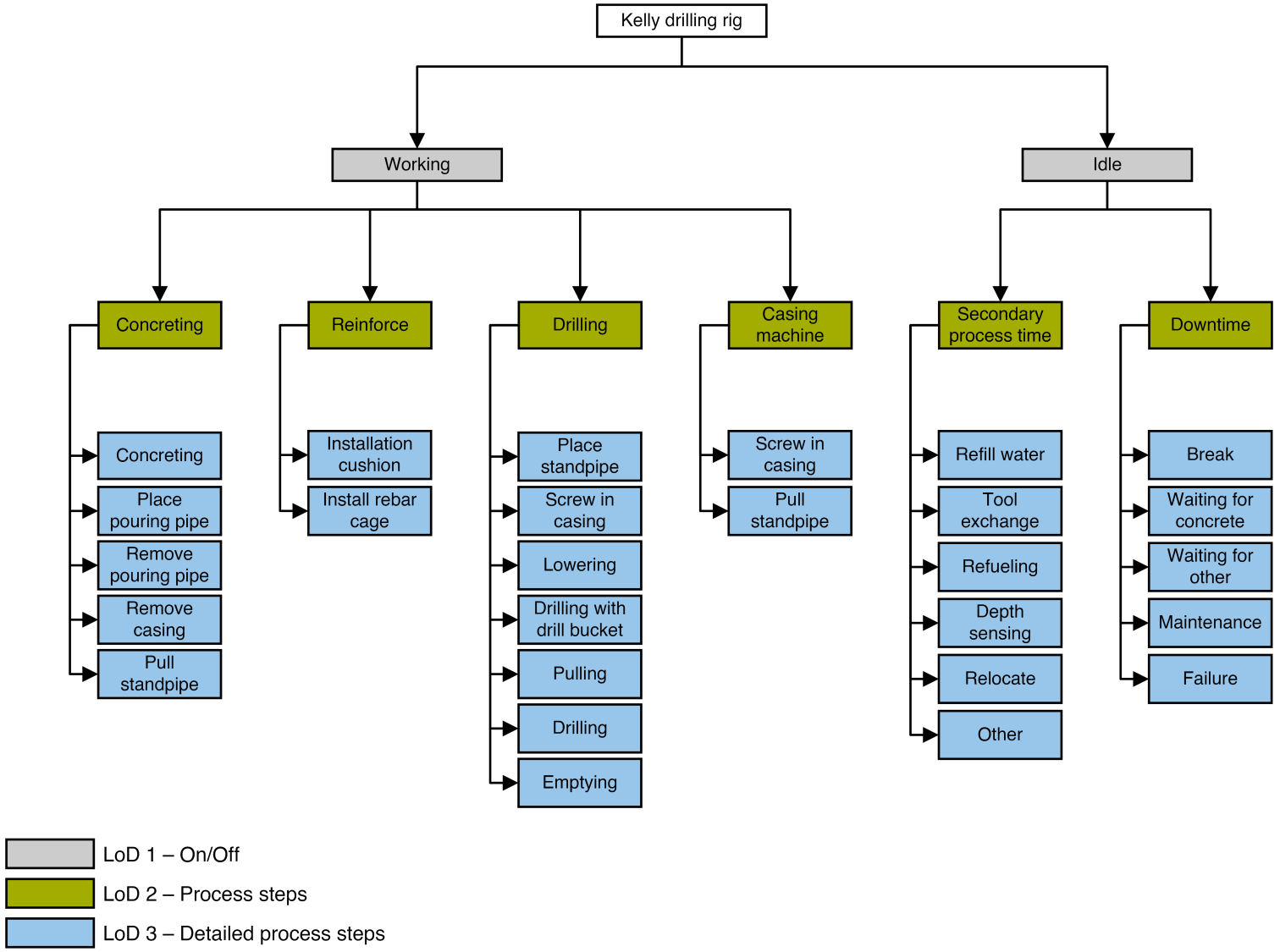


Figure 6-2: Levels of detail (LoD) for hierarchical classification

6.4 Implementation and training

The models were developed in Python using the *Keras* framework, which is based on the popular *Tensorflow* library. Keras has a modular structure, whereby models can be assembled based on different available layers and elements. To further facilitate the exploration, a function was created for each of the four proposed models that allows the models to be generated with the respective architecture depending on various parameters. The source codes are shown in Appendix A.3.

Since it is a multi-class classification problem, the categorical crossentropy is used as the loss function. The goal is to minimize the value of this function, which is the error of the model. The categorical crossentropy J is given by [Gér-2019]:

$$J = \frac{-1}{m} \sum_{i=1}^{m} \sum_{k=1}^{K} y_k^{(i)} \log(p_k^i) \tag{6-1}$$

where $y_k^{(i)}$ are the entries of the one-hot encodings, p_k^i are the output probabilities of the *softmax* layer (see Eq. 3-7), K is the number of classes, and m is the number of instances.

Table 6-1: Hyperparameters

Parameter	Value
Loss function	Categorical crossentropy
Optimizer	Adam
Epochs	50
Batch size	256
Learning rate	0.001

For the optimization of the weights of the models the *Adam* (Adaptive Moment Estimation) algorithm is employed. One of the advantages of this algorithm is that an adaptive learning rate control is already integrated. The learning rate selected for the training is 0.001. To allow a proper comparison of the models and to keep the computation time low, no further hyperparameter tuning was performed. Each model is trained for 50 epochs with a batch size of 256 instances. As training may also contain a random component and the results

may vary slightly, each training process is performed three times. The used hyperparameters are summarized in Table 6-1.

6.5 Performance analysis

In order to assess the quality of the models, the performance must be quantified. This may be accomplished by means of various available metrics. The most intuitive metric is the accuracy, which can be calculated for a particular class as follows:

$$Accuracy = \frac{TP + TN}{P + N} \tag{6-2}$$

where TN denotes the number of correctly negatively classified instances, TP denotes the number of correctly positively classified instances, P denotes the total number of positive instances, and N denotes the total number of negative instances. The main drawback of accuracy as a metric is the possible distortion of the results for imbalanced datasets. This property was demonstrated in Section 4.2 for the analyzed dataset. The key problem is, if a model would always predict the most frequent classes in a strongly imbalanced dataset, then an accuracy equivalent to the frequency of the most frequent class would already be achieved.

To counteract these issues, two other metrics can be employed. If the ratio of correctly positively predicted instances to the total number of positively predicted instances is considered, then the **precision** can be defined as a metric [Gér-2019]:

$$Precision = \frac{TP}{TP + FP} \tag{6-3}$$

where TN denotes the number of correctly negatively classified instances, TP denotes the number of correctly positively classified instances and FP denotes the number of wrongly positively classified instances. While the precision focuses primarily on the positive predictions, the **recall** considers the positive instances present in the dataset in general. The basic idea behind the metric is to consider how often the positive instances that are present

in the dataset are also classified as such. The mathematical description is given by [Gér-2019]:

$$Recall = \frac{TP}{TP + FN} \tag{6-4}$$

where FN denotes the number of wrongly negatively classified instances.

To combine the properties of these two metrics simultaneously, the **F1 score** is introduced (Eq. 6-5).The F1 score can assume values between 0 and 1. The closer to 1 the determined value, the better the performance of the model.

$$F_1 = 2 \cdot \frac{Precision \times Recall}{Precision + Recall} = \frac{TP}{TP + \frac{FN+FP}{2}} \tag{6-5}$$

The metrics introduced so far are valid only for a specific class. Should a value be calculated for the entire data set, then the mean value of the metrics of the individual classes is used. In the scope of this thesis, primarily the accuracy in connection with the F1 score is used.

Part III

Evaluation

7 Results

This chapter presents the results of the experiments proposed in Part 2. First, the performance of each model is compared, both during the training process and the application with the test set. In the second part, the results of the different parameter studies are presented. Finally, in the last part of the chapter, the results of the hierarchical classification are shown. To facilitate the overview, the parameters to which the results of the respective sections refer are indicated at the beginning of the sections within a gray box. The emphasis of the respective investigation is indicated in bold.

7.1 Models

Models	**MLP, LSTM, DeepConvLSTM, DeepConvBiLSTM**
Window size	16 seconds
Overlap	OW
Splitting method	Random

7.1.1 Training

As a first investigation of the performance of the models, the training procedures are analyzed. As the different models have different degrees of complexity, the number of parameters to be trained also varies. The larger the number of parameters, the longer the training process takes. The number of parameters and the duration of the training for the four models are shown in Table 7-1. A difference in the number of parameters between the various models is clearly visible. Although the model consisting of LSTM layers has less than half the number of layers of the MLP, it has approximately four times as many parameters. As a result, the training time is also longer. Adding the three CNN layers for the *DeepConvLSTM*, on the other hand, only results in a significantly smaller increase in the number of parameters. Since the number of LSTM cells is doubled when bidirectional LSTMs are used, another significant increase can be seen. The training time amounts to approx. 15 minutes with the use of a GPU, which is within the tolerable range for a data set of this size.

Table 7-1: Duration of the training

Model	Trainable parameters	Training duration [sec/Epoch]
MLP	55,194	2
LSTM	209,690	9
DeepConvLSTM	262,170	13
DeepConvBiLSTM	626,970	17

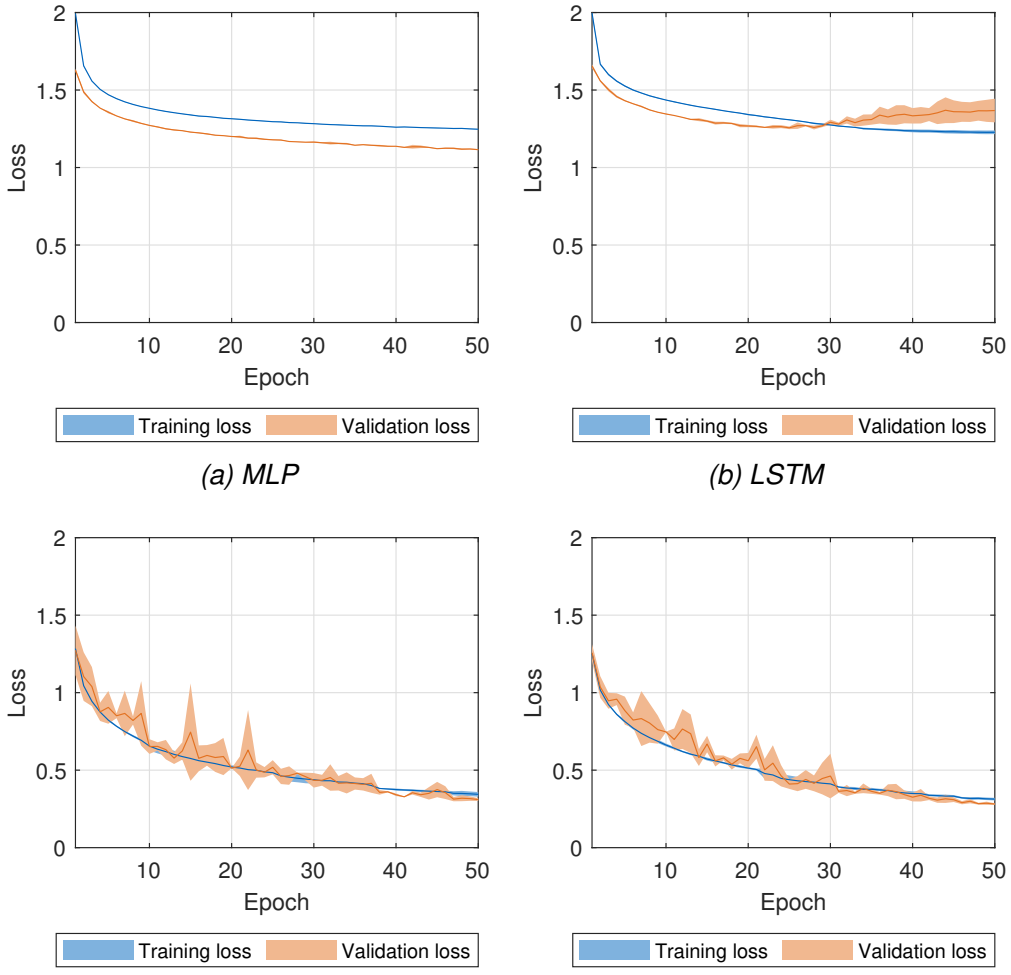
To assess the ability of the models to accurately model the process, the loss during training is considered first. Figure 7-1 shows the curves of the loss function depending on the number of epochs for the different models. Since the training was performed three times, the mean and standard deviation were calculated. The solid lines represent the mean, while the colored bands in the background represent the standard deviation. Additionally, the training and validation accuracies of the different models are shown in Table 7-2.

Table 7-2: Training and validation accuracy for the proposed models

Model	Training Accuracy	Validation Accuracy
MLP	60.85 %	64.03 %
LSTM	61.39 %	57.88 %
DeepConvLSTM	87.13 %	88.52 %
DeepConvBiLSTM	88.14 %	89.59 %

The two upper figures show the loss for the MLP (Fig. 7-1a) and for the LSTM model (Fig. 7-1b). For both cases, it is visible that the loss of the training data set clearly converges towards a constant value. Thus, further training of the model would not lead to a significant increase in performance. However, if the validation loss is considered, then some differences are visible. In the case of the MLP, the validation loss follows the course of the training loss, but shifted downwards. This is a consequence of the high dropout rate. In the case of the LSTM model, an increase of the validation loss can be seen from epoch 30, while the training loss continues to decrease. This is a clear signal of the overfitting of the model (see Section 3.2.1). A further indication of this is also the smaller validation accuracy compared to the training accuracy.

The two bottom figures show the loss for the *DeepConvLSTM* (Fig. 7-1c) and the *DeepConvBiLSTM* (Fig. 7-1d). As for the two baseline models, the training loss converges to a fixed value. However, the value is significantly lower than the final values of the baseline



(a) MLP *(b) LSTM*

(c) DeepConvLSTM *(d) DeepConvBiLSTM*

Figure 7-1: Training and validation loss for different models

models. The validation loss of the two hybrid models roughly follows the course of the training loss for both models. Furthermore, a larger spread of the values of the validation loss is visible, which mainly originates from the beginning of the training and which decreases in the course of the training. In contrast to the baseline LSTM model, where the variance increases towards the end of the training, the variance of the validation loss of the two hybrid models is very low. In summary, it is possible to state that the two hybrid models do not face an overfitting problem. Since for both models high accuracies could be achieved, it can also be argued that for the hybrid models, in contrast to the baseline models, no underfitting takes place.

7.1.2 Test

Although estimations of the quality of the models can be made through training and validation performance, the actual verification of the quality is conducted by examining the performance on the test set. The F1 score introduced in Section 6.5 is used as a metric, as simply considering the accuracy in imbalanced datasets is not recommended. Table 7-3 shows the average F1 score of the respective models. In addition to the normal mean, a weighted average can be formed, which considers the number of respective labels and uses them as weights. The weighted mean can be considered an optimistic estimate since the more frequent labels are weighted more heavily, which is not optimal. Furthermore, the F1 scores of the individual labels for the different models are specified in the Table 7-4.

Table 7-3: Averaged F1-Score for the proposed models

Model	Avg. F1-Score	Weighted Avg. F1-Score
MLP	0.49	0.62
LSTM	0.37	0.52
DeepConvLSTM	0.87	0.89
DeepConvBiLSTM	0.89	0.91

The MLP model, which achieves a validation accuracy of 64.03 %, only achieves an F1 score of 0.49 when tested. If only the labels which can be assigned to the process step **Drilling (LoD 2)** and which have been considered in previous studies (**Lowering**, **Drilling (LoD 3)**, **Emptying**, **Screw in casing** and **Pulling**) are considered from Table 7-4, then higher than average F1 scores can be observed. The average F1 score of these five activities is 0.73, which, nevertheless, remains far below the scores obtained in previous studies [Bi-2020, Ors-2020], all of which are above 0.87. The reason for this deterioration is most likely the absence of strong filtering (removal of outliers and smoothing of the data) of the data in this study. It is evident that a model consisting of MLP does not have such strong modeling capabilities for the process examined here. A further aspect to consider is the impossibility of considering only these five activities separately, since in practice they may be mixed with the rest of the activities and not occur separately.

Another aspect which can be identified during testing is the overfitting of the LSTM model, which had already been mentioned in the previous section. Although theoretically the model

Table 7-4: F1 values per label for all models

Label	MLP	LSTM	DeepConvLSTM	DeepConvBiLSTM
Break	0.64	0.48	0.95	0.96
Failure	0.94	0.91	0.99	0.99
Waiting for concrete	0.42	0.21	0.96	0.95
Waiting for other	0.53	0.07	0.94	0.94
Lowering	0.62	0.67	0.85	0.87
Concreting	0.51	0.24	0.83	0.86
Install cushion	0.73	0.20	0.92	0.94
Drilling	0.79	0.80	0.87	0.89
Drilling with bucket	0.00	0.00	0.74	0.81
Install rebar cage	0.56	0.44	0.93	0.94
Emptying	0.69	0.76	0.93	0.95
Remove pouring pipe	0.22	0.13	0.79	0.73
Place pouring pipe	0.53	0.04	0.92	0.93
Place standpipe	0.74	0.57	0.92	0.94
Pull standpipe	0.31	0.13	0.83	0.85
Remove casing	0.15	0.07	0.79	0.82
Screw in casing	0.84	0.86	0.94	0.96
CM - Pull standpipe	0.47	0.44	0.73	0.87
CM - Screw in casing	0.43	0.25	0.85	0.88
Pulling	0.72	0.73	0.86	0.88
Other	0.10	0.21	0.83	0.85
Refueling	0.56	0.00	0.83	0.83
Depth sensing	0.00	0.00	0.64	0.70
Relocate	0.30	0.30	0.94	0.95
Refill water	0.72	0.74	0.86	0.89
Tool exchange	0.27	0.26	0.90	0.92

complexity is higher, the LSTM model achieves an average F1 score of only 0.37. If, however, only the five activities mentioned above are considered, then an improvement of the results to an average F1 score of 0.76 is actually noticeable. The LSTM layers prove to be an acceptable mechanism to model these activities, which however happens at the expense of the other labels, which experience a significant deterioration. For example, the F1 score of the **Place pouring pipe** class drops from 0.53 to 0.04. This may be, among other issues, a symptom of the class imbalance of the data set. The five activities combined amount to about 40 % of the available data, while other 20 activities amount to only 36 % of the data.

Both hybrid models (*DeepConvLSTM* and *DeepConvBiLSTM*) show a significantly higher average F1 score, exceeding 0.85. For the *DeepConvLSTM* model, the mean F1 score is 0.87 and the weighted F1 score is 0.89. For the *DeepConvBiLSTM* model, the F1 score is 0.89 and 0.91, respectively. Between the MLP/LSTM and *DeepConvLSTM* models, a significant increase of the F1 score is evident for all labels. Between the *DeepConvLSTM* and the *DeepConvBiLSTM*, a further weaker increase and improvement is subsequently evident. The average F1 score of the five activities previously mentioned increases to 0.89, which corresponds to a reduction of the error by more than 50 %. In Table 7-5, the average F1 score for the five activities (**Lowering**, **Drilling (LoD 3)**, **Emptying**, **Screw in casing** and **Pulling**) is shown along with the variation of the error with respect to the previous model.

