Root Cause Analysis in Sparsely Labeled Environments using Machine Learning

Dissertation

zur Erlangung des Doktorgrades
der Naturwissenschaften (Dr. rer. nat.)
der Fakultät Physik
der Universität Regensburg

vorgelegt von

Marinus Bommer

aus Garmisch-Partenkirchen

im Jahr 2022

Promotionsgesuch eingereicht am 18.1.2022.

Die Arbeit wurde in Zusammenarbeit mit Syskron GmbH,
am Institut für Biophysik unter der
Anleitung von Prof. Dr. Elmar W. Lang durchgeführt.

Prüfungsausschuss:

Vorsitzender:	Prof. Dr. Dominique Bougeard
1. Gutachter:	Prof. Dr. Elmar W. Lang
2. Gutachter:	Dr. Stefan Solbrig
weiterer Prüfer:	Prof. Dr. Jascha Repp
Ersatzprüfer:	Prof. Dr. Florian Hartig

Contents

Nomenclature

Symbols describing the line and data

$\mathbf{A}$	Adjacency matrix
$\mathbf{C}$	Set of categories
$\mathbf{M}$	Set of machines
$\mathbf{X}$	Set of examples
$\mathbf{x}$	one example
$\mathbf{Y}$	Set of labels
$\mathbf{y}$	one label
C	Number of categories
m_i	Machine i
M	Number of machines
N	Number of samples
T	Number of time-steps
$\mathcal{T}$	time space
$\mathcal{X}$	feature space
$\mathcal{Y}$	label space

Abstract

Current filling and packaging lines in liquid food industry are highly optimized and process up to 120000 units per hour. To keep performance at a high level, an ongoing analysis process to identify and consequently solve weaknesses inside the line must take place. To support bottling companies in this task, we present and compare various algorithmic approaches to find the root-cause machine for each line stoppage. We introduce a rule-based algorithm as well as an implementation using Random Forest and two different Relational Graph Convolutional Neural Networks. Since information about the true causer machine is very limited, all approaches have to be label-efficient. Therefore, we introduce Active Learning, Algorithmic Labeling, Multi-Task Learning and Transfer Learning. Finally, a new fully-convolutional Neural Network architecture enables us to predict root-cause machines at any line with only needing one trained model.

1 Introduction

This thesis was conducted in close collaboration between the *Computational Intelligence and Machine Learning* group of Prof. Elmar Lang at University of Regensburg and *Syskron GmbH*. As a part of the Krones AG family, Syskron is a globally acting company with the goal of leading the bottling industry into Industry 4.0. The company gathers data of bottling lines and converts them to valuable information for the end-user. One important field is the analysis of error propagation in a bottling line. There are two main reasons: On the one hand, modern production lines are designed to process up to 120,000 units per hour. The high speeds lead to a high susceptibility to faults. On the other hand, line stoppages should be avoided for profitability and quality reasons. Contrary to the strong demand for failure analyses, even in modern bottling companies, the root cause of a problem has to be found and solved by an expert with little digital help. This includes the documentation and consecutive analysis of detecting the main error sources in the line. Altogether, this procedure is not a sustainable for bottling lines increasing in number and size.

The goal of this thesis is to develop an algorithm for automatically detecting the error-causing machine (root cause) for each line stoppage. It should meet the requirements of providing a high accuracy, low costs per roll-out at one filling line and being immediately usable after roll-out. Based on the algorithm's results, the analysis of the main error causes over shifts, days or weeks can be automatically executed in Syskron's Share2Act platform. These statistics are a valuable basis for the maintenance staff and are used to identify and correct the causes of frequent failures, thus improving line efficiency [1].

To gain good accuracies, this task can be solved in two ways: Either by using a rule based system that is set up and configured by experts, or by using Supervised Machine Learning based on experts' labels.

To make these approaches scalable for roll-out on hundreds of filling and packaging lines, an expert's effort per adaption to a new line has to be minimized. On the one hand, this can be achieved by automatizing the rule-based systems' configuration

for each line and on the other hand by avoiding the need of labeling large amounts of data per line for Machine Learning.

For the first option, an unsupervised algorithm to extract temporal characteristics of error propagation in a filling and packaging line is introduced. The results are used to automatically configure rule-based algorithms for root cause analysis. During this work, the automatic configuration is implemented for Syskron's existing 'Avalanche'-algorithm as well as for the newly introduced rule-based algorithm that uses graph theory to represent the line structure and a probabilistic description of temporal error propagation through the production line.

The second option to decrease expert's effort uses the area of Machine Learning approaches: Active Learning and Multi-Task Learning in combination with Transfer Learning as well as the usage of algorithmically created labels are applied to limit labeling effort.

In terms of Machine Learning models, I evaluate Random Forest, Graph Convolution Neural Network and a newly created spatio-temporal Graph Neural Network architecture and compare them to the baseline of Dynamic Time Warping and k-Nearest Neighbors.

For evaluation and comparison of the different approaches, test datasets of four different Syskron customers, each containing hundreds of line stoppages were collected and labeled by experts. The amount of training data was increased from a few hundred examples in previous works (e.g. [2]), to the order of 10 000 which allows for proper training of state-of-the-art Neural Network architectures.