Table 7-5: F1 values and variation for all models - Only Drilling (LoD 2)

Model	Avg. F1-Score - Drilling (LoD 2)	Error variation
MLP	0.73	-
LSTM	0.76	-11 %
DeepConvLSTM	0.89	-54.2 %
DeepConvBiLSTM	0.91	-18.9 %

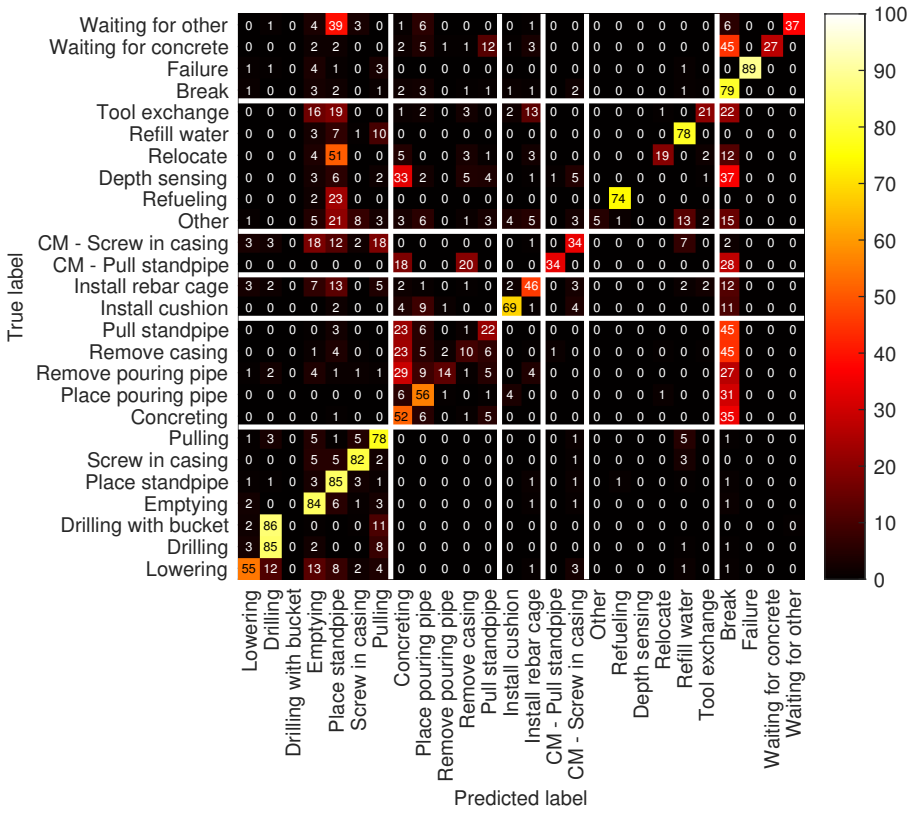
Of further interest is the increase of the F1 score for the label **Lowering**, which in previous work [Ors-2020] exhibited the lowest F1 score, from 0.67 (LSTM) to 0.85 (*DeepConvLSTM*) and 0.87 (*DeepConvBiLSTM*), respectively, and for the label **Emptying** from 0.76 to 0.93. These are similar to the results obtained in [Ors-2020], although no filtering of the data was performed and all activities are considered. In addition to the increase in F1 scores

for these five activities, the hybrid models are notable for the significant improvement in the predictions of the other class, which represent only a much smaller fraction of the total data set. For instance, the F1 score for the activity **Remove casing** increases from 0.15 (baseline models) to 0.82 (*DeepConvBiLSTM*). Similarly, the classes **Relocate** (0.3 to 0.95) and **Concreting** (0.51 to 0.86) also experience significant improvements.

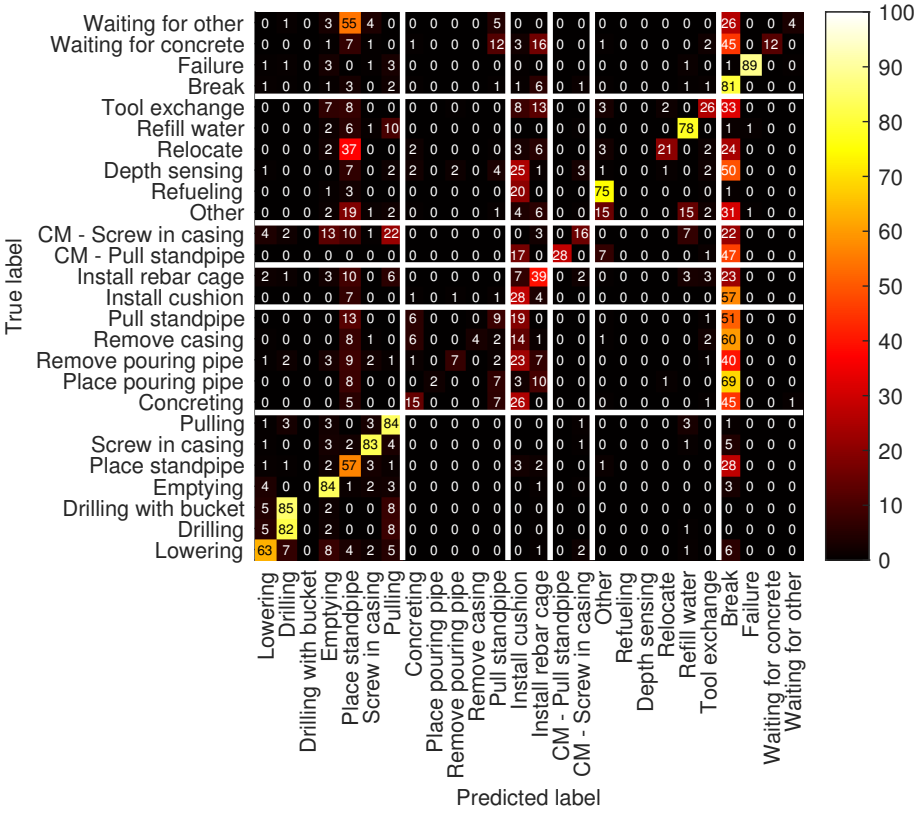
In order to obtain a more detailed insight into the performance of the models, the confusion matrices for the predictions of the *DeepConvLSTM* (Fig .7-2c) and *DeepConvBiLSTM* (Fig. 7-2d) models for the test set are shown in Figure 7-2. Since a large number of activities were considered, the confusion matrices have a large number of entries. For the sake of clarity and due to the high complexity, only a few selected aspects of the confusion matrix are explained in detail.

For both the baseline MLP and baseline LSTM models, the column of samples classified as **Break** draws a considerable attention. Except for the activities that can be assigned to the process step **Drilling (LoD 2)**, nearly all activities are mistaken for the activity **Break** at some level. This behavior is also found to occur with the LSTM model. Based on the confusion matrix of the LSTM model, it can be observed that the activity's low F1 values for the classes **Concreting**, **Place pouring pipe**, **Remove pouring pipe**, **Remove casing** and **Pull standpipe** mainly result from confusion with the classes **Install cushion** and **Break**.

The confusion matrix of the two hybrid models shows a more pronounced diagonality of the matrix. The majority of the entries are on the diagonal, which implies that an overall high accuracy has been achieved. Most noticeable, as in the MLP model, are the many entries of the column of the class **Concreting** as predictions. There is a mix-up especially between the activities of the process step **Concreting**. For both hybrid models, the activity **Depth sensing** is the activity with the lowest accuracy.Another activity which is often misclassified is **Drilling with bucket**, which is mainly confused with the activity **Drilling**. Apart from these mentioned anomalies, the hybrid models reach high overall accuracies.

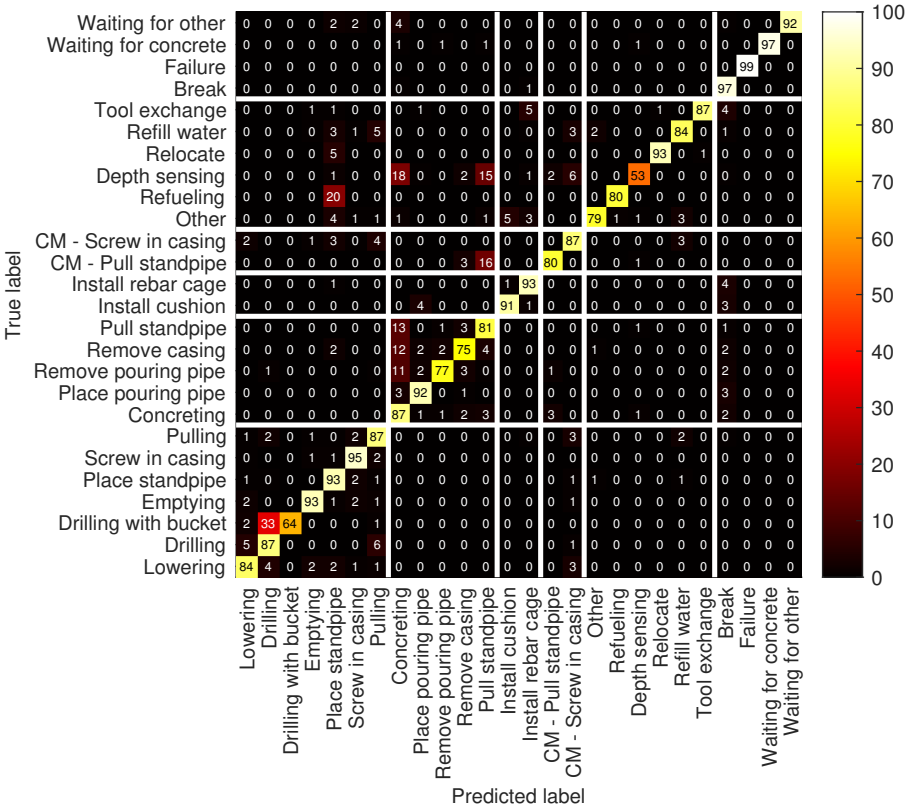


(a) MLP

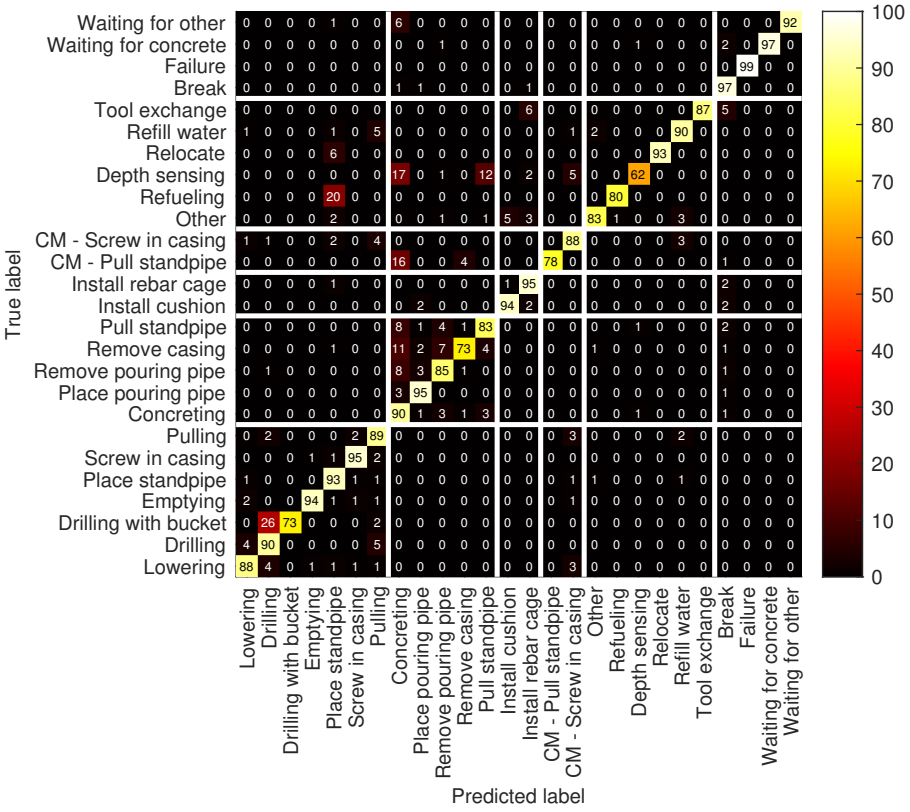


(b) LSTM

Figure 7-2: Confusion matrices - Random splitting



(c) DeepConvLSTM



(d) DeepConvBiLSTM

Figure 7-2: Confusion matrices - Random splitting

7.2 Sensitivity analysis

Based on the results of the previous section, this section presents the results of the investigations on the influences of different parameters on the performance of the models. Since the baseline LSTM models were shown to be inadequate and exhibit signs of overfitting, they were not considered further for the parameter studies.

7.2.1 Window size

Models	DeepConvLSTM, DeepConvBiLSTM
Window size	**8,16,32,48,60 seconds**
Overlap	OW
Splitting method	Random

The first parameter to be investigated was the influence of the window size during segmentation. The average F1 scores for the different window widths are shown for the *DeepConvLSTM* and *DeepConvBiLSTM* models in Figure 7-3.

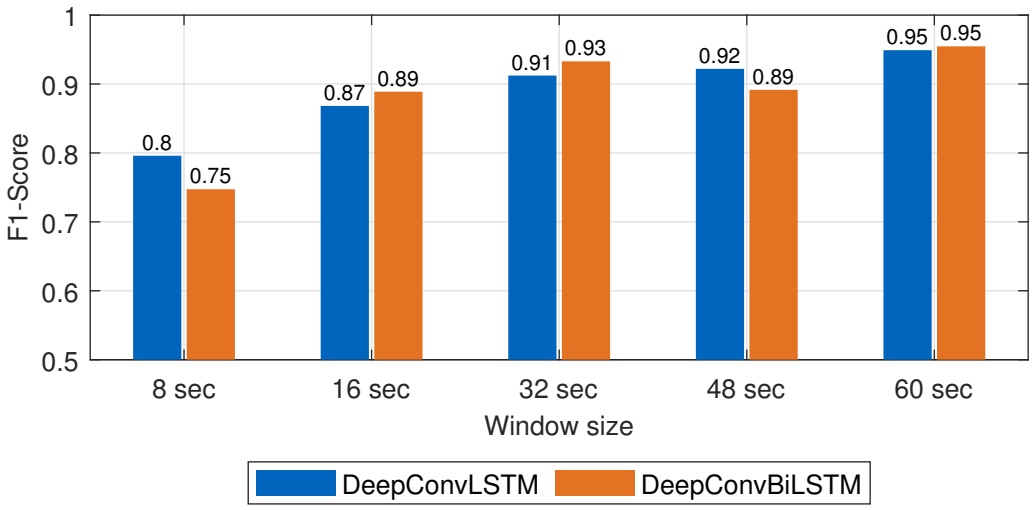


Figure 7-3: Variation of the F1 score as a function of the window width

It can be stated that an increase in the window width also results in an improvement in the performance of the models. The only exception to this is the degradation in performance of the *DeepConvBiLSTM* model at a window width of 48 seconds, which is most likely due to a problem during the optimization of the model. Furthermore, it is also visible that when the

window size is reduced from 16 to 8 seconds, there is a significant drop in performance for both models.

Basically, the improvement in performance can be explained by the nature of the models of both hybrid models. The larger the window size, the larger the temporal context and the amount of information the models have available to model the behavior. However, it is also important to note that increasing the window size also increases the number of trainable parameters and ultimately the training time of the models. Table 7-6 shows the increase in training time as a function of window size. Depending on the size of the dataset and the chosen window size, training the model may require very large training time and/or resources, which may be disadvantageous. There is a tradeoff between the performance of the model and the resources needed for training.

Table 7-6: Required time for processing an epoch of the DeepConvLSTM model depending on the window size

Window size	Training duration [sec/Epoch]
8	3
16	8
32	15
48	28
60	41

Another point that must be considered during the selection of the window size is the future application of the model. If the model is to be used in real time, a high window size may cause a number of shortcomings. Since the sensor measurements must always be segmented, the model has a delay of, at least, the selected window size. If a window size of 60 seconds is chosen, the activity of the machine will only be classified 1 minute after the performed activity. In addition, the processing time of the model is also longer. For some applications this lag may be too large. Yet, if the activity detection is only to be performed afterwards, e.g., to be used as input of a simulation, this lag is negligible.

7.2.2 Overlap

Models	DeepConvLSTM, DeepConvBiLSTM
Window size	16 seconds
Overlap	**OW, SNOW, FNOW**
Splitting method	Random

The second effect studied is that of the degree of overlap of two neighboring samples during segmentation. In particular, three possibilities (OW, SNOW, and FNOW) were investigated. Figure 7-4 shows the average F1 score for both models for the respective degrees of overlap.

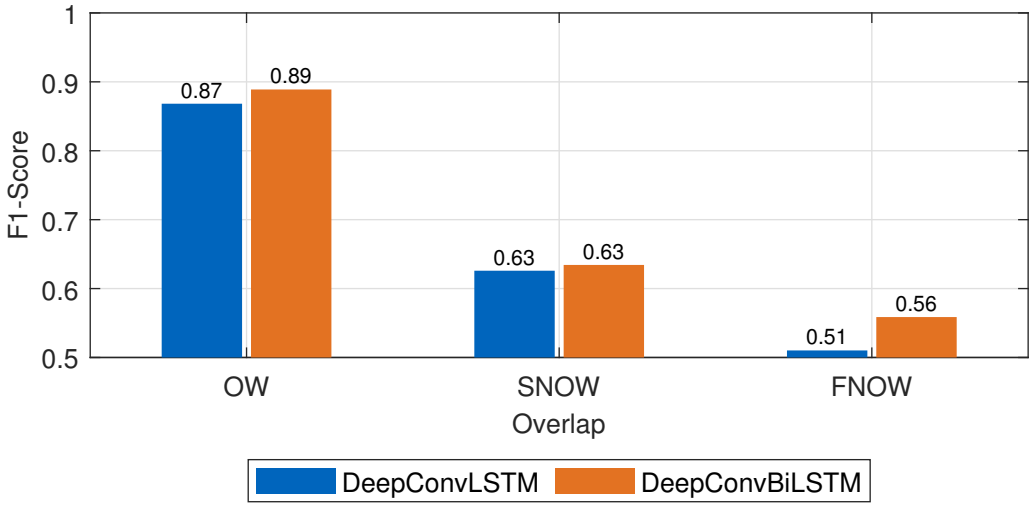


Figure 7-4: Variation of the F1 score as a function of the overlap

Based on the plot, it is possible to observe that a significant difference exists between the results of the different degrees of overlap. Both the *DeepConvLSTM* and the *DeepConvBiLSTM* models deteriorate by about 25 % when the samples overlap by only 50 % instead of fully overlapping. If the samples do not overlap at all, the F1 score drops even further to a low 0.51 in the case of the *DeepConvLSTM*.

This sharp drop can arise from two possible causes. As described in Section 3.3, when using the OW method, a high bias is introduced to the model due to the possible high similarity of neighboring samples. This bias can increase performance in the dataset under consideration, but is a disadvantage when the model is to be applied to a new dataset.

Both the SNOW and FNOW methods add a significantly lower bias to the model, but this degrades the performance of the model.

The second possible reason that leads to the degradation is the number of samples generated. To examine this in more detail, the number of samples generated by each method is shown in Table 7-7.

Table 7-7: Number of generated samples depending on the overlapping degree

Method	Number of training samples
Overlapping window (OW)	220866
Semi non-overlapping window (SNOW)	27608
Fully non-overlapping window (FNOW)	13804

A consequence of the smaller overlap of samples is also the overall reduction in the number of available samples. In the case of the OW method, more than 220 000 samples were available. If the overlap is reduced to 50 %, then the amount is reduced by almost 10 times. If the samples are not supposed to overlap at all, then out of the more than 350 000 measured data points, only 13 804 samples are available for training. This significant reduction in the size of the training data set can lead to problems, since in general Deep Learning approaches rely on a very large amount of available data to achieve good results. In order to determine whether the degradation is due to the high bias or the lower sample number, it would be appropriate for future studies to test the same models on a larger dataset.

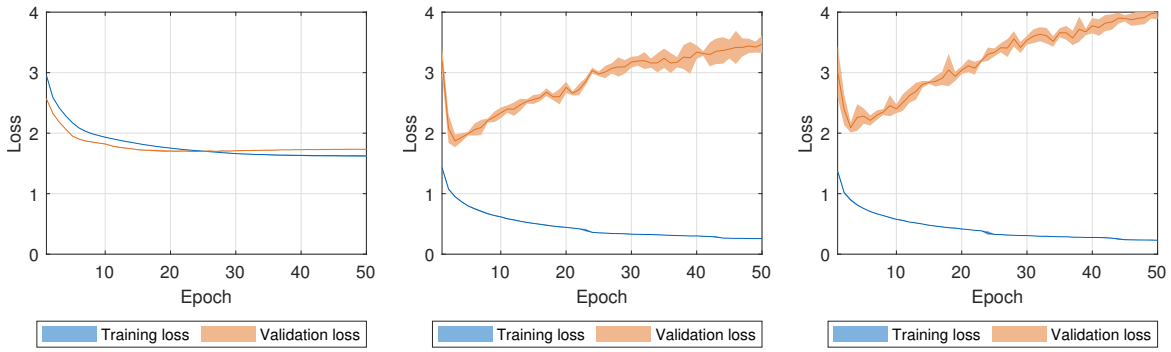
7.2.3 Splitting method

Models	DeepConvLSTM, DeepConvBiLSTM
Window size	16 seconds
Overlap	OW
Sampling method	Last
Splitting method	**By days**

An important aspect to be investigated is the generalization capability of the considered models. In order to examine this in more detail, the selected model architectures are tested on disjoint training and test sets with respect to the days of data. An important aspect to be investigated is the generalization capability of the considered models.

Training

Since a new splitting of the data is considered in this section, the training process of the models is examined again to ensure the convergence of the models. Initially, the exact same models are used with the identical settings as in Section 7.1. The training and validation losses of the models ares shown in Figure 7-5.

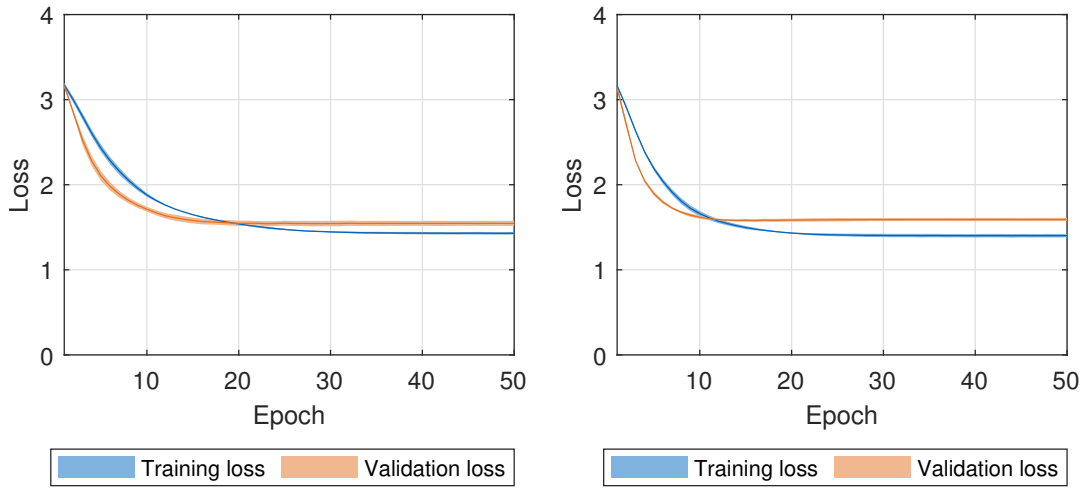


(a) MLP *(b) DeepConvLSTM* *(c) DeepConvBiLSTM*

Figure 7-5: Training and validation loss for different models - Splitting by days

The history of the training process of the MLP model (Fig. 7-5a) is very similar to the history from Section 7.1.1. Both training and validation clearly converge to a fixed value. There is no sign of overfitting occurring. If, on the other hand, Figures 7-5b and 7-5c are considered, a clear difference can be seen compared to Section 7.1.1. While the training loss steadily decreases and converges to a low value, for both the *DeepConvLSTM* models as well as for the bidirectional variant an increase of the validation loss is visible starting at approximately the fifth epoch. It can be clearly stated that both hybrid models suffer from overfitting.

To counteract overfitting, the models are adapted. First, the dropout rate is increased to 50 %. Furthermore, the complexity of the model is reduced by using only 64 cells per layer instead of 128 LSTM cells. The learning rate is reduced from 10^{-3} to 10^{-5}. The adapted models are referred to as *DeepConvLSTM*$_{64}$ and *DeepConvBiLSTM*$_{64}$ in the rest of the section, while the previous models are referred to as *DeepConvLSTM*$_{128}$ and *DeepConvBiLSTM*$_{128}$. The training and validation losses of the adapted models are shown in Figure 7-6.



(a) DeepConvLSTM$_{64}$ *(b) DeepConvBiLSTM*$_{64}$

Figure 7-6: Training and validation loss for the adapted models - Splitting by days

Based on the plots, it is easily recognizable that the actions taken have an effect on the training. Both the training and the validation loss for both hybrid models converge and the mentioned overfitting could be prevented. However, as a consequence of these measures, the models do not reach such a low loss as shown in Figure 7-5. To take a closer look at this aspect, the training and validation accuracies of both the original (MLP, *DeepConvLSTM*$_{128}$ and *DeepConvBiLSTM*$_{128}$) and adapted models (*DeepConvLSTM*$_{64}$ and *DeepConvBiLSTM*$_{64}$) are shown in Table 7-8.

The training and validation accuracies of the *MLP* are lower than the accuracies obtained in Section 7.1.1 by approximately 10 %. The two hybrid models with 128 LSTM cells per layer achieve accuracies comparable to the previous original results of about 90 % on the training set. However, there is a significant drop when considering the validation accuracy, which reaches only 53.75 % and 51.90 %, respectively. This indicates that the model has the ability to replicate the training set data very well, but has difficulty generalizing to new

Table 7-8: Training and validation accuracy for the proposed models - Splitting by days

Model	Training Accuracy	Validation Accuracy
MLP	51.83 %	54.47 %
DeepConvLSTM$_{128}$	90.34 %	53.75 %
DeepConvBiLSTM$_{128}$	91.31 %	51.90 %
DeepConvLSTM$_{64}$	58.53 %	56.50 %
DeepConvBiLSTM$_{64}$	58.38 %	55.77 %

data. For the two adapted hybrid models, the overfitting is no longer evident. The training accuracy roughly equals the validation accuracy. However, it is possible to detect that a significant drop in training accuracy is required as a tradeoff for this.

Test

In order to further test the models, data from 7 additional days are analyzed. The average F1 scores obtained from the application of the five models discussed above to the test set are shown in Table 7-9.

Table 7-9: Averaged F1-Score for the proposed models - Splitting by days

Model	Avg. F1-Score
MLP	0.25
DeepConvLSTM$_{128}$	0.31
DeepConvBiLSTM$_{128}$	0.27
DeepConvLSTM$_{64}$	0.26
DeepConvBiLSTM$_{64}$	0.26

All the models exhibit a significant drop in the performance of the models compared to the training, with a maximum F1 score of 0.31. The MLP, which did not present any problems regarding overfitting, possesses the lowest F1 score of 0.25. The two original hybrid models (*DeepConvLSTM*$_{128}$ and *DeepConvBiLSTM*$_{128}$) achieve the highest F1 scores among the five models with 0.31 and 0.25, respectively. This occurs despite the fact that both models face a clear overfitting issue. The models adapted to avoid overfitting (*DeepConvLSTM*$_{64}$ and *DeepConvBiLSTM*$_{64}$) each achieve an F1 score of only 0.26. Considering only the average F1 score, no significant differences between the five models are apparent. Likewise,

the adaptions to avoid overfitting, which according to Table 7-8 led to a weak increase in validation accuracy, do not increase the average F1 score.

It is important to note that the F1 scores presented above represent only the average of the individual classes. Therefore, it is possible that the F1 scores of the individual classes are unevenly distributed and thus the mean value is negatively influenced. In order to obtain deeper insights into the performance of the models, the F1 scores of the models for the individual labels are examined, as in Section 7.1.2. These are shown in Table 7-10.

Based on their F1 scores, the labels can essentially be divided into three different groups. Besides a limited number of exceptions, these divisions apply to all models. The first group consists of activities with a high F1 score (F1> 0.5). This group consists of the activities **Lowering**, **Drilling**, **Emptying**, **Place standpipe**, **Screw in casing**, **Pulling** and **Refill water**. Except for the last activity, which is classified as secondary process time, the remaining ones belong to the process step **Drilling (LoD 2)**. Within this group, a small impact of the model architecture and overfitting prevention measures on the performance can be seen. The F1 value of the adapted hybrid models is generally higher or at least as high as that of the original models (MLP, *DeepConvLSTM*$_{128}$ and *DeepConvBiLSTM*$_{128}$) for all activities in the group. If the unidirectional is compared with bidirectional variant, it can be argued that the influences are label dependent. While for some activities the results of the bidirectional variant are better, for others the unidirectional variant yields better results.

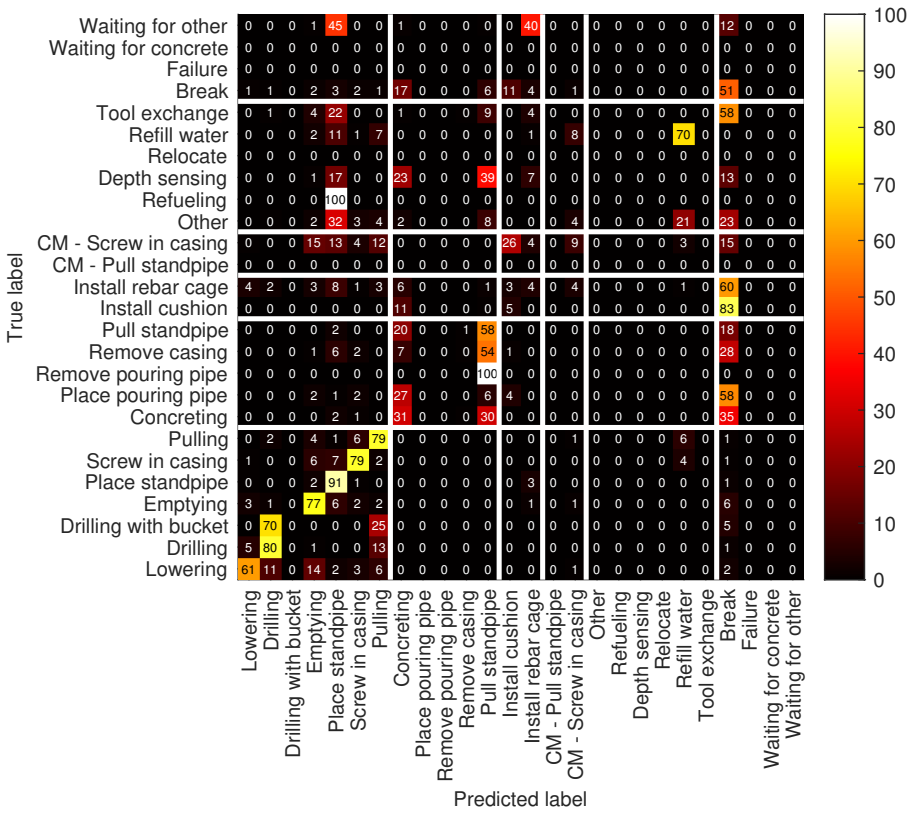
The second group consists of activities that are not recognized at all or only very poorly and feature a very low F1 score (F1< 0.1). This group comprises the activities **Failure**, **Waiting for concrete**, **Waiting for other**, **Install cushion**, **Drilling with bucket**, **Place pouring pipe**, **Remove pouring pipe**, **Casing machine - Pull standpipe**, **Other**, **Refueling**, **Depth sensing** and **Relocate**. Two additional activities which could possibly be assigned to this group are **Remove casing** and **Tool exchange**. These exhibit values above 0.1 for the *DeepConvLSTM*$_{128}$ and *DeepConvBiLSTM*$_{128}$ models, but values very close to zero for the MLP and adapted hybrid models. The labels **Break**, **Concreting**, **Pull standpipe**, **Casing machine - Screw in casing** form the last group. All of them have low F1 scores, but these differ in magnitude from those of the second group.

Table 7-10: F1 values per label for all models - Splitting by days

Label	MLP	*DeepConvLSTM*$_{128}$	*DeepConvBiLSTM*$_{128}$	*DeepConvLSTM*$_{64}$	*DeepConvBiLSTM*$_{64}$
Break	0.39	0.38	0.38	0.37	0.41
Failure	0.00	0.00	0.00	0.00	0.00
Waiting for concrete	0.00	0.00	0.00	0.00	0.00
Waiting for other	0.00	0.00	0.00	0.00	0.00
Lowering	0.66	0.66	0.66	0.67	0.68
Concreting	0.27	0.22	0.27	0.32	0.23
Install cushion	0.05	0.02	0.04	0.04	0.02
Drilling	0.80	0.80	0.76	0.80	0.82
Drilling with bucket	0.00	0.00	0.00	0.00	0.00
Install rebar cage	0.06	0.39	0.36	0.10	0.10
Emptying	0.71	0.70	0.71	0.78	0.74
Remove pouring pipe	0.00	0.00	0.00	0.00	0.00
Place pouring pipe	0.00	0.02	0.02	0.05	0.09
Place standpipe	0.72	0.70	0.67	0.75	0.73
Pull standpipe	0.34	0.23	0.24	0.22	0.26
Remove casing	0.01	0.26	0.15	0.03	0.00
Screw in casing	0.83	0.85	0.84	0.88	0.87
CM - Pull standpipe	0.00	0.00	0.00	0.00	0.00
CM - Screw in casing	0.12	0.21	0.21	0.21	0.20
Pulling	0.76	0.74	0.74	0.74	0.78
Other	0.00	0.05	0.03	0.00	0.00
Refueling	0.00	0.00	0.00	0.00	0.00
Depth sensing	0.00	0.00	0.00	0.00	0.00
Relocate	0.00	0.00	0.00	0.00	0.00
Refill water	0.68	0.62	0.67	0.74	0.75
Tool exchange	0.00	0.10	0.17	0.00	0.00

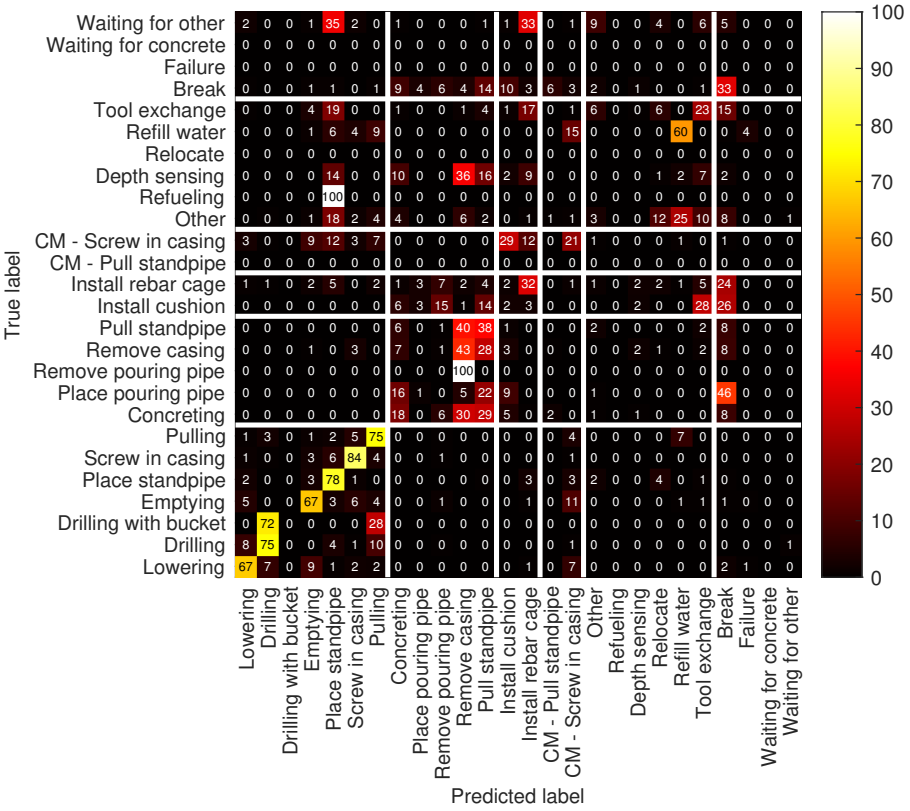
Furthermore, the confusion matrix of the predictions can be used to identify any mix-ups which may be taking place. The confusion matrices for the baseline MLP model, the *DeepConvLSTM*$_{128}$ and the *DeepConvLSTM*$_{64}$ model are given in Figure 7-7. As just stated, the performances of the unidirectional and bidirectional variants are very similar. Hence, only the confusion matrix for the unidirectional variant is shown here. In the following, a few points will be discussed in more detail.

In contrast to Figure 7-2, where in general most entries are located on the diagonal, this phenomenon is limited only to the lower left square, which corresponds to the process step **Drilling (LoD 2)**. Another interesting aspect is that samples associated with one of the labels of the process step **Drilling (LoD 2)** are mainly confused with labels from the same process. Almost no confusion with other labels is observed.

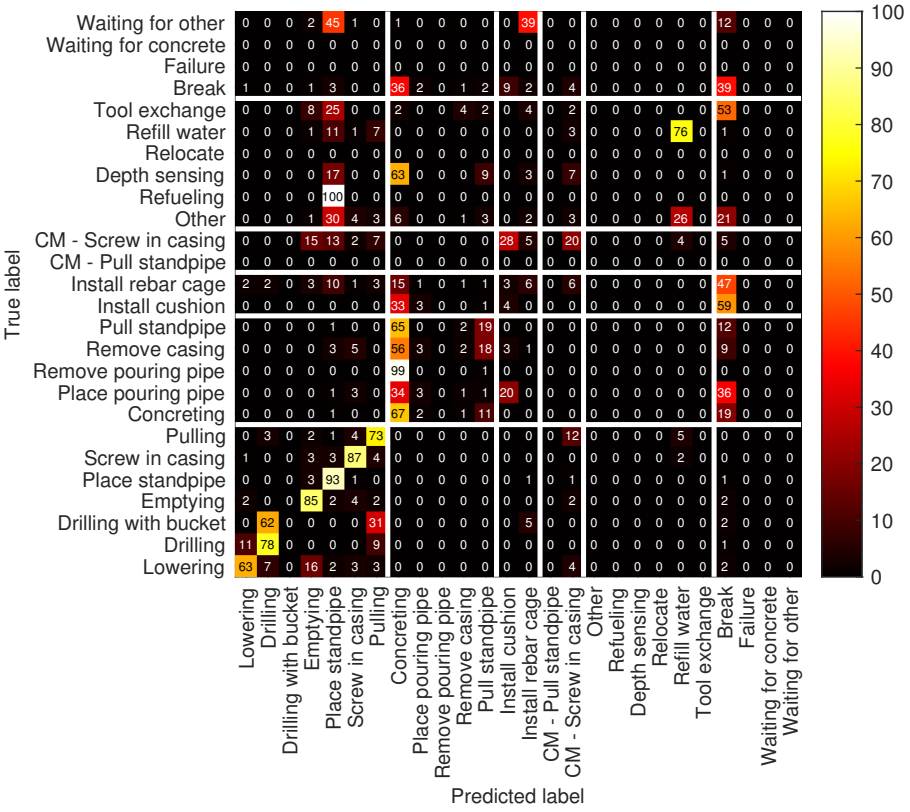


(a) MLP

Figure 7-7: Confusion matrices - Splitting by days



(b) DeepConvLSTM$_{128}$



(c) DeepConvLSTM$_{64}$

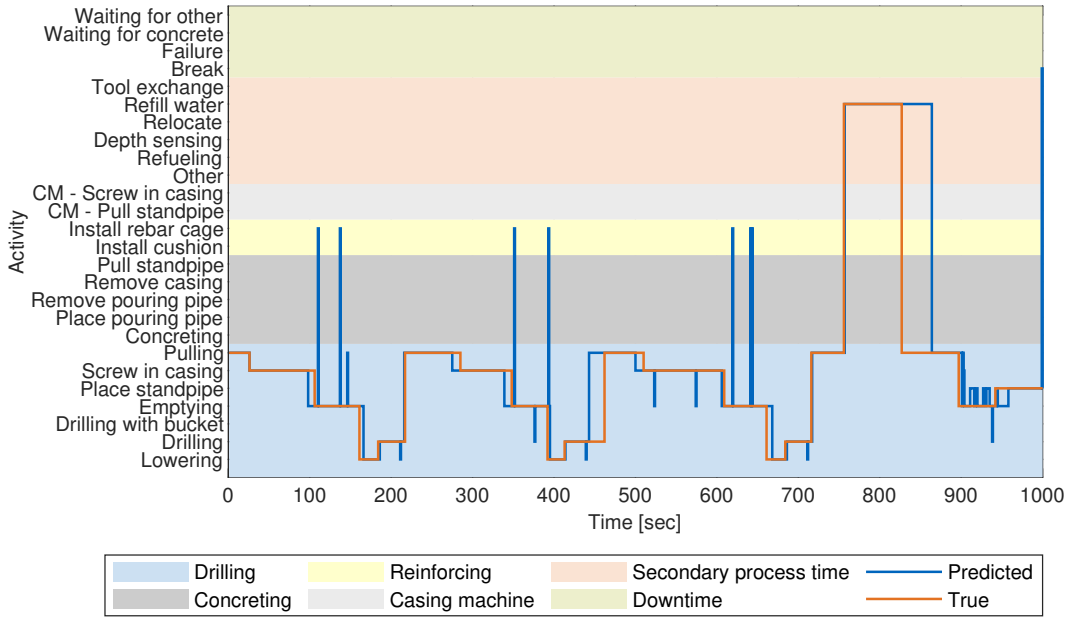
Figure 7-7: Confusion matrices - Splitting by days

On the other hand, numerous samples not belonging to this group are classified as such. Especially noticeable is a confusion between labels of the group of secondary processes (e.g. **Tool exchange**, **Refueling** or **Other**) with the activity **Place standpipe**. This is likely due to the imbalance of the dataset, since models trained on imbalanced datasets tend to output the most frequently occurring class as the prediction, and **Place standpipe** is the most common class, accounting for about 11 % of the dataset.