2 Theoretical background

In this chapter, the reader will be introduced to the background behind extracting the root-cause machine of a line stoppage. Therefore, the world of a filling and packaging line including the components and the control mechanisms is presented, and the properties of error propagation through a line are explained. Afterwards, the available data and different data preprocessing approaches are shown. The field of time-series classification in general is presented and an introduction to the corresponding algorithms is given. This includes Dynamic Time Warping as baseline model, as well as Random Forest and Graph Convolution Networks. To tackle the problem of only sparsely labeled data, tools from the field of Weak Supervision including Active Learning, Multi-Task Learning, Transfer Learning and Algorithmic Labeling will be used in this work and presented to the reader in this chapter. Last but not least an overview over related work is given.

2.1 Bottling Plant

In this work we use data from Krones filling and packaging lines. These can be very diverse, ranging from processing cans over plastic-bottles to glass-bottles. Additionally, there are bottling lines using new bottles and ones that use returnable bottles. Finally, every line is custom tailored to the needs of the customer, his building and his products. These points show that generic analysis is a challenging task. But there is standardized data that can be collected from every machine and there are reoccurring patterns from a line perspective that can be used as a data-basis to develop algorithms that yield great added value.

In this chapter the basic structure and behavior of filling and packaging lines is described. Thus the background of the used data is presented.

2.1.1 Manufacturing flow lines

The definition of a production line in general can be phrased like the following:

> *""Manufacturing flow line systems consist of material, work areas, and storage areas. Material flows from work area to storage area to work area; it visits each work and storage area exactly once in a fixed sequence""*[3]

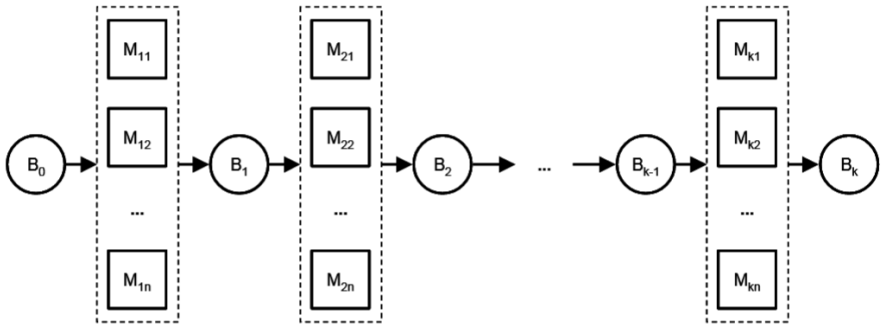


Figure 2.1 *schema of production line with k work areas M consisting of n machines each. $k+1$ buffers B are used for storing the material in between work-steps [2]*

In this work the terms "Manufacturing flow line" and "production line" are used synonymously. A generalized schema for this kind of line is depicted in Figure 2.1 and describes the material flow through storage areas and work-areas. Storage areas act as buffers between the manufacturing steps and thus enable a work area to keep producing for a while although e.g. the previous work step has no material output due to some error.

Packaging and bottling lines represent a special case of a manufacturing flow line and differ from the general schema by some special characteristics. These will be presented in detail in the following.

Different machine speeds

While a production line in general shows constant production speed over all work areas, the machine speeds along the bottle flow of a bottling line are not constant. This is based on presence of a central aggregate, also called lead-machine. The main task is to achieve a continuous operation of the lead-machine. For that reason the buffers between the machines are optimized to detach the lead-machine as much as possible from failures of surrounding aggregates. As an example, the buffer previous

to the lead-machine should be as full as possible to enable the lead aggregate a maximal duration of ongoing production during an upstream failure along the bottle flow.

Let's further examine this example and assume that the failure was corrected. Now the mentioned buffer, of course, has less population than before. That amount of missing bottles in the buffer has to be refilled. Therefore, the previous machines have to be able to operate faster than the lead-machine. This is called *in over-performance*. According to this example, the whole line is built up based on a V-diagram (compare Figure 2.2). This shows the machine speeds along the bottle flow of a filling and packaging line. Usually the filling machine is the lead-machine for qualitative reasons, e.g. prevention of warming and oxidation of the product due to frequent stops.

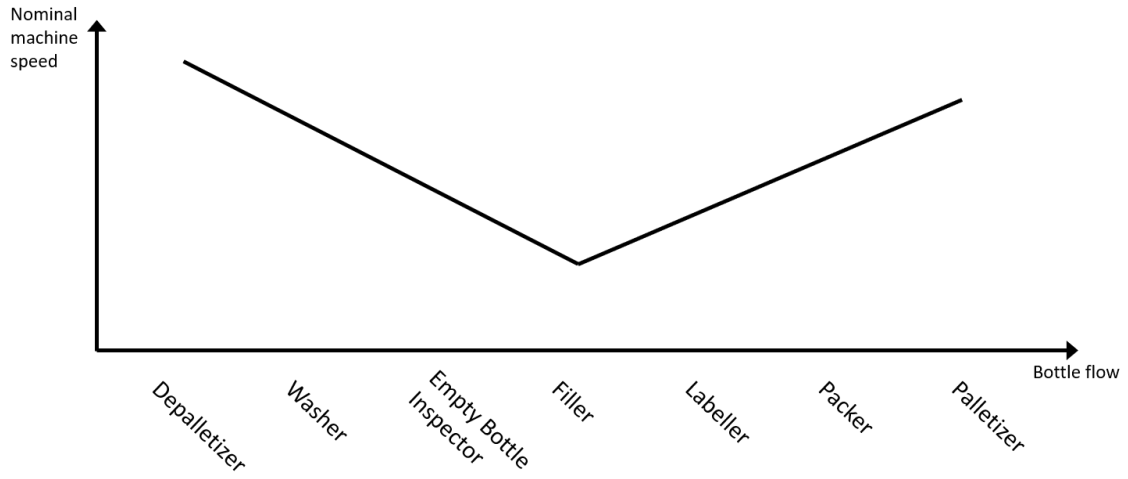


Figure 2.2 *Graph showing machine speed against the main stream of a filling line, exemplary for machines of a returnable glass line. It needs to have a V-shape to secure the best possible capacity utilization of the lead-machine (Filler). This shape is also called* Berg'sche Kurve *[4].*

High susceptibility to faults

Due to very high production speeds of e.g. 60000 bottles per hour (almost 17 bottles per second) for a glass line and up to 120000 cans per hour in a can line and the corresponding need for high mechanical precision the chances of aggregate failure are significantly higher than they were at slower production speeds. Thus the production

line has to be prepared to absorb most failures that appear in the line by exploiting its buffers. Additionally, there is always the need to find the source of frequent errors in a line to be able to eliminate the root cause.

Run-through buffer

Unlike usual production lines (which use e.g. pallet racks), buffering in the filling and packaging industry is done on conveyors. In general there are three types of conveyors: Pallet-, crate- and bottle-conveyors. For pallets and crates the conveyors are FIFO-(First In First Out)-buffers. This category includes all one-lane conveyors. For the buffering effect, they have to be large enough to be able to hold a sufficient amount of material to buffer minor machine-stops. In the case of bottle transport

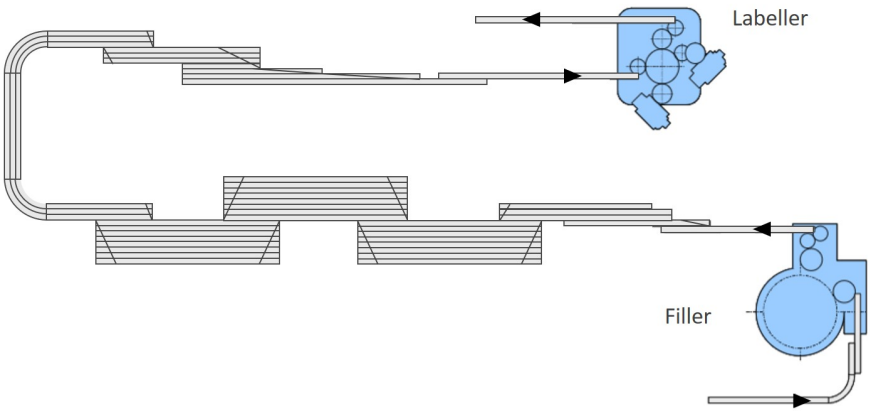


Figure 2.3 *Example for a multi-lane bottle transportation conveyor, view from above (adapted from [2])*

we have to look a little closer (compare Figure 2.3). Inside the machines, bottles are usually processed in a single-line. Therefore, the conveyors directly before and after these machines are single-lined, too. But as mentioned above, due to high production speeds a single-line transport of bottles is erroneous. In mass-transport (multi-line), the speed of each single bottle is heavily decreased and thus also the relative speed between bottles e.g. at the edge regions of a tailback area are decreased. This results in multiple advantages like noise reduction, less falling bottles and the possibility to buffer.

Multi-line conveyors provide buffer capacity according to the following principle: If the transporteur is empty and the first bottles are transported, they are led through

the multi-line section without spreading to the full width but staying one or two (out of e.g. 8) lanes. Thus the transportation speed stays high and the first bottles arrive at the upcoming machine fast. If the previous machine has a higher production speed than the upcoming one, there will be a bottle-tailback on the conveyor. In this case the bottles will slowly spread up on the full width of the conveyor and the buffer occupancy is rising. Occasionally, filling and packaging lines also possess buffers that rely on principles besides the run-through buffer. One example is a tower buffer for empty crates that works according to Last In First Out (LIFO) principle.

Auxiliary flows

Besides the main flow (bottle flow) there are multiple auxiliary flows in a returnable-bottle line: On the one hand empty pallets are transported e.g. from the depalletizer to the palletizer, on the other hand there is a connection for empty crates between unpacker and packer. Besides these two main auxiliary connections in a returnable glass line, there are of course many auxiliary material inputs: e.g. the product itself, caps, labels, new bottles to counterbalance rejects and foil for wrapping the pallets.