If the labels **Install rebar cage** and **Install cushion** or the labels **Casing machine - Screw in casing** and **Casing machine - Pull standpipe** are considered, it is recognizable from Figure 7-7 that for all three models they are poorly detected. Likewise, out of the secondary processes (**Tool exchange**, **Refill water**, **Relocate**, **Depth sensing**, **Refueling** and **Other**), all except **Refill water** are very poorly recognized.

Noteworthy is also the confusion of several activities with the class **Break**. Worth mentioning are the confusions with the classes **Tool exchange**, **Other**, **Install rebar cage**, **Install cushion**, **Place pouring pipe** and **Concreting**. The label **Concreting** is also mistaken for a lot of activities. Regarding these confusions, two considerations can be formulated to provide an explanation. The first is the high incidence of the Break class in the dataset. The second explanation is the inherent nature of the confused activities. During the activities mentioned above, the drill is not directly involved in the process and is just passively waiting until the process step is finished. As different labels were used for the same "waiting" of the machine, which describe the process rather than the behavior of the machine, further differentiating these activities proves to be very difficult for the model. For example, during **Concreting** the machine is not directly involved, but only waits nearby until the drilled hole has been filled with concrete. Therefore, this is confused with the activity **Remove pouring pipe** (see Fig. 7-7c), which is carried out by employees and not by the machine.

This confusion with the **Break** class could already be detected for the MLP from Section 7.1 (see Fig. 7-2a). Both hybrid models, however, were able to model these weak and barely noticeable differences of these similar classes. Based on Fig. 7-7b it can be acknowledged that this good modeling does not generalize well to other days.



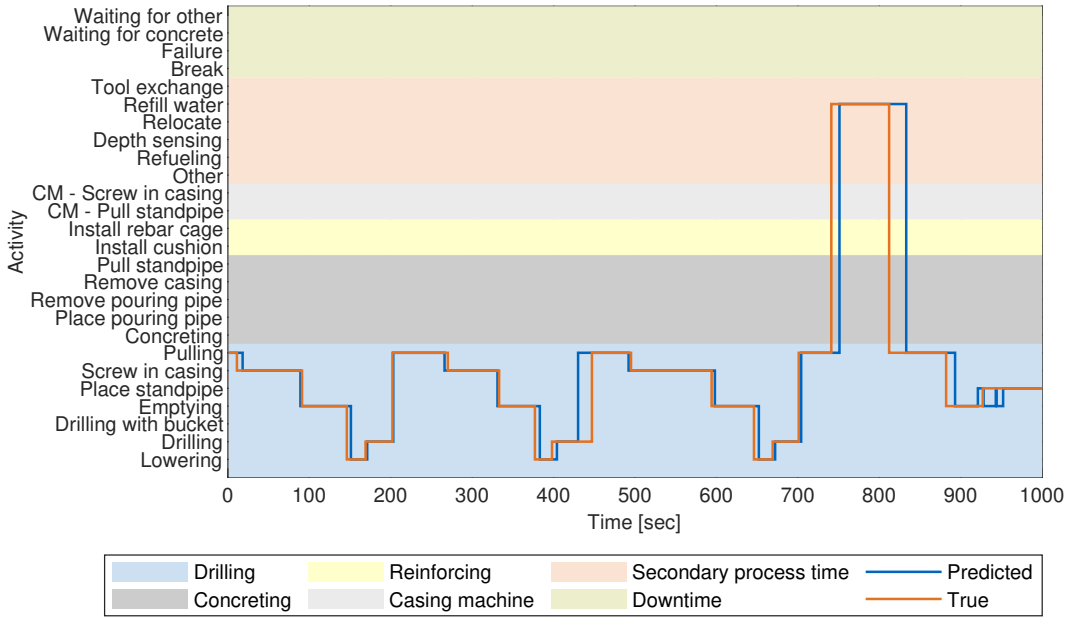
(a) MLP

Figure 7-8: True and predicted labels for a selected interval

One advantage of considering a continuous time window as a test set is the possibility to investigate the application of the model to several consecutive samples. In Figure 7-8 the predicted activity is depicted along with the real activity for a 1000 seconds time windows. One further example can be found in Appendix B-10.

Although only slight advantages of the hybrid models have been identified so far in this section in comparison to the baseline MLP, one additional advantage can be derived from Figure 7-8. While the MLP (Fig. 7-8a) provides a solution that fundamentally shows the correct course, several jumps of the predictions to wrong labels can be observed. In contrast, the solution provided by the *DeepConvLSTM*$_{64}$ model (Fig. 7-8b) exhibits a much smoother profile. In the 1000 seconds considered, no jump is visible at all. This behavior can also be observed in the other two examples.

Further, based on Figure 7-8, it can be stated that there is a certain shift in the predictions compared to the true labels. This might be due to the fact that the model needs a certain number of time steps as context to predict the activity correctly. Until there is enough context, the last activity is still output as the prediction.



(b) DeepConvLSTM$_{64}$

Figure 7-8: True and predicted labels for a selected interval

Another possible reason for the offset of the predictions is the uncertainty of the manual labelling. Since the labels were recorded manually, the labels depend on the judgement of the employee on the construction site. The exact boundaries of the activities depend on the subjectivity of the employees.

Overall, it can be said that there is a strong day dependency of individual activities. The activities, which could be modeled very well by the hybrid models in Section 7.1, could only be partially recognized if tested on data which originated from another day. Activities where the machine is actively involved could generally be recognized well, while other labels that mainly describe the process and do not contain any active involvement of the machine are often mistaken. This phenomenon will be considered in further detail in the next section.

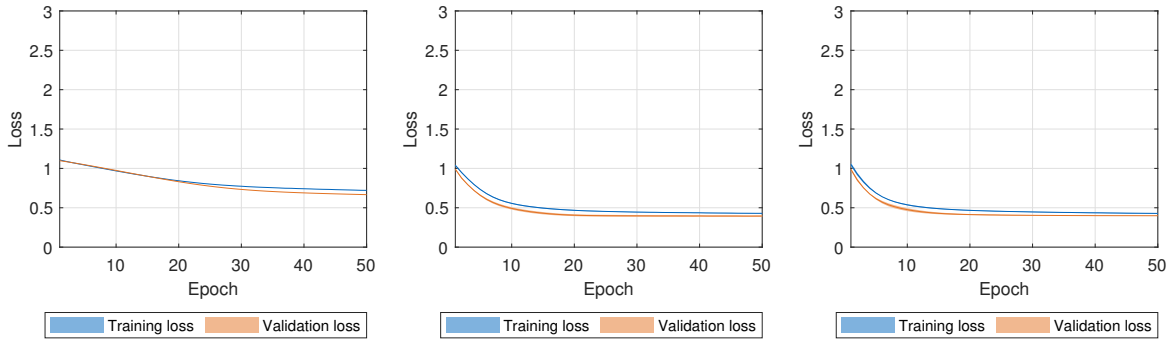
7.3 Hierarchical classification

Experiment 5	
Models	MLP, DeepConvLSTM, DeepConvBiLSTM
Window size	16 seconds
Overlap	OW
Splitting method	By days

This section presents the results of the activity recognition investigations considering different levels of detail. The investigations follow the subdivisions and levels of detail from Figure 6-2.

7.3.1 LoD 1 - Working/Idle

In the first level of detail considered, the activities are only divided into **Working** and **Idle**. Figure 7-9 shows the training process of the MLP, *DeepConvLSTM* and *DeepConvBiLSTM*. As can be seen in the plots, no overfitting of the models takes place, as both the training and validation loss converge.



(a) MLP *(b) DeepConvLSTM* *(c) DeepConvBiLSTM*

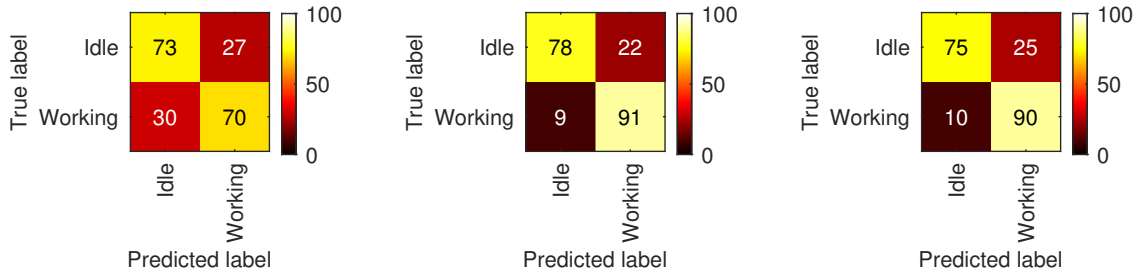
Figure 7-9: Training and validation loss for different models - LoD 1

The average F1 scores and the F1 scores for each class when applying the models to the test set are shown in Table 7-11. Both the baseline MLP and the two hybrid model variants have a similar mean F1 score of approximately 0.83. Among the individual labels, in the three cases, the **Working** label has a higher F1 score of approximately 0.86. Meanwhile, the F1 score for the label **Idle** is 0.80 for the MLP and the *DeepConvBiLSTM* and 0.82 for the *DeepConvLSTM*.

Table 7-11: F1-Scores for the proposed models - LoD 1

Model	Idle - F1	Working - F1	Avg. F1-Score
MLP	0.80	0.86	0.83
DeepConvLSTM	0.82	0.87	0.84
DeepConvBiLSTM	0.80	0.86	0.83

If the confusion matrix of the predictions is considered (Fig. 7-10), it is possible to identify differences in performance among the models. For the MLP (Fig. 7-10a), an accuracy of 73 % is achieved for the **Idle** class and 70 % for the **Working** class. An increase in accuracy is visible for the two hybrid models (Fig. 7-10b and 7-10c). In particular, the accuracy of the class **Working** increases to 90 %. For the class **Idle** only a small increase is visible.



(a) MLP *(b) DeepConvLSTM* *(c) DeepConvBiLSTM*

Figure 7-10: Confusion matrices - LoD 1

It can be argued that the class **Working** is identified very well by the hybrid models. The percentage of samples which actually belong to the class **Working** but are classified as **Idle** is only 9 %. The main problem lies in the identification of the class **Idle**. One reason for the difficulty is the low number of samples in the dataset. As shown in Table 7-12, the samples of class **Idle** amount to less than one fourth of the dataset.

Table 7-12: Number of samples in the training set - LoD 1

Model	Number of samples	% of samples
Idle	51042	23.3 %
Working	167707	76.7 %

7.3.2 LoD 2 - Process steps

Within the second level of detail, the activities within the groups **Working** and **Idle** are considered. The model for the **Working** group deals with the **Casing machine**, **Concreting**, **Drilling** and **Reinforcing** classes. The model for the **Idle** group is responsible for distinguishing between the **Secondary processes** and the **Downtime** activities. For the sake of conciseness, the individual training and validation losses are only provided in the Appendix.

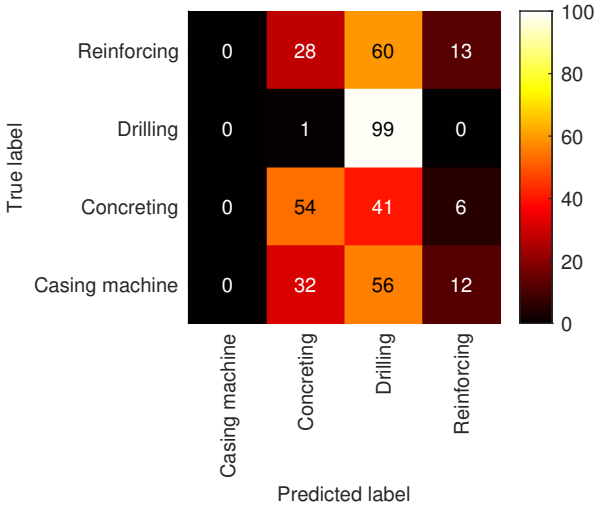
Working

The training procedure of the three considered models for the subsets of the class **working** are shown in Figure B-2 in the Appendix. None of the three models exhibit an overfitting problem. The F1 scores of the test set for the particular labels as well as the average F1 score are shown in Table 7-13. Furthermore, Figure 7-11 shows the confusion matrices of the predictions of the individual models.

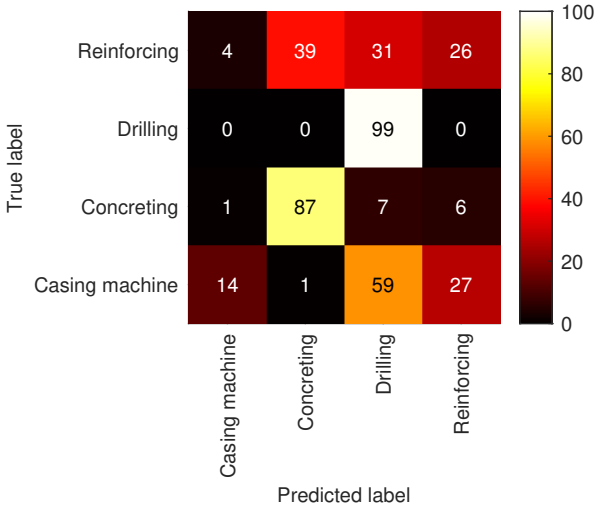
Table 7-13: F1-Scores for the proposed models - LoD 2 - Working

Label	MLP	*DeepConvLSTM*	*DeepConvBiLSTM*
Casing machine	0	0.21	0.21
Concreting	0.57	0.79	0.81
Drilling	0.89	0.94	0.95
Reinforcing	0.20	0.36	0.49
Avg. F1-Score	**0.42**	**0.58**	**0.62**

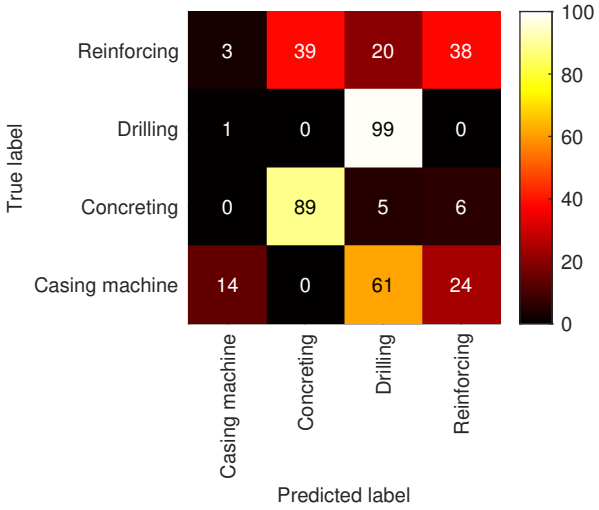
There are noticeable differences between the performance of the different models. The baseline MLP model, for instance, achieves an average F1 score of 0.42. Yet there are considerable differences between the individual classes. For the class **Drilling** a high F1-score of 0.89 is achieved. Moreover, if the confusion matrix (Fig. 7-11a) is considered, it can be seen that 99 % of the samples labeled as **Drilling** were also classified as such. The lower F1 value compared to the very high accuracy stems from the tendency of the model to generally classify samples as **Drilling**. 41 % of samples of class **Concreting** and 60 % of samples of class **Reinforcing** are classified as **Drilling**. For the class **Concreting**, an F1 value of 0.57 and an accuracy of 54 % are achieved. Especially the recognition of the classes **Reinforcing** and **Casing machine** prove to be very difficult for the model.



(a) MLP



(b) DeepConvLSTM



(c) DeepConvBiLSTM

Figure 7-11: Confusion matrices - LoD 2 - Working

Both hybrid models exhibit a better overall performance. The average F1 score increases to 0.58 in the case of the *DeepConvLSTM* and to 0.62 in the case of the bidirectional variant *DeepConvBiLSTM*. Based on the confusion matrices of the models (Fig. 7-11b and 7-11c), it is possible to observe that the entries are more concentrated on the secondary diagonal. For example, the column of samples predicted as **Drilling** is less occupied compared to the MLP model (Fig. 7-11a). The detection of the **Concreting** class is significantly improved when the hybrid models are used, with 87 % and 89 % of the **Concreting** samples being detected as such, respectively. The F1 score for this class increases to approximately 0.8 for both models. Furthermore, the class **Casing machine** is recognized at least to some extent. The F1 score of this class, which amounts to 0 for the MLP, increases to 0.21 for both hybrid models. A further improvement takes place with respect to the class **Reinforcing**.If the MLP correctly classifies only 13 % of the sample of this class, the accuracy for the *DeepConvBiLSTM* is almost three time as high with 38 %. In general, it can be seen that the use of the bidirectional LSTM layers in this considered scenario leads to an improvement in the performance of the models, since there is an increase in the F1 scores for all classes.

Idle

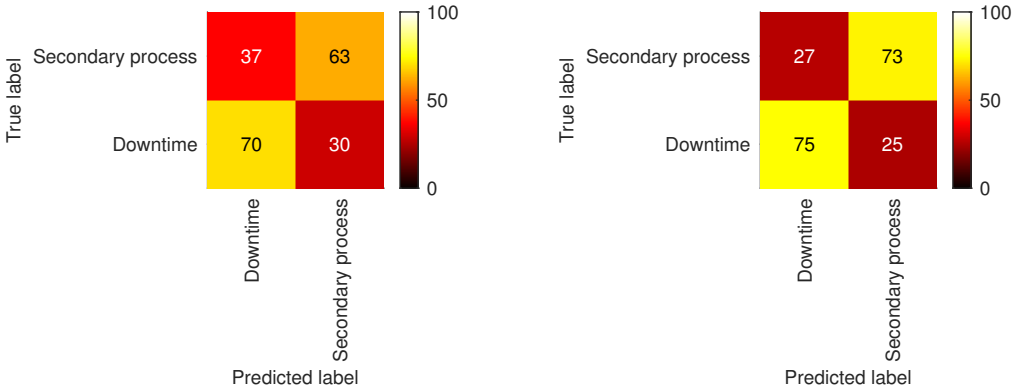
The second group within this level of detail is the **Idle** group and includes both the downtime and the secondary processes necessary for drilling and concreting. The training and validation loss of the models can be found in the Appendix in Figure B-3. The F1 scores for the baseline MLP and the two hybrid models are shown in Table 7-14. In addition, Figure 7-12 shows the confusion matrices for the MLP and for the *DeepConvLSTM* model. The confusion matrix for the bidirectional variant is provided only in the Appendix 7-15c due to its similarity to that of the unidirectional variant.

Table 7-14: F1-Scores for the proposed models - LoD 2 - Idle

Label	MLP	*DeepConvLSTM*	*DeepConvBiLSTM*
Downtime	0.72	0.78	0.78
Secondary processes	0.60	0.69	0.67
Avg. F1-Score	**0.66**	**0.74**	**0.73**

The distinction between activities from the **Downtime** class and the **Secondary processes** class, which proved difficult in Section 7.2.3, is better accomplished when considered at a

different level of detail. The baseline MLP model correctly classifies 70 % of the samples of the class **Downtime** and 63 % of the samples of the class **Secondary processes**. This leads to a mean F1 score of 0.66. For the two hybrid models, the accuracies increase to 75 % and 73 %, respectively. The F1 score rises to approx. 0.73.



(a) MLP *(b) DeepConvLSTM*

Figure 7-12: Confusion matrices - LoD 2 - Idle

7.3.3 LoD 3 - Detailed process steps

In the third and most detailed level, each process step considered in LoD 2 was again examined in more depth and detail. The results for the individual process steps are presented below. Since the test contains only one activity of the class **Downtime** and one activity of the class **Casing machine**, these are not considered in the investigations.

Concreting

The training process of the models is shown in Figure B-7 in the Appendix. Table 7-15 shows the F1 values for the respective models. Based on the F1 scores and the confusion matrix shown in Figure B-9 as an example, it is possible to conclude that classifying within the superclass **Concreting** proves to be very difficult. For all three models, there is a tendency to output the two most common activities (**Concreting** and **Place pouring pipe**) as predictions.

Table 7-15: F1-Scores for the proposed models - LoD 3 - Concreting

Label	MLP	*DeepConvLSTM*	*DeepConvBiLSTM*
Pull standpipe	0.06	0.19	0.13
Remove casing	0.00	0.00	0.08
Remove pouring pipe	0.00	0.00	0.00
Place pouring pipe	0.49	0.56	0.51
Concreting	0.43	0.47	0.48
Avg. F1-Score	**0.20**	**0.24**	**0.24**

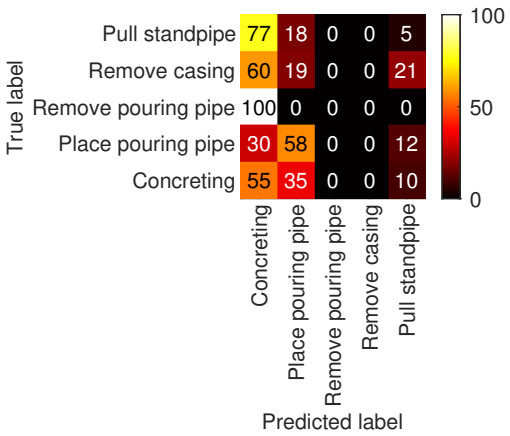


Figure 7-13: Confusion matrix - LoD 3 - Concreting - MLP

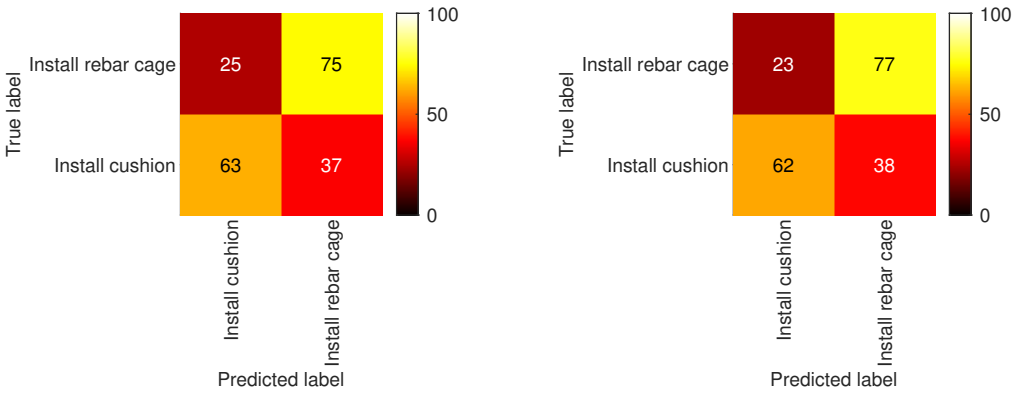
Reinforcing

The process step **Reinforcing** includes the activities **Install rebar cage** and **Install cushion**. The training and validation loss of the models can be found in the Appendix in Figure B-4. The F1 scores for the baseline MLP and the two hybrid models are shown in Table 7-16.

Table 7-16: F1-Scores for the proposed models - LoD 3 - Reinforcing

Label	MLP	*DeepConvLSTM*	*DeepConvBiLSTM*
Install rebar cage	0.79	0.80	0.80
Install cushion	0.56	0.56	0.56
Avg. F1-Score	**0.68**	**0.68**	**0.65**

The performance of the three models considered are very similar and do not exhibit any meaningful differences. Due to the strong similarities, only the confusion matrices for the MLP and *DeepConvLSTM* are presented. The average F1 value is 0.68 for the MLP and *DeepConvLSTM* and 0.65 for the *DeepConvBiLSTM*. It is important to mention the similarity of the considered activities. Both the **Installation of the rebar cage** and the **Installation of the cushion** can be performed either by the machine or also installed by an additional crane. These multiple possible executions mean that the models must model different behaviors for the same label. For example, if the rebar cage is installed using a crane, no activity can be detected from the machine data. The approximately 25 % of the **Install rebar cage** samples and 37 % of the **Install cushion** samples that are misclassified may stem from this strong similarity in machine behavior.



(a) MLP *(b) DeepConvLSTM*

Figure 7-14: Confusion matrices - LoD 3 - Reinforcing

Drilling

The consideration of the process step **Drilling** is the investigation most similar to the previous studies [Ors-2020, Bi-2020, Lia-2020]. In addition to the 5 activities examined in the other studies, the activity **Place standpipe** is also considered here. The training and validation loss of the models can be found in the Appendix in Figure B-6. Table 7-17 shows the mean F1 score and the F1 score of the respective classes for the individual models. Furthermore, Figure 7-15 shows the confusion matrices of the predictions of the three models.

Table 7-17: F1-Scores for the proposed models - LoD 3 - Drilling

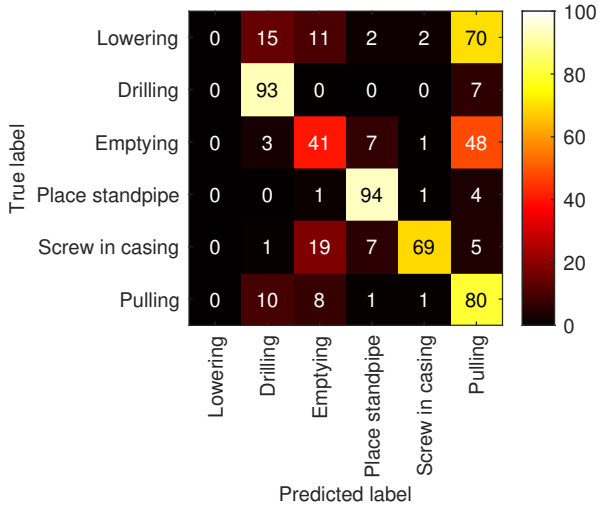
Label	MLP	*DeepConvLSTM*	*DeepConvBiLSTM*
Lowering	0.00	0.73	0.72
Drilling	0.81	0.84	0.83
Emptying	0.43	0.85	0.86
Place standpipe	0.89	0.93	0.93
Screw in casing	0.81	0.90	0.89
Pulling	0.60	0.86	0.87
Avg. F1-Score	**0.59**	**0.85**	**0.85**

The first model considered is the baseline MLP, which achieves an average F1 score of 0.59. Among the six considered activities, three are recognized well, two are recognized averagely, and one is recognized very poorly. The best classified activity is the **Place standpipe**, with an F1 score of 0.89. For the samples of this class, 94 % were correctly classified and the number of samples of other classes classified as **Place standpipe** is low. Similarly, 93 % of the samples of the class **Drilling** were correctly detected. However, in contrast to the class **Place standpipe**, samples of other classes were more often classified as **Drilling**. This resulted in a lower F1 score of 0.81. Most of the incorrect samples classified as **Drilling** originated from the **Lowering** and **Pulling** activities. Samples from the **Drilling** class were misclassified as **Pulling** only 7 % of the time. Also with an F1 score of 0.81 is the class **Screw in casing**. The high F1 score originates primarily from the very low number of false positives, as otherwise an accuracy of only 69 % was achieved for the samples of this class. In contrast, the **pulling** class, which classified 80 % of its samples correctly, only achieves an F1 score of 0.6. This is due to the high number of false positives. 70 % of the **Lowering** and almost 50 % of the **Emptying** samples were classified as **Pulling**. The **Emptying** class

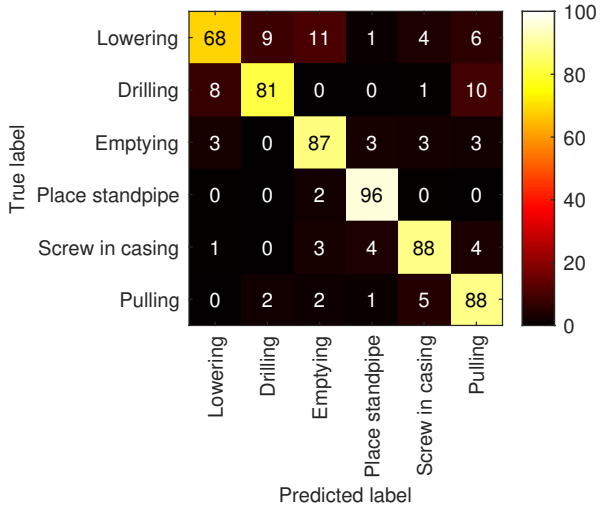
has the second worst F1 score at 0.43. As mentioned earlier, about 50 % of the samples in this class were assigned to the **Pulling** class. Last, the activity **Lowering** is not considered at all in the modeling and no sample was classified correctly. This, obviously, results in an F1 score of 0.

The adoption of the hybrid models results in a significant increase of the average F1-score. It amounts to 0.85 for the *DeepConvLSTM* as well as for the *DeepConvBiLSTM* model. All individual labels show an increase of the F1 score. The F1 score of the **Place standpipe** class, which yielded the highest F1 score for the MLP, increases to 0.93 due to a decrease in the number of false positives and an increase in accuracy to 96 %. In Figures 7-15b and 7-15c, a drop in the accuracy of the **Drilling** class to 81 % is noticeable. However, this did not result in a decrease of the F1 score, since simultaneously the number of false positives also decreased. The misclassified samples were mainly assigned to the classes **Lowering** and **Pulling**. This is attention-grabbing since these two classes characterize the activities which occur directly before or after the drilling. It is possible that, as also mentioned in Section 7.2.3, the difficulty in classification lies at the transitions of two subsequent activities.

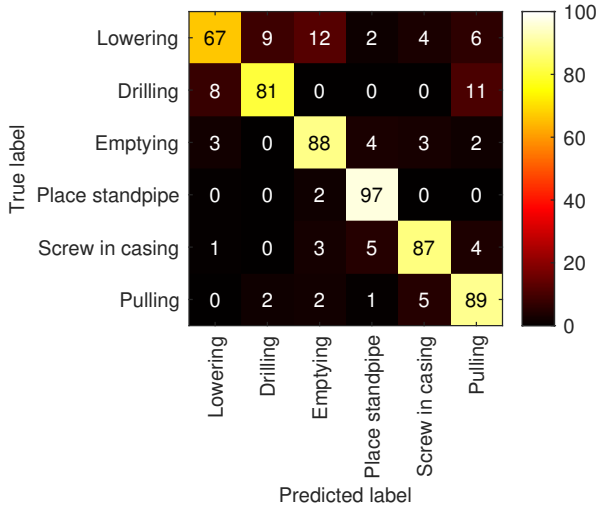
Another aspect which is better addressed by the hybrid models is the classification of the activity **Screw in casing**, which had the second lowest accuracy with 69 % for the baseline MLP. The accuracy increases to about 87 % for the two hybrid models, while the number of false positives barely changes. This leads to an increase in F1 scores to 0.90 for the *DeepConvLSTM* and 0.89 for the *DeepConvBiLSTM*. However, the strongest improvements take place for the **Emptying** and **Lowering** classes. For instance, the activity **Emptying**, which is correctly recognized by the MLP only 41 % of the time, is correctly recognized by the *DeepConvLSTM* model 87 % of the time. In the case of the **Lowering** class, about 67 % of the samples are correctly recognized, which represents a drastic increase compared to the 0 % of the MLP. Nonetheless, this class is the worst recognized activity, which is also consistent with the results from previous work [Ors-2020, Bi-2020, Lia-2020].



(a) MLP



(b) DeepConvLSTM



(c) DeepConvBiLSTM

Figure 7-15: Confusion matrices - LoD 3 - Drilling

Secondary process

The plots showing the training and validation loss can be found in the appendix in Figure B-5. Table 7-18 shows the F1 scores for the respective models.

Table 7-18: F1-Scores for the proposed models - LoD 3 - Secondary processes

Label	MLP	*DeepConvLSTM*	*DeepConvBiLSTM*
Refill water	0.82	0.82	0.82
Relocate	-	-	-
Depth sensing	0.40	0.40	0.46
Refueling	0.00	0.00	0.00
Other	0.35	0.35	0.37
Avg. F1-Score	**0.39**	**0.39**	**0.41**

Both the baseline MLP and the two hybrid models have an F1 score of about 0.4, indicating no advantages of using the hybrid model over the MLP. As in the analyses from Section 7.2.3, only the activity **Refill water** can be properly detected among the secondary processes. The F1 score for this activity is 0.82 for all models.

This is another case in which the choice of used labels plays a major role. As can be seen in Figure 7-16, the activities **Relocate**, **Depth sensing**, **Refueling** and **Other** are not well distinguished, since these only describe the actual process and do not imply any movement of the machine.

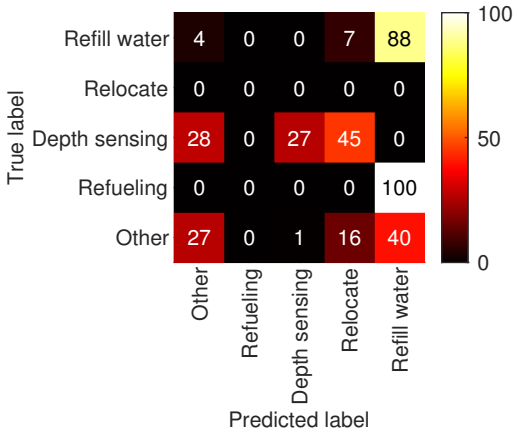


Figure 7-16: Confusion matrix - LoD 3 - Secondary processes - MLP

8 Discussion and future work

In this chapter, the knowledge acquired in the previous chapter is discussed and explained with regard to the overall context. To facilitate the overview, individual aspects are addressed separately in subsections. Furthermore, possible approaches and suggestions for further work are discussed.

8.1 Discussion

8.1.1 Hybrid models

Overall, it is apparent that the use of hybrid models provides a significant advantage over the baseline models. Although the number of LSTM layers, which are theoretically responsible for modeling the temporal dependencies, are exactly the same for the LSTM model and the hybrid models, the quality of the predictions of the hybrid models is significantly higher. Adding the CNN layers as a mechanism for feature extraction proves to be a powerful tool. Merely feeding unprocessed sensor measurements to the LSTM model is not sufficient. On the other hand, as already proposed by Ordóñez and Roggen [Ord-2016], the CNN layers show to be very effective in extracting low and high level features from raw sensor data. Furthermore, the use of bidirectional LSTM layers in the DeepConvBiLSTM model also appear to have a positive impact on the results. An increase of the F1 score could be identified for all classes. The mean F1 score increased from 0.87 to 0.89, an improvement of about 15%. These improvements are consistent with expectations of a weak gain in performance (see Section 6.2.2).

8.1.2 Generalization capabilities

When considering the generalization capabilities of the models to data obtained from other working days, several difficulties emerged. The very good results obtained in Section 7.1 on training and test data obtained from the same day suffered a significant deterioration. Based on these results, it can be concluded that, especially for the considered downtime and secondary processes which were initially very well modeled by the hybrid models,

there is a strong dependency on the respective days and these may vary between different days. Thus, a mere transfer of the excellent results from Section 7.1 to other days is not possible.

8.1.3 Labeling strategy as a limitation

The selection of labels could be identified as a significant problem for the activity recognition. For both the flat classification in Section 7.2.3 as well as for the hierarchical classification from Section 7.3, certain activities proved to be very difficult to classify. In addition to this, most confusion misclassifications took place between these activities.

The root cause of these problems is the fact that the machine behavior was not taken into account when selecting the labeling strategy, which in this case was only based on the process steps. This resulted in different labels being assigned in cases where the machine does not exhibit different behaviors. As a simple example, the activities **Concreting**, **Waiting for concrete** and **Other** can be considered. In all three activities, the machine just passively waits by while the process is performed by other equipment and workers. A differentiation of the activities on the basis of the machine data is thus only barely possible. However, if activities were considered in which the machine is actively involved, such as **Drilling**, **Screw in casing** or **Emptying**, then satisfactory results were achieved in all cases.

It can be pointed out that already during the selection of the labels for the automatic activity recognition based on machine data, the basic machine behavior has to be considered.

8.2 Future work

Based on the obtained findings, several recommendations for further research can be formulated. The first group of possibilities relates directly to the models used in this thesis. Due to the high number of different models and parameter studies, no hyperparameter tuning was performed in this study. However, this may further improve the achieved results without demanding major changes. The depth of the models, number of LSTM cells, number of convolution filters and filter size can also be considered as hyperparameters.

Since the considered hybrid models have shown to be generally able to model well the behavior of the different activities and the problems arise mainly in the generalization of the models to data collected on other days, the use of a larger data set can possibly benefit the generalization. Another aspect which still has to be investigated is the generalization of the models to other construction sites. For example, it can be investigated whether the boundary conditions of the construction site, such as the geology of the soil, influence the machine data and ultimately the activity recognition. It may also be possible to examine the influence of the driving style of different machine operators on the activity detection.

The second group of suggestions deals with the approaches to activity recognition per se. As already described on several occasions in Chapter 7 and explained again in Section 8.1, the activities were labeled with regard to the performed process. Instead of focusing on the process, a more appropriate approach might be to consider the activities with respect to the actual movements of the machines. As could be shown in the course of this thesis, activities in which the machine actively participates are well recognized. A possible division of the labels used in this work into **Active** and **Idle** can be found in Table 8-1. All the activities labeled as **Idle** could be merged into a single class, which would presumably simplify modeling.

This approach can further be used in combination with other, non-Deep Learning based, approaches. As an example, modeling the process as a statechart would be a suitable approach. In this case, the detected activities based on the movement of the machine may be used to trigger the transitions between the individual steps. Advantage of such a modeling would be the limitation of the possible state changes to the only reasonable transition, something that is not achieved in the considered models.

Table 8-1: Proposed classification of activities into Active/Idle

Label	Active	Idle	Label	Active	Idle
Break		X	Place standpipe	X	
Failure		X	Pull standpipe	X	
Waiting for concrete		X	Remove casing	X	
Waiting for other		X	Screw in casing	X	
Lowering	X		CM - Pull standpipe		X
Concreting		X	CM - Screw in casing		X
Install cushion		X	Pulling	X	
Drilling	X		Other		X
Drilling with bucket	X		Refueling		X
Install rebar cage		X	Depth sensing		X
Emptying	X		Tool exchange		X
Remove pouring pipe		X	Relocate		X
Place pouring pipe		X	Refill water		X

In addition to the mentioned points, there is the possibility to extract further information based on the existing telematics data. One possibility, for example, is to investigate the extent to which information about the geology of the soil can be extracted from the machine data.

9 Conclusion

The goal of this thesis was to investigate the possibilities of performing automatic activity recognition based on already available machine data of a Kelly drilling machine. For this purpose, different Deep Learning architectures were considered and investigated.

In the first part of the thesis, the state of the art regarding automatic activity recognition of construction machines was presented and possible problems of these approaches regarding special civil engineering were highlighted. Furthermore, the first part presented the theoretical background of the Kelly drilling process and Deep Learning. In the second part, the methodology employed in this thesis was explained. The acquisition and exploration of the data set, as well as the segmentation, scaling and splitting into training and test set as parts of the preprocessing were explained.

In Chapter 6 the used models were presented and explained. Emphasis was placed on two different hybrid models that are intended to combine the capabilities of both convolutional and recurrent neural networks. Both a unidirectional and a bidirectional variant were examined. To allow comparison, a multilayer perceptron and a pure LSTM model were used as baseline models.

The acquired knowledge can be summarized with respect to the research objectives formulated in Section 2.2 as follows:

1. Although further improvement is still possible, automatic activity recognition based on telematics data is shown to fundamentally be a solid alternative to existing approaches.
2. Existing model architectures for activity recognition based on, mainly, acceleration data, show good portability to activity recognition based on machine data.
3. The use of hybrid models based on the feature extraction capabilities of CNNs and the temporal modeling capabilities of LSTM is shown to be beneficial. These also proved to be robust against outliers and were able to achieve satisfactory results even without strong preprocessing of the data. A limitation so far has been the non-optimal generalization to data from other working days.

4. The selection of the labeling strategy proves to be a key issue for a satisfactory classification of the activities. For this purpose, it is important to ensure that labels are not only selected from a pure process view, but also account for the machine behavior during the process.

Part IV

Supplement

References

[Ahn-2015] Ahn, Changbum R.; Lee, SangHyun; Peña-Mora, Feniosky: Application of Low-Cost Accelerometers for Measuring the Operational Efficiency of a Construction Equipment Fleet. In: Journal of Computing in Civil Engineering 29.2 (2015), p. 04014042.