2.1.2 Common machines

The data in this work has its origin mostly in returnable glass lines that are usually used e.g. by breweries (compare exemplary architecture in Figure 2.4). Therefore the usually installed machines are described in the following. A similar description can also be found in [5]. Generally there are two types of machines: cycle machines (stepwise working) and those with constant material flow through the machine. In the latter category one can again distinguish between machines consisting of a carousel that carries bottles a little less than one turn and processes them within this period and those machines that process bottles while they are on a multi-lane conveyor.

Depalletizer

Since, as mentioned, used bottles are filled in returnable lines the pallets carrying empty bottles have to be unloaded in the first step. This is done by the Depalletizer. It takes crates with empty bottles from the pallet layer by layer and puts them on the crate transport while empty pallets are afterwards transported to the Palletizer. Since the process steps are done one after another in a temporal view, this machine

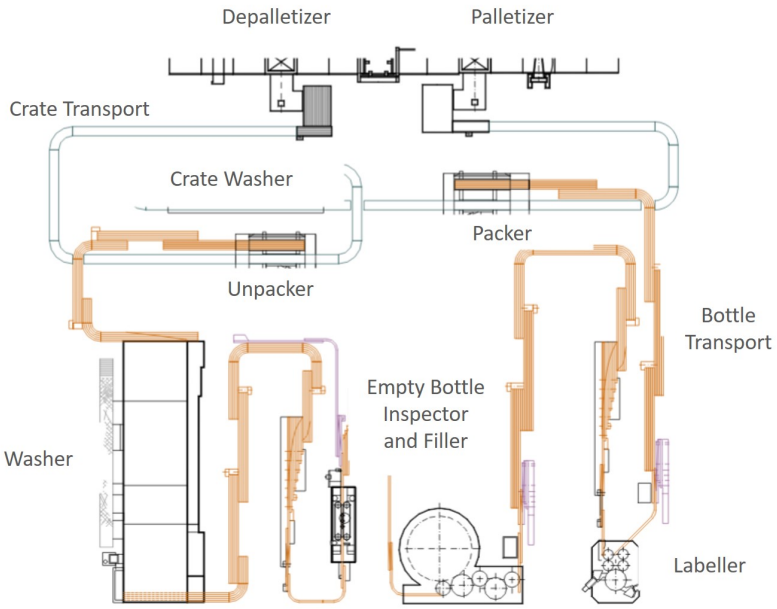


Figure 2.4 *exemplary architecture of a filling and packaging line for returnable glass-bottles like e.g. for a brewery. (adapted from [2])*

is an example of a cycle-machine with following steps: firstly the pallet is driven into the machine, afterwards, the highest layer of crates is lifted and placed on conveyor belt by a robot, finally, the crates are moved out of the machine. And only after finishing the previous working step the next one is started.

Unpacker

The Unpacker takes crates with empty bottles and lifts the bottles from the crate to the bottle conveyor. The empty crates are transported to the Packer and usually pass a Crate-Washer on their way. This machine also is a cycle machine.

Washer

Dirty bottles from unpacker arrive in multiline transport at the washer. The washer is one of only two machines that process bottles in multi-line manner with constant speed and the first in the bottle flow that works in go-through style. Although this machine also possesses a multitude of process steps, the way it works is different than a cycle-machine since all steps are run simultaneously inside the machine and the bottles run through the areas of the different process steps that continuously run.

For hygienic reasons the bottles have to go through a multitude of cleaning steps and therefore stay in the washer for approximately half an hour. With production speeds of about 60000 bottles per hour one can approximate the large amount of bottles that are processed at the same time (≈ 30000). This implies the huge size of this machine.

Empty bottle inspector (EBI)

When clean bottles are transported towards the filling process, the quality of the empty bottle has to be secured. One simply does not want to fill valuable product into a cracked or dirty bottle. The quality of a bottle is inspected by the Empty-Bottle-Inspector (EBI). Since the checks have to be very accurate they are made on each single bottle. For this reason the former multi-line bottle stream is converted to single-line transport ahead the EBI. Inside the machine, a couple of images of every bottle are taken along a linear conveyor. Based on these pictures e.g. cracks at different positions of the bottle (opening, neck, body, base) or remaining contamination can be detected.

Filler

Consecutive to the EBI, without interjacent multi-line transport, the filling aggregate is placed. Here the product is filled into the bottle and the cap is added. The filler is built up as a carousel of big diameter (e.g. 7m) with around 100 to 300 filling stations that each hold one bottle. At the input, the bottles are separated and mounted to the main carousel. In a little less than one rotation of the filler, a bottle is filled (in about 10s). Afterwards, the bottle gets directly into a so called *capper head*, one station of the capper carousel, and back onto a single-lined conveyor. This leads through a closely connected inspector (usually called *Checkmat*) that inspects e.g. the filling height and rejects the bottle in case of being out of specification.

A set of machines that are connected as closely as EBI and Filler is called 'blocked'. If machines are in one block, there are no buffers in between and the machines possess direct information exchange.

Labeler

Like the Filler the Labeler is a constantly producing machine and consists of a carousel. But this time labels are glued on the bottles. After this step again a

Checkmat controls among others the correct position of all labels.