[Akh-2015] Akhavian, Reza; Behzadan, Amir H.: Construction equipment activity recognition for simulation input modeling using mobile sensors and machine learning classifiers. In: Advanced Engineering Informatics 29.4 (2015). PII: S1474034615000282, pp. 867–877.

[Ban-2014] Banos, Oresti et al.: Window size impact in human activity recognition. eng. In: Sensors (Basel, Switzerland) 14.4 (2014). Journal Article Research Support, Non-U.S. Gov't, pp. 6474–6499.

[Bau-2021] Bauer Group: Kelly drilling. 2021. Url: https://bit.ly/2RXi8Qd (visited on 03/10/2021).

[Ber-2019] Bertschek, Irene; Niebel, Thomas; Ohnemus, Jörg: Zukunft Bau – Beitrag der Digitalisierung zur Produktivität in der Baubranche. 2019.

[Bi-2020] Bi, H.: Application of artificial neural networks for the analysis of sensor data from machines. Garching b. München: Technical University of Munich, 2020.

[Cha-2000] Chapman, Pete et al.: CRISP-DM 1.0. Step-by-step data mining guide. SPSS, 2000.

[Cho-2018] Chollet, François: Deep learning with Python. eng. Safari Tech Books Online. Chollet, François (VerfasserIn). Manning, Shelter Island, NY: 2018. 361 pp. Url: http://proquest.safaribooksonline.com/9781617294433.

[Fis-2020] Fischer, A. et al.: Begleitende Prozesssimulation für das Kellybohrverfahren. In: 8. Fachtagung Baumaschinentechnik 2020. 2020.

[Fis-2021] Fischer, A. et al.: Detecting Equipment Activities by Using Machine Learning Algorithms. In: 17th IFAC Symposium on Information Control Problems in Manufacturing (INCOM 2021). Budapest, Hungary: 2021.

[Fu-2011] Fu, Tak-chung: A review on time series data mining. In: Engineering Applications of Artificial Intelligence 24.1 (2011). PII: S0952197610001727, pp. 164–181.

[Gér-2019] Géron, Aurélien: Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow. Concepts, tools, and techniques to build intelligent systems. Second edition. O'Reilly Media: 2019. 819eiten.

[Gol-2013] Golparvar-Fard, Mani; Heydarian, Arsalan; Niebles, Juan Carlos: Vision-based action recognition of earthmoving equipment using spatio-temporal features and support vector machine classifiers. In: Advanced Engineering Informatics 27.4 (2013). PII: S1474034613000761, pp. 652–663.

[Gon-2011] Gong, Jie; Caldas, Carlos H.; Gordon, Chris: Learning and classifying actions of construction workers and equipment using Bag-of-Video-Feature-Words and Bayesian network models. In: Advanced Engineering Informatics 25.4 (2011). PII: S1474034611000346, pp. 771–782.

[Goo-2018] Goodfellow, Ian.; Courville, Aaron; Bengio, Yoshua: Deep Learning. Das umfassende handbuch : grundlagen, aktuelle verfahren und algorithmen, neue forschungsansätze. ger. 1. Auflage. Goodfellow, Ian. (VerfasserIn) Courville, Aaron (joint author) Lenz, Guido (ÜbersetzerIn) Bengio, Yoshua (joint author). Verlags GmbH & Co. KG, Frechen: 2018.

[Gra-2005] Graves, Alex; Schmidhuber, Jürgen: Framewise phoneme classification with bidirectional LSTM and other neural network architectures. eng. In: Neural networks : the official journal of the International Neural Network Society 18.5-6 (2005). Journal Article Research Support, Non-U.S. Gov't, pp. 602–610.

[Hoc-1997] Hochreiter, S.; Schmidhuber, J.: Long short-term memory. eng. In: Neural computation 9.8 (1997). Journal Article Research Support, Non-U.S. Gov't, pp. 1735–1780.

[Ism-2019] Ismail Fawaz, Hassan et al.: Deep learning for time series classification: a review. In: Data Mining and Knowledge Discovery 33.4 (2019). PII: 619, pp. 917–963.

[Kim-2019] Kim, Jinwoo; Chi, Seokho: Action recognition of earthmoving excavators based on sequential pattern analysis of visual features and operation cycles. In: Automation in Construction 104 (2019). PII: S0926580518312731, pp. 255–264.

[Lan-2021] Langroodi, Armin Kassemi; Vahdatikhaki, Faridaddin; Doree, Andre: Activity recognition of construction equipment using fractional random forest. In: Automation in Construction 122 (2021). PII: S0926580520310451, p. 103465.

[LeC-2015] LeCun, Yann; Bengio, Yoshua; Hinton, Geoffrey: Deep learning. eng. In: Nature 521.7553 (2015). Journal Article Research Support, Non-U.S. Gov't Research Support, U.S. Gov't, Non-P.H.S. Review, pp. 436–444.

[Lew-2011] Lewis, Phil et al.: Assessing Effects of Operational Efficiency on Pollutant Emissions of Nonroad Diesel Construction Equipment. In: Transportation Research Record: Journal of the Transportation Research Board 2233.1 (2011), pp. 11–18.

[Lia-2020] Liang, M.: Analysis of Machine Data in Special Foundation Engineering using Machine Learning. Garching b. München: Technical University of Munich, 2020.

[May-2011] Maybaum, Georg: Verfahrenstechnik und Baubetrieb im Grund- und Spezialtiefbau. Baugrund - Baugruben - Baugrundverbesserung - Pfahlgründungen - Grundwasserhaltung. ger. 2., überarbeitete und aktualisierte Auflage. Praxis. Vieweg+Teubner Verlag / Springer Fachmedien Wiesbaden GmbH Wiesbaden, Wiesbaden: 2011.

[Min-2020] Minh Dang, L. et al.: Sensor-based and vision-based human activity recognition: A comprehensive survey. In: Pattern Recognition 108 (2020). PII: S0031320320303642, p. 107561.

[Mur-2017] Murad, Abdulmajid; Pyun, Jae-Young: Deep Recurrent Neural Networks for Human Activity Recognition. eng. In: Sensors (Basel, Switzerland) 17.11 (2017). Journal Article.

[Ola-2015] Olah, Christopher: Bidirectional Recursive Neural Networks. 2015. Url: https://bit.ly/35kNyCX (visited on 06/10/2021).

[Ord-2016] Ordóñez, Francisco Javier; Roggen, Daniel: Deep Convolutional and LSTM Recurrent Neural Networks for Multimodal Wearable Activity Recognition. eng. In: Sensors (Basel, Switzerland) 16.1 (2016). Journal Article Research Support, Non-U.S. Gov't.

[Ors-2020] Orschlet, V.: Analysis for Activity Recognition in Construction Machinery using Machine Learning. Master Thesis. Garching b. München: Technical University of Munich, 2020.

[PwC-2013] PwC: Baubranche im Fokus. Wie das deutsche Baugewerbe in die Zukunft blickt. 2013.

[Ras-2020] Rashid, Khandakar M.; Louis, Joseph: Automated Activity Identification for Construction Equipment Using Motion Data From Articulated Members. In: Frontiers in Built Environment 5 (2020).

[Ras-2019] Rashid, Khandakar M.; Louis, Joseph: Times-series data augmentation and deep learning for construction equipment activity recognition. In: Advanced Engineering Informatics 42 (2019). PII: S1474034619300886, p. 100944.

[Sch-1997] Schuster, M.; Paliwal, K. K.: Bidirectional recurrent neural networks. In: IEEE Transactions on Signal Processing 45.11 (1997), pp. 2673–2681.

[Sci-2021] Scikit-Learn: sklearn.model_selection.train_test_split. 2021. (Visited on 06/05/2021).

[She-2020] Sherafat, Behnam et al.: Automated Methods for Activity Recognition of Construction Workers and Equipment: State-of-the-Art Review. In: Journal of Construction Engineering and Management 146.6 (2020), p. 03120002.

[Shr-2019] Shrestha, Ajay; Mahmood, Ausif: Review of Deep Learning Algorithms and Architectures. In: IEEE Access 7 (2019), pp. 53040–53065.

[Sil-2011] Silla, Carlos N.; Freitas, Alex A.: A survey of hierarchical classification across different application domains. In: Data Mining and Knowledge Discovery 22.1-2 (2011). PII: 175, pp. 31–72.

[Sin-2021] Singh, Satya P. et al.: Deep ConvLSTM with self-attention for human activity decoding using wearables. In: IEEE Sensors Journal 21.6 (2021). 8 pages, 2 figures, 3 tables. IEEE Sensors Journal, 2020, pp. 8575–8582. Url: http://arxiv.org/pdf/2005.00698v2.

[Sla-2020] Slaton, Trevor; Hernandez, Carlos; Akhavian, Reza: Construction activity recognition with convolutional recurrent networks. In: Automation in Construction 113 (2020). PII: S0926580519310234, p. 103138.

[Sta-2020] Statista; Statistisches Bundesamt: Prognostizierte Umsatzentwicklung in der Branche Baugewerbe in Deutschland in den Jahren von 2012 bis 2024. Ed. by Statista. 2020. (Visited on 06/15/2021).

[Sta-2021] Statistisches Bundesamt; Hauptverband der Deutschen Bauindustrie: Anzahl der Beschäftigten im Bauhauptgewerbe in Deutschland in den Jahren 2003 bis 2020. Ed. by Statista. 2021. (Visited on 06/15/2021).

[Stu-2021] Studer, Stefan et al.: Towards CRISP-ML(Q): A Machine Learning Process Model with Quality Assurance Methodology. In: Machine Learning and Knowledge Extraction 3.2 (2021). PII: make3020020, pp. 392–413.

[Tec-2021] Technical University of Munich: Building 4.0 - Digitalization of the construction site. 2021. Url: https://www.mw.tum.de/en/fml/research/current-research-projects/building-40-digitalization-of-the-construction-site/ (visited on 01/20/2021).

[U.S-2008] U.S. Environmental Protection Agency, ed.: Quantifying Greenhouse Gas Emissions. from Key Industrial Sectors in the United States. 2008.

[Xu-2019] Xu, Cheng et al.: InnoHAR: A Deep Neural Network for Complex Human Activity Recognition. In: IEEE Access 7 (2019), pp. 9893–9902.

[Yu-2019] Yu, Yong et al.: A Review of Recurrent Neural Networks: LSTM Cells and Network Architectures. eng. In: Neural computation 31.7 (2019). Journal Article Research Support, Non-U.S. Gov't Review, pp. 1235–1270. eprint: 31113301.

[Zou-2007] Zou, Junhao; Kim, Hyoungkwan: Using Hue, Saturation, and Value Color Space for Hydraulic Excavator Idle Time Analysis. In: Journal of Computing in Civil Engineering 21.4 (2007), pp. 238–246.

List of Figures

List of Tables

A Source code

A.1 Merging of machine and activity data

```
1  import glob
2  import pandas as pd
3  import matplotlib.pyplot as plt
4  import numpy as np
5  import datetime
6
7  def import_file(path):
8      data = pd.read_excel(path)
9      return data
10
11 def search_start_end_time(path_list, file):
12     time_list = import_file(path_list)
13     selected_row = time_list[time_list["b_report"] == file]
14     start_time = selected_row["start"].iloc[0]
15     end_time = selected_row["end"].iloc[0]
16     return start_time, end_time
17
18 path_b_report = r'???'
19 path_activity = r'???'
20 path_list = r'???'
21 path_output = r'???'
22
23 all_files_b_report = glob.glob(path_b_report + "/*.xlsx")
24 all_files_activity = glob.glob(path_activity + "/*.xlsx")
25
26 m=0
27
28 for file_b_report in all_files_b_report:
29     m=m+1
30     file_name = file_b_report[len(path_b_report)+1:]
31     date = file_name[:8]
32     date_complete = file_name[:-5]
33     file_activity = path_activity+"\\"+date+'.xlsx'
34
35     data_b_report = import_file(file_b_report)
36
37     try:
```

```
38          data_activity = import_file(file_activity)
39          print("Successful import: ", date)
40      except:
41          print("Activity data does not exist for: ", date)
42
43      # Search start and end time
44      start_time, end_time = search_start_end_time(path_list,
            ↪ date_complete)
45
46      # Add timestamp to data_b_report
47      data_b_report["time"]=pd.Series([start_time] * len(data_b_report))
48      n=len(data_b_report)
49
50      for i in range(1,n):
51          data_b_report["time"].iloc[i] =  data_b_report["time"].iloc[i-1]
                ↪  + pd.Timedelta(seconds=1);
52
53      # Delete B-Report data before and after start of Activity-data
54      for index, row in data_b_report.iterrows():
55          if row['time'] < data_activity["Datum"].iloc[0]:
56              data_b_report.drop(index, inplace=True)
57          elif row['time'] > data_activity["Datum"].iloc[-1]:
58              data_b_report.drop(index, inplace=True)
59
60      # Create labels from activity data
61      labels = []
62      n = len(data_b_report)
63      k = 0
64
65      for i in range(n-1):
66          if data_b_report["time"].iloc[i] >= data_activity["Datum"].iloc[
                ↪ k+1]:
67              k=k+1
68          labels.append(data_activity["Name"].iloc[k])
69
70      labels.append(labels[-1])
71
72      # Merge labels to B-Report data
73      data_b_report["labels"]=labels
74
75      # Reset index
```

```
76      data_b_report.reset_index()
77
78      # Export data
79      data_b_report.to_csv(path_output+"\\dataset"+str(m)+".csv")
80      print(date, "␣successfully␣merged␣and␣exported")
```

A.2 Data preprocessing pipeline

A.2.1 Random split

```
1  def data_pipeline(data_list,n_timesteps, labelling_strategy, overlap):
2    data_merged = pd.concat(data_list)
3    data = data_merged.drop(["time", "Absolut␣depth␣[m]","Casing␣Length␣[m
       ↪ ]","Boring␣threshold␣[m]","Status␣Rig␣[␣]"], axis = 1)
4    data.replace("Nebenprozesszeit␣-␣Leeren", "Hauptzeit␣-␣Leeren",
       ↪ inplace = True)
5
6    labels = data["labels"].values
7    unique_labels = np.unique(labels)
8    unique_labels = np.delete(unique_labels,np.argwhere(unique_labels=='
       ↪ Ausfallzeit␣-␣Wartung'),0)
9    print(unique_labels)
10   inputs = data.drop(["labels"], axis=1).values
11   scaler = MinMaxScaler();
12   inputs = scaler.fit_transform(inputs)
13   X = []
14   y = []
15   i=0
16   step = (1-overlap)*n_timesteps
17   if step == 0:
18      step =1
19   while i<inputs.shape[0]:
20      end_index = i + n_timesteps
21
22      if end_index >= inputs.shape[0]:
23         break;
24
25      X_temp = inputs[i:end_index,:]
26
27      if labelling_strategy == "last":
28         y_temp = labels[end_index-1]
29
```

```
30          if labelling_strategy == "frequency":
31              y_temp = stats.mode(labels[i:end_index])[0]
32
33          X.append(X_temp)
34          y.append(y_temp)
35
36          i = int(i + step)
37
38      X = np.array(X)
39      y = np.array(y)
40
41      indices_maintenance = np.argwhere(y=='Ausfallzeit - Wartung')
42
43      X = np.delete(X,indices_maintenance,0)
44      y = np.delete(y,indices_maintenance,0)
45
46      y = label_binarize(y,unique_labels)
47
48      X_train_full, X_test, y_train_full, y_test = train_test_split(X, y,
            ↪ test_size=0.3, random_state = 0)
49      X_train, X_valid, y_train, y_valid = train_test_split(X_train_full,
            ↪ y_train_full, test_size=0.2, random_state = 0)
50
51      if n_timesteps == 1:
52          X_train = X_train.squeeze()
53          X_test = X_test.squeeze()
54          X_valid = X_valid.squeeze()
55
56      return X_train, y_train, X_valid, y_valid, X_test, y_test,
            ↪ unique_labels
```