Packer

The Packer reverses the action of the Unpacker and puts the now filled, closed and labeled bottles into crates.

Palletizer

Crates containing full bottles are put on a pallet.

Auxiliary machines

In between the described main machines multiple smaller machines like an Uncapper and multiple control stations like an inspector for crates containing empty bottles may occur but are left aside in this overview since they show a low probability of errors and thus are not taken into account in terms of root cause analysis.

2.1.3 Bottle-flow control

Usually neighbouring machines do not have a direct communication. The bottle flow is controlled by so called back-up switches. In the case of bottle transport, these are sensors in multi-lane transportation segments that give information if the full width of the conveyor is used, what is equal to the information that there are buffered bottles. In the case of pallet or crate transport there are also back-up switches with a little different mechanism but the same task. For each machine, two main factors have to be given to enable it to run:

- Enough bottles in front of the machine
- Sufficient free conveyor capacity on the downstream conveyor segment

These points are ensured by using the information of the back-up switches: Before the machine there has to be at least one sensor 'active' to secure enough buffered material from upstream direction. This secures a controlled run-down in case of a stoppage of the previous machine. On the other hand, at least one back-up switch behind the machine has to be 'not active' to secure enough space for product output.

For machines that can be speed-regulated, additionally one or two more back-up switches are connected to the machine that decrease the machine speed by some percent when 'active'/'not active'.

2.1.4 Error propagation

Due to the above described interaction between machines, errors can propagate through a line. In Figure 2.5, one can see the error appearing as a downtime at the

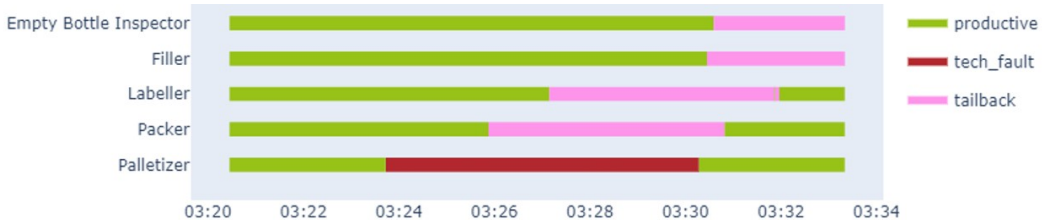


Figure 2.5 *Gantt representation of an examplary error propagation from Palletizer in upstream direction and thus also to Filler(lead-machine)*

Palletizer and propagating through the line via machines that change into machine-state 'tailback'. As this example shows, the error propagates in spatial dimension as well as in temporal dimension. The temporal sequence of error propagation is highly dependent on the occupancy of the intermediate conveyors and thus underlies strong fluctuations. Although the spatial error propagation path usually happens via the bottle flow, other paths beside the main stream like empty-crate transport are possible and occur in practice.

2.2 Line data

This section introduces different types of data that is available for most machines in a standardized way. Advantages and disadvantages of every type are discussed. Subsequently, the characteristics of machine-state data, as the data type that will be used in this work, are presented. One property is the categorical nature of machine states. Therefore, different ways of encoding categorical data into numerical style and thus being processible by Machine Learning algorithms are shown.

2.2.1 Available signals for representing line behaviour

Filling and packaging lines differ strongly from each other. On the contrary, the Syskron-services have to be designed to fit for as many lines as possible with the least possible amount of configuration work since they are going to be installed in hundreds of lines within the next few years. Thus, one of the main tasks for being able to analyze line-dynamic problems is to find a generally valid data-basis. There are some signals that are available at every machine. Within the digitalization, the output of all signals has to be standardized over thousands of Krones' machines. Thus, improving the standardization is a ongoing process.

Machine state

Filling and bottling machines provide standardized machine-state, -mode and -program according to weihenstephan standard [6]. The mentioned triple is converted into one of the categories shown in Table 2.1 which usually are (although it is inaccurate) also called machine-state. To use common wording we will also use the term machine-state for the mentioned combination of state, mode and program in the rest of this work.

machine state	description
productive	the machine produces
lack	stopped as a result of missing material at input
tailback	stopped as a result of missing space at output
own-fault	stopped as a result of an internal error
planned-downtime	stopped due to a scheduled task (changeover or cleaning)

Table 2.1 *Machine states contained in the data*

At the moment, machine-states are the most reliable signals from machines since they are also needed to compute Key Performance Indicators (KPIs) which are very important for current analysis.

Bottle counters

Knowing the current distribution of bottles in the line was the basis for the most intuitive representation for a line's dynamic in a bottling plant. If operating staff is asked what they looked for to find the error causing machine for a line downtime,

they always answer to first look at the conveyors, searching for tailback or lack. The problem is that the current distribution of bottles is currently not available in the lines' data. But by using bottle counters some information about the position of bottles in the line can be extracted. Theoretically, every machine should have an input counter and additionally a counter for good units and rejected units if there is a rejection possibility in the machine. In practice, the quality of counters is not as good as one would expect at the first view. The origin lies in the fact that digitalization is still an ongoing progress in the bottling industry. Machines are not intended to be data suppliers. Thus sensors are only installed if they are needed for the functionality of the machine. Additional sensors that would be needed to know the exact number of processed bottles are not installed due to financial reasons. A washer for example just acts like a giant conveyor. Every bottle that is inside will be cleaned but there is no information available if a certain bottle-slot is currently used. Thus processed bottles can only be estimated very roughly.

Counters hold great potential of providing information about the bottle flow in the line. Their disadvantages lie in insufficient data-quality and the missing availability over all machines.

Machine speeds

Machine speeds provide information if the machine is producing and additionally about phases of production with decreased speed that can result from conveyor circumstances, operator interventions or equipment malfunctions. On the other hand, there is no information if the stopped machine did this as a result of an own fault or induced by the connected conveyor. This information could be used as an addition to machine states but not as only input for detecting the root-cause machine.

Backup switches

Like described in subsection 2.1.3, the line is controlled by using backup switches. So their data contain secure information if a machine stop was self-induced (when no backup switch was the cause).

The main drawback for these sensors is the missing standardization, mainly regarding the information about localization of each sensor. At the moment, this would have to be done manually during data connection and thus is financially not

reasonable as standardized process.

2.2.2 Data characteristics

Machine states are used as Machine Learning data in this work. For every line downtime (stop of the lead-machine), a multivariate time-series of machine states is extracted from the data (compare Figure 2.5). In the first step, this results in an array $\mathbf{X}$ of shape $M \times T$ per example where M describes the number of machines and T the number of time-steps.

At this step, the machine states are described by strings like shown in Table 2.1. Since Machine Learning algorithms need numeric input (compare [7]), the data has to be encoded from e.g. 'productive' and 'own-fault' into numeric style.

In the following, different approaches for handling the categorical nature of data is presented and subsequently data preprocessing into different shapes for different kinds of subsequent Machine Learning algorithms is shown.

2.2.3 Encoding of categorical data

In contrast to numerical data, categorical variables represent types of data which may be divided into groups called categories. Examples are sex, nationality or, in our case, machine states. But Machine Learning algorithms need numerical data as their input. The problem is: Humans do not name their categories 1, 2, 3 but with meaningful names like 'productive' and 'equipment-failure' for the example of machine-states. These are string values and have to be converted into numerical style. In the following, different approaches for this problem are presented and the two main solutions, integer encoding and one-hot encoding are explained in detail as they will be used in this work.

Integer encoding

Each category is mapped to a natural number: Categories are ordered. Then category i is mapped to i.

$$\text{enc}_{int}(\mathbf{x}) : \mathcal{C} \rightarrow \{0, 1, ..., C\} \subset \mathbb{N} \quad (2.1)$$

e.g. the categorical variable 'age group' consists of 'junior' and 'senior':

$$\text{enc}_{int}(\text{'junior'}) = 0$$
$$\text{enc}_{int}(\text{'senior'}) = 1$$

Categorical data looks as follows: A dataset $\mathbf{X}_{cat}$ of categorical data consists of N sample-points $\mathbf{x}$. Each sample point cor-

responds to one category of a set of categories $\mathcal{C}$ with cardinality $C = |\mathcal{C}|$.

In general, there are many ways to encode categorical data and its success is highly dependent on the data. To get an overview, we use the partitioning from [8] and divide encoding mechanisms into three categories: Determined, algorithmic and automatic. Determined describes mechanisms with a low amount of computational complexity such as *integer encoding* or *one-hot encoding*.

One-hot encoding

Each category is mapped to a vector of length C containing zeros everywhere besides a single 'one'.

$$\text{enc}_{one-hot}(\mathbf{x}) : \mathcal{C} \rightarrow \{0,1\}^C \tag{2.2}$$

$$\mathbf{x}_{one-hot_j} = \delta_{j\mathbf{x}_{int}} \tag{2.3}$$

e.g. the categorical variable 'age group' consist of 'junior' and 'senior':

$$\text{enc}_{one-hot}(\text{'junior'}) = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

$$\text{enc}_{one-hot}(\text{'senior'}) = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

Their advantage is to show negligible computation time and being stable and repeatable (determined). Integer encoding (also compare box 'Intger encoding') is commonly used for 'classic' ML-algorithms like Random Forest or Support Vector Machine. In contrast Neural Networks usually need categorical input to be one-hot encoded (also compare box 'One-hot encoding').

The disadvantage comes for examples with a great amount of categories for a variable. In the field of *natural language processing* (NLP), every word of a language has to be represented by a vector. In common languages, a vocabulary includes at least some thousand words. When using one-hot encoding, this results in vectors with some thousand dimensions and due to the resulting data-sparsity, the processing costs are disproportionately high [9].

Algorithmic techniques encode categorical variables into a better condensed space than determined approaches by using better elaborated algorithms. In contrast to automatic encodings, they do not use the training data and e.g. are not directly included in neural network training.