A.2.2 Split by days

```
1   def data_pipeline_cont(data_list,file_names,n_timesteps,
        ↪ labelling_strategy, overlap):
2       index_train_names = [4,7,9,10,11,12,14,15,17,19]
3       index_val_names = [1,6,20]
4       index_test_names = [2,3,5,8,13,16,18]
5
6       labels_working = ["Hauptzeit - Bohren", "Hauptzeit - Bohren -
            ↪ Bohreimer", "Hauptzeit - Verrohrung eindrehen", "
            ↪ Nebenprozesszeit - Leeren", "Hauptzeit - Ablassen", "Hauptzeit -
```

```
    ↪ ␣Ziehen", "Hauptzeit␣-␣Standrohr␣setzen", "Hauptzeit␣-␣Standrohr
    ↪ ␣ziehen", "Hauptzeit␣-␣Verrohrung␣abnehmen",]
7   labels_idle = ["Ausfallzeit␣-␣Schaden","Ausfallzeit␣-␣Pause","
    ↪ Ausfallzeit␣-␣Warten␣auf␣-␣Sonstiges","Ausfallzeit␣-␣Warten␣auf␣
    ↪ -␣Beton","Nebenprozesszeit␣-␣Sonstiges","Nebenprozesszeit␣-␣
    ↪ Tiefenmessung","Hauptzeit␣-␣Bewehrung␣einbauen","Hauptzeit␣-␣
    ↪ Sch ttrohr␣-␣Entfernen", "Hauptzeit␣-␣Sch ttrohr␣-␣Setzen","
    ↪ Hauptzeit␣-␣Einbau␣Kiespolster","Hauptzeit␣-␣Betonieren", "
    ↪ Hauptzeit␣-␣Verrohrungsanlage␣-␣Standrohr␣ziehen", "Hauptzeit␣-␣
    ↪ Verrohrungsanlage␣-␣Verrohrung␣eindrehen","Nebenprozesszeit␣-␣
    ↪ Umsetzen","Nebenprozesszeit␣-␣Wasser␣nachf llen","
    ↪ Nebenprozesszeit␣-␣Werkzeugwechsel","Nebenprozesszeit␣-␣Tanken"]
8
9   labels_concreting = ["Hauptzeit␣-␣Betonieren", "Hauptzeit␣-␣
    ↪ Sch ttrohr␣-␣Entfernen", "Hauptzeit␣-␣Sch ttrohr␣-␣Setzen", "
    ↪ Hauptzeit␣-␣Verrohrung␣abnehmen","Hauptzeit␣-␣Standrohr␣ziehen"]
10  labels_drilling = ["Hauptzeit␣-␣Bohren","Hauptzeit␣-␣Bohren␣-␣
    ↪ Bohreimer","Hauptzeit␣-␣Verrohrung␣eindrehen", "Hauptzeit␣-␣
    ↪ Leeren", "Hauptzeit␣-␣Ablassen", "Hauptzeit␣-␣Ziehen", "
    ↪ Hauptzeit␣-␣Standrohr␣setzen"]
11  labels_reinforce = ["Hauptzeit␣-␣Bewehrung␣einbauen","Hauptzeit␣-␣
    ↪ Einbau␣Kiespolster"]
12  labels_cm = ["Hauptzeit␣-␣Verrohrungsanlage␣-␣Standrohr␣ziehen", "
    ↪ Hauptzeit␣-␣Verrohrungsanlage␣-␣Verrohrung␣eindrehen"]
13  labels_secondary = ["Nebenprozesszeit␣-␣Sonstiges","Nebenprozesszeit␣-
    ↪ ␣Tiefenmessung","Nebenprozesszeit␣-␣Umsetzen","Nebenprozesszeit␣
    ↪ -␣Wasser␣nachf llen","Nebenprozesszeit␣-␣Werkzeugwechsel","
    ↪ Nebenprozesszeit␣-␣Tanken"]
14  labels_downtime = ["Ausfallzeit␣-␣Schaden","Ausfallzeit␣-␣Pause","
    ↪ Ausfallzeit␣-␣Warten␣auf␣-␣Sonstiges","Ausfallzeit␣-␣Warten␣auf␣
    ↪ -␣Beton"]
15
16
17  index_train = []
18
19  for elem in index_train_names:
20    string = 'dataset'+str(elem)+'.csv'
21    index = file_names.index(string)
22    index_train.append(index)
23
24  index_val = []
```

```
25
26    for elem in index_val_names:
27        string = 'dataset'+str(elem)+'.csv'
28        index = file_names.index(string)
29        index_val.append(index)
30
31
32    index_test = []
33
34    for elem in index_test_names:
35        string = 'dataset'+str(elem)+'.csv'
36        index = file_names.index(string)
37        index_test.append(index)
38
39    data_list_train = []
40    data_list_test = []
41    data_list_val = []
42
43    for i in range(len(index_test)):
44        data_list_test.append(data_list[index_test[i]])
45
46    for i in range(len(index_val)):
47        data_list_val.append(data_list[index_val[i]])
48
49    for i in range(len(index_train)):
50        data_list_train.append(data_list[index_train[i]])
51
52    data_train_merged = pd.concat(data_list_train)
53    data_train_merged.head()
54    data_train = data_train_merged.drop(["time", "Absolut␣depth␣[m]","
          ↪ Casing␣Length␣[m]","Boring␣threshold␣[m]","Status␣Rig␣[␣]"],
          ↪ axis = 1, errors='ignore')
55    data_train.replace("Nebenprozesszeit␣-␣Leeren", "Hauptzeit␣-␣Leeren",
          ↪ inplace = True)
56    #data_train = data_train.replace(labels_concreting, "Concreting")
57    #data_train = data_train.replace(labels_drilling, "Drilling")
58    #data_train = data_train.replace(labels_reinforce, "Reinforcing")
59    #data_train = data_train.replace(labels_cm, "Casing machine")
60    #data_train = data_train.replace(labels_secondary, "Secondary process
          ↪ ")
61    #data_train = data_train.replace(labels_downtime, "Downtime")
```

```
62
63
64    data_test_merged = pd.concat(data_list_test)
65    data_test_merged.head()
66    data_test = data_test_merged.drop(["time", "Absolut␣depth␣[m]","Casing
          ↪ ␣Length␣[m]","Boring␣threshold␣[m]","Status␣Rig␣[␣]"], axis = 1,
          ↪  errors='ignore')
67    data_test.replace("Nebenprozesszeit␣-␣Leeren", "Hauptzeit␣-␣Leeren",
          ↪ inplace = True)
68    #data_test = data_test.replace(labels_concreting, "Concreting")
69    #data_test = data_test.replace(labels_drilling, "Drilling")
70    #data_test = data_test.replace(labels_reinforce, "Reinforcing")
71    #data_test = data_test.replace(labels_cm, "Casing machine")
72    #data_test = data_test.replace(labels_secondary, "Secondary process")
73    #data_test = data_test.replace(labels_downtime, "Downtime")
74
75    data_val_merged = pd.concat(data_list_val)
76    data_val_merged.head()
77    data_val = data_val_merged.drop(["time", "Absolut␣depth␣[m]","Casing␣
          ↪ Length␣[m]","Boring␣threshold␣[m]","Status␣Rig␣[␣]"], axis = 1,
          ↪ errors='ignore')
78    data_val.replace("Nebenprozesszeit␣-␣Leeren", "Hauptzeit␣-␣Leeren",
          ↪ inplace = True)
79    #data_val = data_val.replace(labels_concreting, "Concreting")
80    #data_val = data_val.replace(labels_drilling, "Drilling")
81    #data_val = data_val.replace(labels_reinforce, "Reinforcing")
82    #data_val = data_val.replace(labels_cm, "Casing machine")
83    #data_val = data_val.replace(labels_secondary, "Secondary process")
84    #data_val = data_val.replace(labels_downtime, "Downtime")
85
86
87    data_merged = pd.concat(data_list)
88    data = data_merged.drop(["time", "Absolut␣depth␣[m]","Casing␣Length␣[m
          ↪ ]","Boring␣threshold␣[m]","Status␣Rig␣[␣]"], axis = 1)
89    data.replace("Nebenprozesszeit␣-␣Leeren", "Hauptzeit␣-␣Leeren",
          ↪ inplace = True)
90    #data = data.replace(labels_concreting, "Concreting")
91    #data = data.replace(labels_drilling, "Drilling")
92    #data = data.replace(labels_reinforce, "Reinforcing")
93    #data = data.replace(labels_cm, "Casing machine")
94    #data = data.replace(labels_secondary, "Secondary process")
```

```
95    #data = data.replace(labels_downtime, "Downtime")
96
97
98    labels = data["labels"].values
99    unique_labels = np.unique(labels)
100   unique_labels = np.delete(unique_labels,np.argwhere(unique_labels=='
          ↪ Ausfallzeit - Wartung'),0)
101   inputs = data.drop(["labels"], axis=1).values
102   scaler = MinMaxScaler();
103   scaler.fit(inputs)
104
105   print(data_train.shape)
106   print(data_test.shape)
107   print(data_val.shape)
108
109   #----------------TRAINING----------------------------
110
111   labels = data_train["labels"].values
112   inputs = data_train.drop(["labels"], axis=1).values
113   inputs = scaler.transform(inputs)
114
115   X_train = []
116   y_train = []
117   i=0
118   step = (1-overlap)*n_timesteps
119   if step == 0:
120      step =1
121   while i<inputs.shape[0]:
122      end_index = i + n_timesteps
123
124      if end_index >= inputs.shape[0]:
125         break;
126
127      X_temp = inputs[i:end_index,:]
128
129      if labelling_strategy == "last":
130         y_temp = labels[end_index-1]
131
132      if labelling_strategy == "frequency":
133         y_temp = stats.mode(labels[i:end_index])[0]
134
```

```
135         X_train.append(X_temp)
136         y_train.append(y_temp)
137 
138         i = int(i + step)
139 
140     X_train = np.array(X_train)
141     y_train = np.array(y_train)
142 
143     indices_maintenance = np.argwhere(y_train=='Ausfallzeit - Wartung')
144 
145     X_train = np.delete(X_train,indices_maintenance,0)
146     y_train = np.delete(y_train,indices_maintenance,0)
147 
148     #indices_sel = np.where((y_train=='Hauptzeit - Verrohrung abnehmen') |
            ↪  (y_train=='Hauptzeit - Verrohrung eindrehen') | (y_train==''
            ↪ Hauptzeit - Bohren') | (y_train=='Hauptzeit - Leeren') | (
            ↪ y_train=='Hauptzeit - Ziehen') | (y_train=='Hauptzeit - Ablassen
            ↪ ') | (y_train=='Hauptzeit - Standrohr setzen') | (y_train==''
            ↪ Hauptzeit - Standrohr ziehen'))
149     #indices_sel = np.where((y_train=='Hauptzeit - Verrohrung eindrehen')
            ↪ | (y_train=='Hauptzeit - Bohren') | (y_train=='Hauptzeit -
            ↪ Leeren') | (y_train=='Hauptzeit - Ziehen') | (y_train==''
            ↪ Hauptzeit - Ablassen') | (y_train=='Hauptzeit - Standrohr setzen
            ↪ ') | (y_train=='Hauptzeit - Bohren - Bohreimer'))
150     indices_sel = np.where((y_train=='Hauptzeit - Betonieren') | (y_train
            ↪ =='Hauptzeit - Sch ttrohr - Setzen') | (y_train=='Hauptzeit - 
            ↪ Verrohrung abnehmen') | (y_train=='Hauptzeit - Sch ttrohr - 
            ↪ Entfernen') | (y_train=='Hauptzeit - Standrohr ziehen'))
151     #indices_sel = np.where((y_train=='Ausfallzeit - Pause') | (y_train==''
            ↪ Ausfallzeit - Schaden') | (y_train=='Ausfallzeit - Wartung') | (
            ↪ y_train=='Ausfallzeit - Warten auf - Beton') | (y_train==''
            ↪ Ausfallzeit - Warten auf - Sonstifes'))
152     #indices_sel = np.where((y_train=='Hauptzeit - Bewehrung einbauen') |
            ↪ (y_train=='Hauptzeit - Einbau Kiespolster'))
153     #indices_sel = np.where((y_train=='Hauptzeit -  Verrohrungsanlage -
            ↪ Standrohr ziehen') | (y_train=='Hauptzeit - Verrohrungsanlage -
            ↪ Verrohrung eindrehen'))
154     #indices_sel = np.where((y_train=='Nebenprozesszeit - Sonstiges') | (
            ↪ y_train=='Nebenprozesszeit - Tanken') | (y_train==''
            ↪ Nebenprozesszeit - Tiefenmessung') | (y_train=='Nebenprozesszeit
            ↪  - Umsetzen') | (y_train=='Nebenprozesszeit - Wasser nachf llen
```

```
        ↪ ') | (y_train=='Nebenprozesszeit - Werkzeugwechsel'))
155     #indices_sel = np.where((y_train=='Drilling') | (y_train=='Concreting
        ↪ ') | (y_train=='Reinforcing')| (y_train=='Casing machine'))
156     #indices_sel = np.where((y_train=='Secondary process') | (y_train=='
        ↪ Downtime'))
157
158     X_train = np.take(X_train,indices=indices_sel,axis=0)
159     X_train = np.squeeze(X_train, axis = 0)
160     y_train = np.take(y_train,indices=indices_sel,axis=0)
161
162     y_train = label_binarize(y_train.squeeze(),unique_labels)
163
164     #------------------VALIDATION---------------------------
165
166     labels = data_val["labels"].values
167     inputs = data_val.drop(["labels"], axis=1).values
168     inputs = scaler.transform(inputs)
169
170     X_valid = []
171     y_valid = []
172     i=0
173     step = (1-overlap)*n_timesteps
174     if step == 0:
175         step =1
176     while i<inputs.shape[0]:
177         end_index = i + n_timesteps
178
179         if end_index >= inputs.shape[0]:
180             break;
181
182         X_temp = inputs[i:end_index,:]
183
184         if labelling_strategy == "last":
185             y_temp = labels[end_index-1]
186
187         if labelling_strategy == "frequency":
188             y_temp = stats.mode(labels[i:end_index])[0]
189
190         X_valid.append(X_temp)
191         y_valid.append(y_temp)
192
```

```
193         i = int(i + step)
194
195     X_valid = np.array(X_valid)
196     y_valid = np.array(y_valid)
197
198     indices_maintenance = np.argwhere(y_valid=='Ausfallzeit - Wartung')
199
200     X_valid = np.delete(X_valid,indices_maintenance,0)
201     y_valid = np.delete(y_valid,indices_maintenance,0)
202
203     #indices_sel = np.where((y_valid=='Hauptzeit - Verrohrung abnehmen') |
            ↪  (y_valid=='Hauptzeit - Verrohrung eindrehen') | (y_valid==''
            ↪ Hauptzeit - Bohren') | (y_valid=='Hauptzeit - Leeren') | (
            ↪ y_valid=='Hauptzeit - Ziehen') | (y_valid=='Hauptzeit - Ablassen
            ↪ ') | (y_valid=='Hauptzeit - Standrohr setzen') | (y_valid==''
            ↪ Hauptzeit - Standrohr ziehen'))
204     #indices_sel = np.where((y_valid=='Hauptzeit - Verrohrung eindrehen')
            ↪ | (y_valid=='Hauptzeit - Bohren') | (y_valid=='Hauptzeit -
            ↪ Leeren') | (y_valid=='Hauptzeit - Ziehen') | (y_valid==''
            ↪ Hauptzeit - Ablassen') | (y_valid=='Hauptzeit - Standrohr setzen
            ↪ ') | (y_valid=='Hauptzeit - Bohren - Bohreimer'))
205     indices_sel = np.where((y_valid=='Hauptzeit - Betonieren') | (y_valid
            ↪ =='Hauptzeit - Sch ttrohr - Setzen') | (y_valid=='Hauptzeit -
            ↪ Verrohrung abnehmen') | (y_valid=='Hauptzeit - Sch ttrohr -
            ↪ Entfernen') | (y_valid=='Hauptzeit - Standrohr ziehen'))
206     #indices_sel = np.where((y_valid=='Ausfallzeit - Pause') | (y_valid==''
            ↪ Ausfallzeit - Schaden') | (y_valid=='Ausfallzeit - Wartung') | (
            ↪ y_valid=='Ausfallzeit - Warten auf - Beton') | (y_valid==''
            ↪ Ausfallzeit - Warten auf - Sonstifes'))
207     #indices_sel = np.where((y_valid=='Hauptzeit - Bewehrung einbauen') |
            ↪ (y_valid=='Hauptzeit - Einbau Kiespolster'))
208     #indices_sel = np.where((y_valid=='Hauptzeit -  Verrohrungsanlage -
            ↪ Standrohr ziehen') | (y_valid=='Hauptzeit - Verrohrungsanlage -
            ↪ Verrohrung eindrehen'))
209     #indices_sel = np.where((y_valid=='Nebenprozesszeit - Sonstiges') | (
            ↪ y_valid=='Nebenprozesszeit - Tanken') | (y_valid==''
            ↪ Nebenprozesszeit - Tiefenmessung') | (y_valid=='Nebenprozesszeit
            ↪ - Umsetzen') | (y_valid=='Nebenprozesszeit - Wasser nachf llen
            ↪ ') | (y_valid=='Nebenprozesszeit - Werkzeugwechsel'))
210     #indices_sel = np.where((y_valid=='Drilling') | (y_valid=='Concreting
            ↪ ') | (y_valid=='Reinforcing')| (y_valid=='Casing machine'))
```

```
211    #indices_sel = np.where((y_valid=='Secondary process') | (y_valid=='
           ↪ Downtime'))
212
213
214    X_valid = np.take(X_valid,indices=indices_sel,axis=0)
215    X_valid = np.squeeze(X_valid, axis = 0)
216    y_valid = np.take(y_valid,indices=indices_sel,axis=0)
217
218    y_valid = label_binarize(y_valid.squeeze(),unique_labels)
219
220
221    #------------------TEST-------------------------
222
223    labels = data_test["labels"].values
224    inputs = data_test.drop(["labels"], axis=1).values
225    inputs = scaler.transform(inputs)
226
227    X_test = []
228    y_test = []
229    i=0
230    step = (1-overlap)*n_timesteps
231    if step == 0:
232       step =1
233    while i<inputs.shape[0]:
234       end_index = i + n_timesteps
235
236       if end_index >= inputs.shape[0]:
237          break;
238
239       X_temp = inputs[i:end_index,:]
240
241       if labelling_strategy == "last":
242          y_temp = labels[end_index-1]
243
244       if labelling_strategy == "frequency":
245          y_temp = stats.mode(labels[i:end_index])[0]
246
247       X_test.append(X_temp)
248       y_test.append(y_temp)
249
250       i = int(i + step)
```

```
251
252     X_test = np.array(X_test)
253     y_test = np.array(y_test)
254
255     indices_maintenance = np.argwhere(y_test=='Ausfallzeit - Wartung')
256
257     X_test = np.delete(X_test,indices_maintenance,0)
258     y_test = np.delete(y_test,indices_maintenance,0)
259
260     #indices_sel = np.where((y_test=='Hauptzeit - Verrohrung abnehmen') |
            ↪ (y_test=='Hauptzeit - Verrohrung eindrehen') | (y_test=='
            ↪ Hauptzeit - Bohren') | (y_test=='Hauptzeit - Leeren') | (y_test
            ↪ =='Hauptzeit - Ziehen') | (y_test=='Hauptzeit - Ablassen') | (
            ↪ y_test=='Hauptzeit - Standrohr setzen') | (y_test=='Hauptzeit -
            ↪ Standrohr ziehen'))
261     #indices_sel = np.where((y_test=='Hauptzeit - Verrohrung eindrehen') |
            ↪  (y_test=='Hauptzeit - Bohren') | (y_test=='Hauptzeit - Leeren')
            ↪  | (y_test=='Hauptzeit - Ziehen') | (y_test=='Hauptzeit -
            ↪ Ablassen') | (y_test=='Hauptzeit - Standrohr setzen') | (y_test
            ↪ =='Hauptzeit - Bohren - Bohreimer'))
262     indices_sel = np.where((y_test=='Hauptzeit - Betonieren') | (y_test=='
            ↪ Hauptzeit - Sch ttrohr - Setzen') | (y_test=='Hauptzeit -
            ↪ Verrohrung abnehmen') | (y_test=='Hauptzeit - Sch ttrohr -
            ↪ Entfernen') | (y_test=='Hauptzeit - Standrohr ziehen'))
263     #indices_sel = np.where((y_test=='Ausfallzeit - Pause') | (y_test=='
            ↪ Ausfallzeit - Schaden') | (y_test=='Ausfallzeit - Wartung') | (
            ↪ y_test=='Ausfallzeit - Warten auf - Beton') | (y_test=='
            ↪ Ausfallzeit - Warten auf - Sonstifes'))
264     #indices_sel = np.where((y_test=='Hauptzeit - Bewehrung einbauen') | (
            ↪ y_test=='Hauptzeit - Einbau Kiespolster'))
265     #indices_sel = np.where((y_test=='Hauptzeit -  Verrohrungsanlage -
            ↪ Standrohr ziehen') | (y_test=='Hauptzeit - Verrohrungsanlage -
            ↪ Verrohrung eindrehen'))
266     #indices_sel = np.where((y_test=='Nebenprozesszeit - Sonstiges') | (
            ↪ y_test=='Nebenprozesszeit - Tanken') | (y_test=='
            ↪ Nebenprozesszeit - Tiefenmessung') | (y_test=='Nebenprozesszeit
            ↪ - Umsetzen') | (y_test=='Nebenprozesszeit - Wasser nachf llen')
            ↪  | (y_test=='Nebenprozesszeit - Werkzeugwechsel'))
267     #indices_sel = np.where((y_test=='Drilling') | (y_test=='Concreting')
            ↪ | (y_test=='Reinforcing')| (y_test=='Casing machine'))
268     #indices_sel = np.where((y_test=='Secondary process') | (y_test=='
```

```
        ↪ Downtime ') )
269
270     X_test = np.take(X_test,indices=indices_sel,axis=0)
271     X_test = np.squeeze(X_test, axis = 0)
272     y_test = np.take(y_test,indices=indices_sel,axis=0)
273
274     y_test = label_binarize(y_test.squeeze(),unique_labels)
275
276     # ----------------------------------
277
278     if n_timesteps == 1:
279       X_train = X_train.squeeze()
280       X_test = X_test.squeeze()
281       X_valid = X_valid.squeeze()
282
283     return X_train, y_train, X_valid, y_valid, X_test, y_test,
          ↪ unique_labels
```

A.3 Models

A.3.1 MLP

```
1  def DenseNN(n_features, n_outputs, n_layers, neuros_per_layer,
       ↪ activation, dropout, optimizer):
2       temp = range(n_layers)
3       init = keras.initializers.glorot_uniform(seed=1)
4       model = keras.models.Sequential()
5       model.add(keras.layers.InputLayer(input_shape=n_features))
6
7       for k in temp:
8           if k<n_layers-1:
9               model.add(keras.layers.Dense(neuros_per_layer,
                    ↪ kernel_initializer=init, activation = activation))
10              model.add(keras.layers.Dropout(dropout))
11          else:
12              model.add(keras.layers.Dense(n_outputs, kernel_initializer=
                    ↪ init,activation = "softmax"))
13
14      model.compile( loss="categorical_crossentropy", metrics = "accuracy"
            ↪ , optimizer = optimizer)
15
16      return model
```

A.3.2 LSTM

```
def Vanilla_LSTM(n_features, n_outputs, lstm_size, lstm_act, dropout,
    ↪ optimizer):
    model = keras.models.Sequential()

    for i in range(lstm_size[0]):
        if i==0:
            model.add(keras.layers.LSTM(lstm_size[1], activation =
                ↪ lstm_act ,return_sequences = True, dropout = dropout,
                ↪ input_shape = (None,n_features)))
        elif i<lstm_size[0]-1:
            model.add(keras.layers.LSTM(lstm_size[1], activation =
                ↪ lstm_act ,return_sequences = True, dropout = dropout))
        else:
            model.add(keras.layers.LSTM(lstm_size[1], activation =
                ↪ lstm_act ,return_sequences = False, dropout = dropout)
                ↪ )

    model.add(keras.layers.Dense(n_outputs, activation = 'softmax'))

    model.compile(optimizer = optimizer, loss="categorical_crossentropy"
        ↪ , metrics = "accuracy")

    return model
```

A.3.3 DeepConvLSTM

```
def deep_conv_LSTM(n_features, n_outputs, timesteps, cnn_size, cnn_act,
    ↪ lstm_size, lstm_act, batch_norm, dropout, optimizer):
    model = keras.models.Sequential()

    for i in range(cnn_size[0]):
        if i == 0:
            model.add(keras.layers.Conv1D(cnn_size[1],cnn_size[2],
                ↪ input_shape=(timesteps, n_features), padding="same"))
        else:
            model.add(keras.layers.Conv1D(cnn_size[1],cnn_size[2],
                ↪ padding="same"))

        if batch_norm == True:
            model.add(keras.layers.BatchNormalization())

```

```
13          model.add(keras.layers.Activation(cnn_act))
14
15      for i in range(lstm_size[0]):
16          if i==0:
17              model.add(keras.layers.LSTM(lstm_size[1], activation =
                    ↪ lstm_act ,return_sequences = True, dropout = dropout,
                    ↪ input_shape = (None,n_features)))
18          elif i<lstm_size[0]-1:
19              model.add(keras.layers.LSTM(lstm_size[1], activation =
                    ↪ lstm_act ,return_sequences = True, dropout = dropout))
20          else:
21              model.add(keras.layers.LSTM(lstm_size[1], activation =
                    ↪ lstm_act ,return_sequences = False, dropout = dropout)
                    ↪ )
22
23      model.add(keras.layers.Dense(n_outputs, activation = 'softmax'))
24
25      model.compile(optimizer = optimizer, loss="categorical_crossentropy"
            ↪ , metrics = "accuracy")
26
27      return model
```

A.3.4 DeepConvBiLSTM

```
1   def deep_conv_BiLSTM(n_features, n_outputs, timesteps, cnn_size, cnn_act
        ↪ , lstm_size,  lstm_act, batch_norm, dropout, optimizer):
2       model = keras.models.Sequential()
3
4       for i in range(cnn_size[0]):
5           if i == 0:
6               model.add(keras.layers.Conv1D(cnn_size[1],cnn_size[2],
                    ↪ input_shape=(timesteps, n_features)))
7           else:
8               model.add(keras.layers.Conv1D(cnn_size[1],cnn_size[2]))
9
10          if batch_norm == True:
11              model.add(keras.layers.BatchNormalization())
12
13          model.add(keras.layers.Activation(cnn_act))
14
15      for i in range(lstm_size[0]):
16          if i==0:
```

```
17              model.add(keras.layers.Bidirectional(keras.layers.LSTM(
                    ↪ lstm_size[1], activation = lstm_act ,return_sequences
                    ↪ = True, dropout = dropout), input_shape = (None,
                    ↪ n_features)))
18          elif i<lstm_size[0]-1:
19              model.add(keras.layers.Bidirectional(keras.layers.LSTM(
                    ↪ lstm_size[1], activation = lstm_act ,return_sequences
                    ↪ = True, dropout = dropout)))
20          else:
21              model.add(keras.layers.Bidirectional(keras.layers.LSTM(
                    ↪ lstm_size[1], activation = lstm_act ,return_sequences
                    ↪ = False, dropout = dropout)))
22
23      model.add(keras.layers.Dense(n_outputs, activation = 'softmax'))
24
25      model.compile(optimizer = optimizer, loss="categorical_crossentropy"
            ↪ , metrics = "accuracy")
26      #model.compile(optimizer = optimizer, loss="binary_crossentropy",
            ↪ metrics = "accuracy")
27
28      return model
```

B Appendix

B.1 Histograms

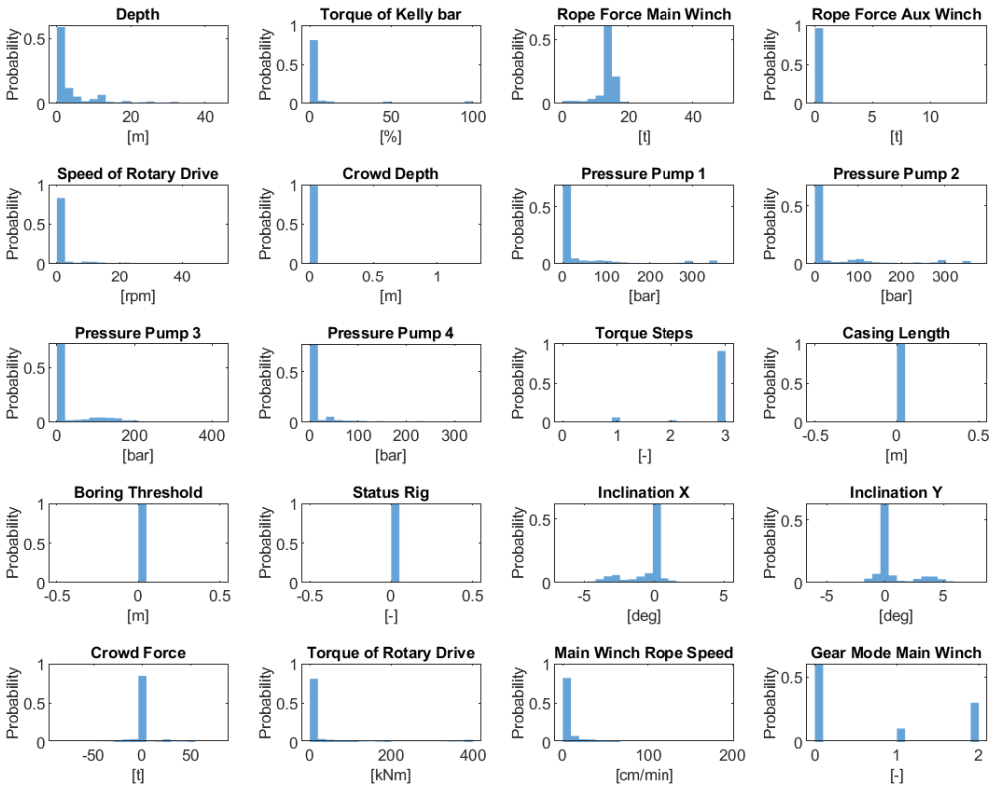
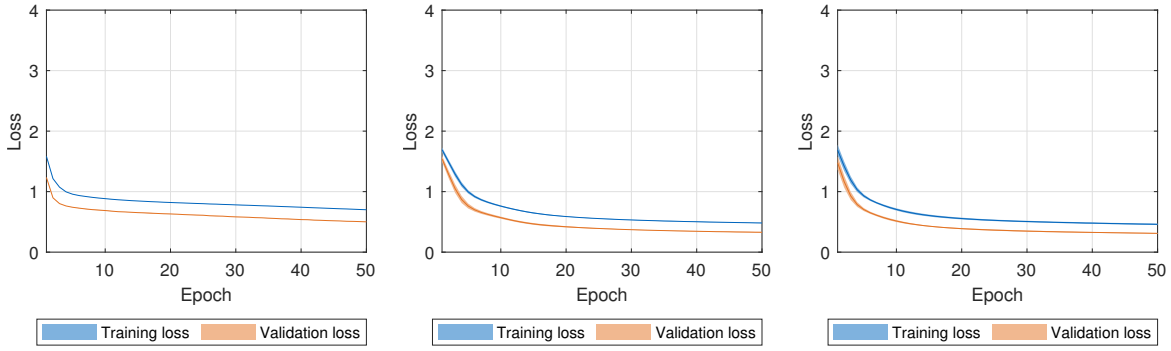


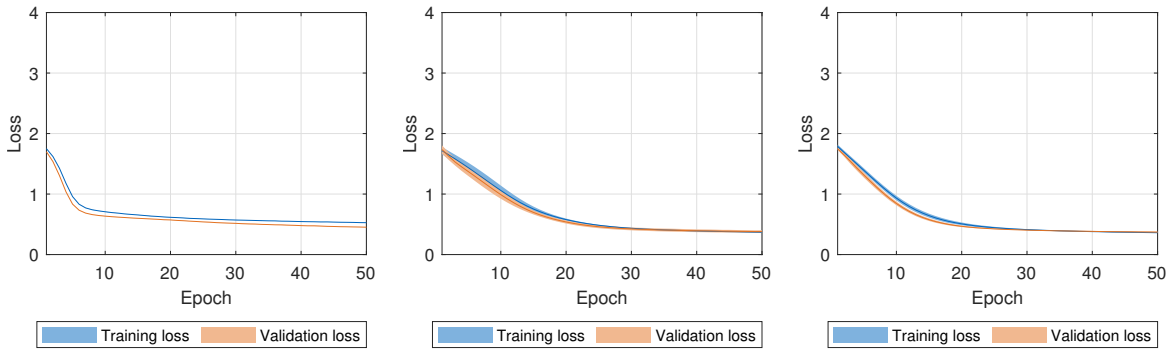
Figure B-1: Histograms of the data of the available sensors

B.2 Loss plots



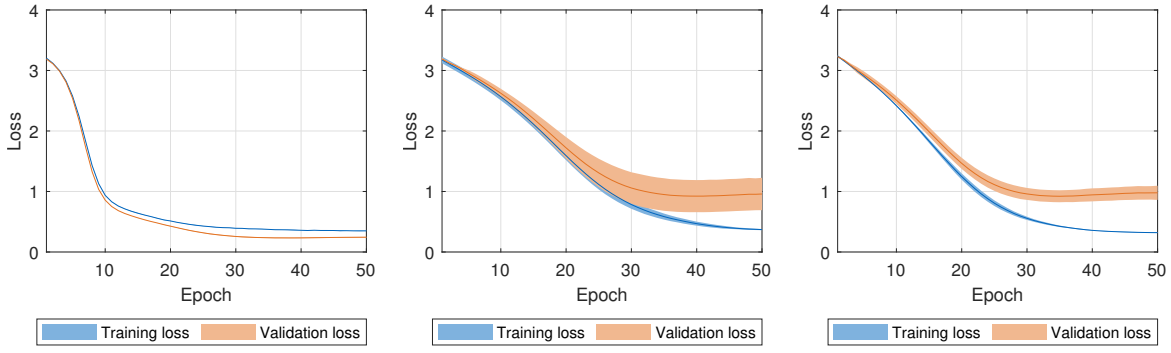
(a) MLP (b) DeepConvLSTM (c) DeepConvBiLSTM

Figure B-2: Training and validation loss for different models - LoD 2 - Working



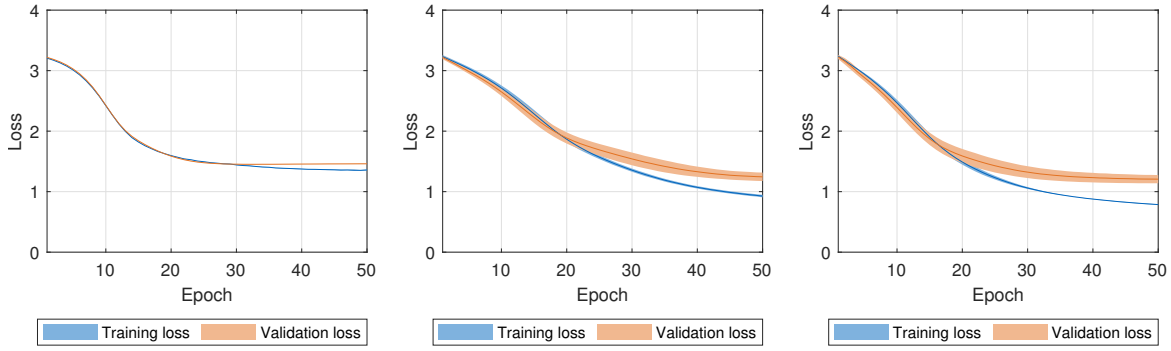
(a) MLP (b) DeepConvLSTM (c) DeepConvBiLSTM

Figure B-3: Training and validation loss for different models - LoD 2 - Idle



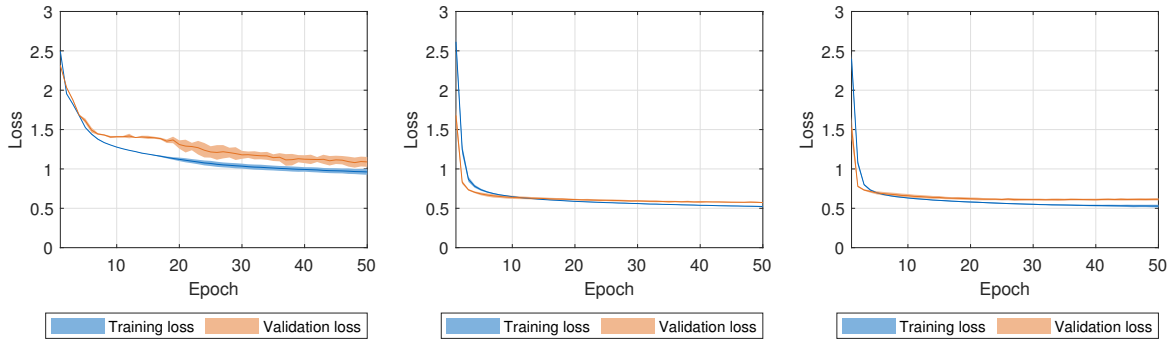
(a) MLP (b) DeepConvLSTM (c) DeepConvBiLSTM

Figure B-4: Training and validation loss for different models - LoD 3 - Reinforcing



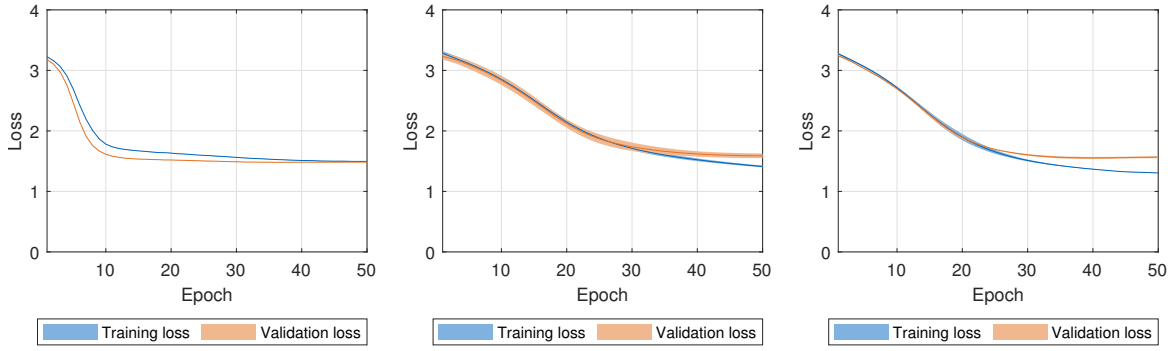
(a) MLP (b) DeepConvLSTM (c) DeepConvBiLSTM

Figure B-5: Training and validation loss for different models - LoD 3 - Secondary processes



(a) MLP (b) DeepConvLSTM (c) DeepConvBiLSTM

Figure B-6: Training and validation loss for different models - LoD 3 - Drilling



(a) MLP (b) DeepConvLSTM (c) DeepConvBiLSTM

Figure B-7: Training and validation loss for different models - LoD 3 - Concreting

B.3 Confusion matrices

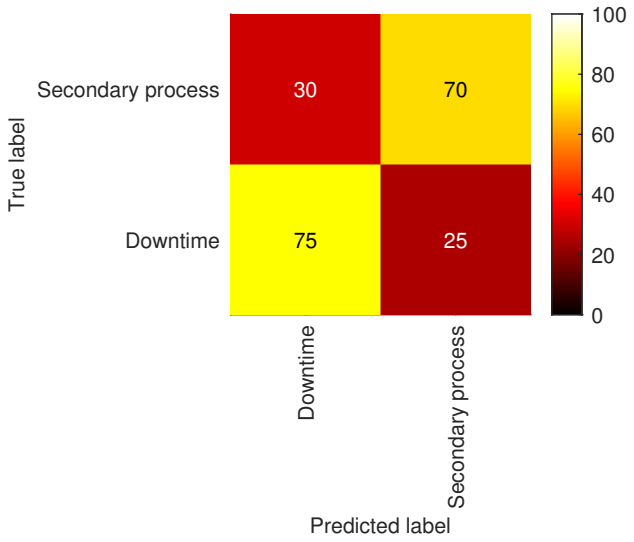
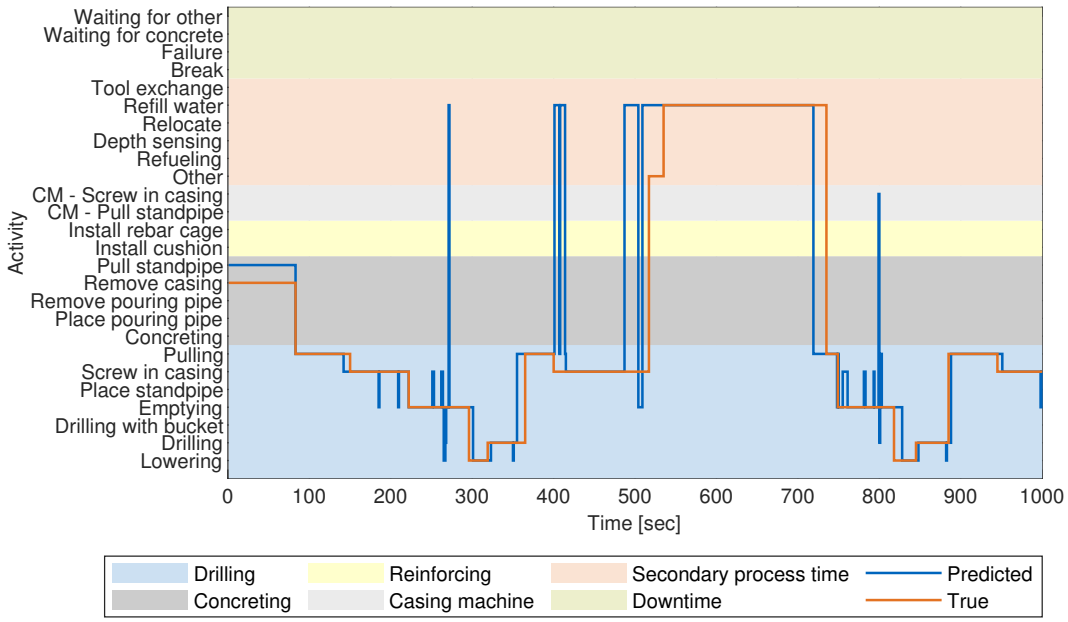


Figure B-8: Confusion matrix - LoD 2 - Idle - DeepConvBiLSTM

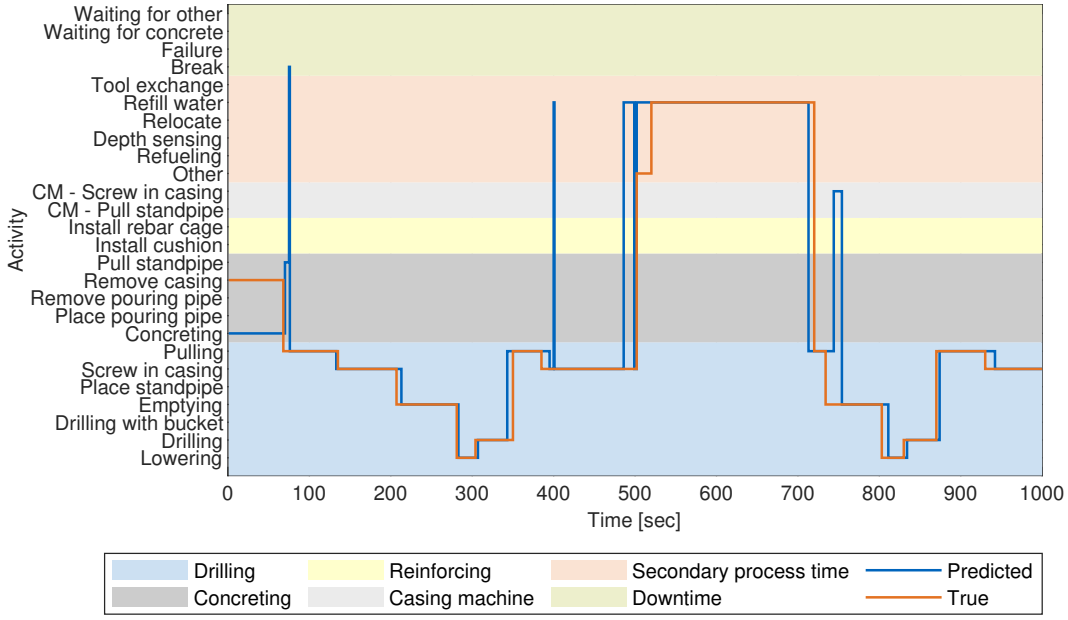
(a) DeepConvLSTM *(b) DeepConvBiLSTM*

Figure B-9: Confusion matrix - LoD 3 - Concreting

B.4 Predictions



(a) MLP



(b) DeepConvLSTM$_{64}$

Figure B-10: True and predicted labels for a selected interval (2)

Declaration

I hereby certify that I have written the thesis submitted by me independently and that I have
I have not used any sources or aids other than those indicated.

Garching, 15.06.2021