Accordingly automatic techniques use training data and adapt the encoding directly to it. This approach is widely used in recent research. Especially *embedding* mechanisms set up new standards. These are represented by neural network layers that are fed with data in one-hot encoding (e.g. [10]). Their advantage lies in the ability to bring context into data representation like minimal distances between similar words in NLP [11]. Thus the used encoding process is a combination of one-hot encoding and embedding while the embedding part is learned during the

neural network training. Combining multiple techniques is in general a very widely used strategy [8].

2.2.4 Representing a filling line

For very simple filling and packaging lines, a simple list of machines in the order they are passed e.g. by the bottle is sufficient. But usually the architecture of a line is way more complex and consists of auxiliary flows like for empty pallets or crates. The most precise mapping of the physical topology of a filling and packaging line is by using a graph structure. An examplary graph is given in Figure 2.6 which displays the graph corresponding to the typical filling line that can be found at a brewery (compare subsection 2.1.2).

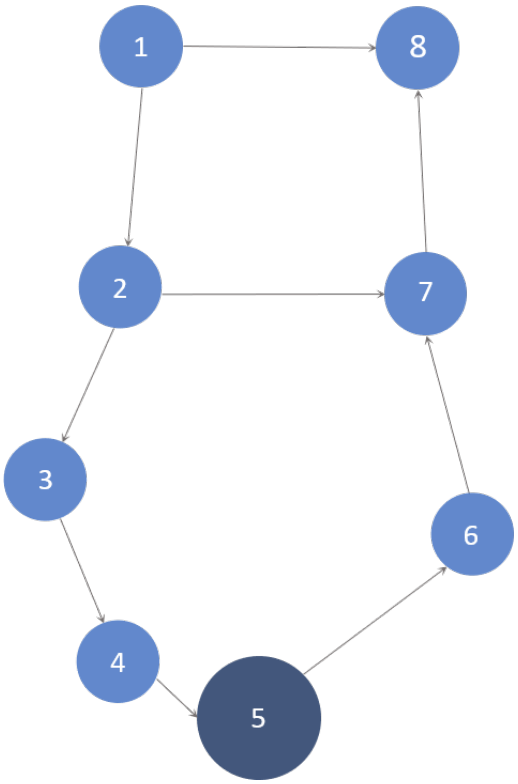


Figure 2.6 *Graph for an exemplary line consisting of machines like described in 2.1.2. 1: Depalletizer, 2: Unpacker, 3: Washer, 4: Empty Bottle Inspektor, 5: Filler, 6: Labeler, 7: Packer, 8:Palletizer. Connections between machines 1 and 8 as well as 2 and 7 describe auxiliary flows while the other connections correspond to the main stream.*

A graph G consists of vertices $\mathbf{V}$ and edges $\mathbf{E}$. In the case of a bottling and packaging plant, the set of machines $\mathbf{M} = \{m_1, ..., m_M\}$ act as vertices ($\mathbf{V} = \mathbf{M}$) and the conveyors as their connections act as edges (e.g. $\mathbf{E} = \{\{m_1, m_2\}, \{m_1, m_8\}, \{m_2, m_3\}, \{m_3, m_4\}, ...\}$). In the following, the terms vertex and node will be used sysnonymously.

To mathematically describe a graph structure, usually an adjacency matrix

$$\mathbf{A}_{ij} = \begin{cases} 1 & \text{if } m_i, m_j \text{ connected} \\ 0 & \text{otherwise} \end{cases} \tag{2.4}$$

with $\mathbf{A}_{ij}$ describing the connection between node i and node j, $\mathbf{A}_{ij} = 1$ indicating a connection and $\mathbf{A}_{ij} = 0$ unconnected machines, is used.

The mentioned approach is valid for undirected graphs. But the material flow in a filling and packaging line is directed and thus we also need to cover this information in the adjacency matrix:

$\mathbf{A}$ is extended to a $M \times M$ matrix containing $\mathbf{A}_{ij} = 1$ if the edge shows the direction of the bottle flow and $\mathbf{A}_{ij} = -1$ if units are transported from machine j to machine i.

$$\mathbf{A}_{ij} = \begin{cases} 1 & \text{if material flows from } m_i \text{ to } m_j \\ -1 & \text{if material flows from } m_j \text{ to } m_i \\ 0 & \text{otherwise} \end{cases} \tag{2.5}$$

For the given example, this results in the adjacency matrix given in Table 2.2.

2.3 Time-series classification (TSC)

Time-series data is one of the main types of data for Machine Learning since all data that shows some kind of ordering can be cast as a TSC-problem. Accordingly, the variety of problems in TSC is large and include translation, text sentiment analysis, cyber-security and medical assessment as well as many more [12]. Since classification in general is a kind of supervised learning, in time-series classification one example in the dataset is represented by a feature-label pair $\{\boldsymbol{x}, y\}$ with $\boldsymbol{x} = (x_1, ..., x_T)$ consisting of T time-steps and the class label $y \in \mathcal{C}$. Here $\mathcal{C} = [0, 1]^C$ represents

Adjacency Matrix	Depalletizer	Unpacker	Washer	Empty Bottle Inspektor	Filler	Labeler	Packer	Palletizer
Depalletizer	0	1	0	0	0	0	0	1
Unpacker	-1	0	1	0	0	0	1	0
Washer	0	-1	0	1	0	0	0	0
Empty Bottle Inspektor	0	0	-1	0	1	0	0	0
Filler	0	0	0	-1	0	1	0	0
Labeler	0	0	0	0	-1	0	1	0
Packer	0	-1	0	0	0	-1	0	1
Palletizer	-1	0	0	0	0	0	-1	0

Table 2.2 *Exemplary adjacency matrix for the graph given in Figure 2.6. Each cell describes the connection from machine A on the left(y-axis) towards machine B, given at the top(x-axis). If material flows from A to B the entry is one, for the opposite direction minus one is inserted and finally, a zero represents the case that no connection is given.*

the space of class probabilities consisting of C different classes. The classifier is a function

$$f_{clf} : \mathcal{X} \to [0,1]^C \tag{2.6}$$

from the space of possible input values $\mathcal{X}$ to the space of class membership probabilities. The goal of model training is to optimize the model, that should be used as f_{clf} to predict $\tilde{\boldsymbol{y}}$ in the best possible way:

$$f_{clf}(\boldsymbol{x}) \approx \tilde{\boldsymbol{y}} \tag{2.7}$$

Since $\tilde{\boldsymbol{y}}$ represents class probabilities $\sum_{c \in \mathcal{C}} \tilde{y}_c = 1$ has to hold. To get one concrete prediction the class with the maximum predicted probability is chosen.

Since we work with multivariate time-series, every $\boldsymbol{x}_t$ consists of multiple dimen-

sions D s.t. $\boldsymbol{x}_t \in \mathbb{R}^D$. Thus, in contrast to univariate time-series classification, patterns have to be found not only in the temporal dimension of the data, but also in the dependencies between D features of each time-step.

Since the problem is known

"as one of the most challenging problems in data mining"[12]

it took Deep Learning until 2015 to take root in this area. One reason for this is the "strong baseline" laid by a combination of Dynamic Time Warping (DTW) distance in combination with a Nearest Neighbour classifier (kNN)[13], [14]. This also is the reason to use Dynamic Time Warping and kNN as the baseline model in this thesis. A further solution approach for multivariate time-series classification (MTSC) is to develop a feature extraction (based on expert knowledge) and use a standard classifier on these features. Due to the decreased complexity of the problem, classifiers like Random Forest, Support Vector Machine and similar can be used. Like all fields of Machine Learning, also in TSC nowadays usually Deep Neural Networks like Convolutional Neural Networks are used [15]. Due to the graph structure of each filling line, Graph Convolutional Neural Networks will be used in this work.

In the following, after introducing basic Machine Learning nomenclature, the algorithmic approaches for

- Dynamic Time Warping
- manual feature extraction combined with Random Forest
- Graph Neural Networks

will be introduced.

2.3.1 Machine Learning nomenclature

In this work, the terms *example* as well as *sample-point* are used for describing the input of the classifier $\boldsymbol{x}$ corresponding to one lead-machine stop. A classifier consists of a Machine Learning model like DTW+kNN, Random Forest or Graph Neural Network, or alternatively a rule-based algorithm. The classifier *predicts* or *classifies* and thus produces its output, the class-probabilities, which is compared (since classification is a Supervised Learning approach) to the *ground-truth* or *label*, that was provided or *labeled* by an expert. Thus, our datasets always have to consist of example-label pairs.

Before training a model, the available data is split into training- and test-set. For Neural Network training often also a validation-set is extracted from the data. When talking about model training, the above mentioned optimization of the model to fulfill Equation 2.7 is meant and conducted using the training-set. In the end, we are always interested in the model's *generalization ability*. This means the ability of the model to correctly predict unknown input which was not part of the training data. Thus, we test the model's performance using a test-set that consists of examples that are unknown for the model. Additionally, especially during Neural Network training, a validation-set is used to estimate the generalization ability during training, after each training epoch. This is useful in cases when the model starts to overfit during the training process. We talk about *overfitting* when the training accuracy of the model rises but simultaneously the validation accuracy decreases. In this phase of training the model learns patterns inside the training data that are too specific and thus it's generalization ability decreases although it still improves it's performance on the training set. Thus, often the model parametrization which showed the highest validation accuracy during training, is assumed to have the best generalization ability.

2.3.2 Dynamic Time Warping and k-Nearest Neighbours

Dynamic Time Warping and k-Nearest Neighbours (DTW+k-NN) is a simple but in general powerful approach (compare [14]) for Time-Series Classification which consists of two main parts:

Dynamic Time Warping (DTW) is an algorithm that computes a similarity measure between single examples of data and k-Nearest Neighbours (k-NN) is the classifier that uses this similarity measure.

DTW, as many other time-series specific algorithms, has its origin in speech recognition but was early adapted to other inherently temporal problems [16]. The original problem was about recognizing words, although they provide different timing (speed of each syllable or sound in pronunciation) and accents, what results in non-linearly expanded or contracted signals.

Thus the similarity between the mentioned signals can not be calculated by common similarity measures like euclidean distance (compare Figure 2.7) which are not able to temporally warp the signals while comparing them. Therefore, DTW was introduced providing this special ability which also includes the capability of comparing time-

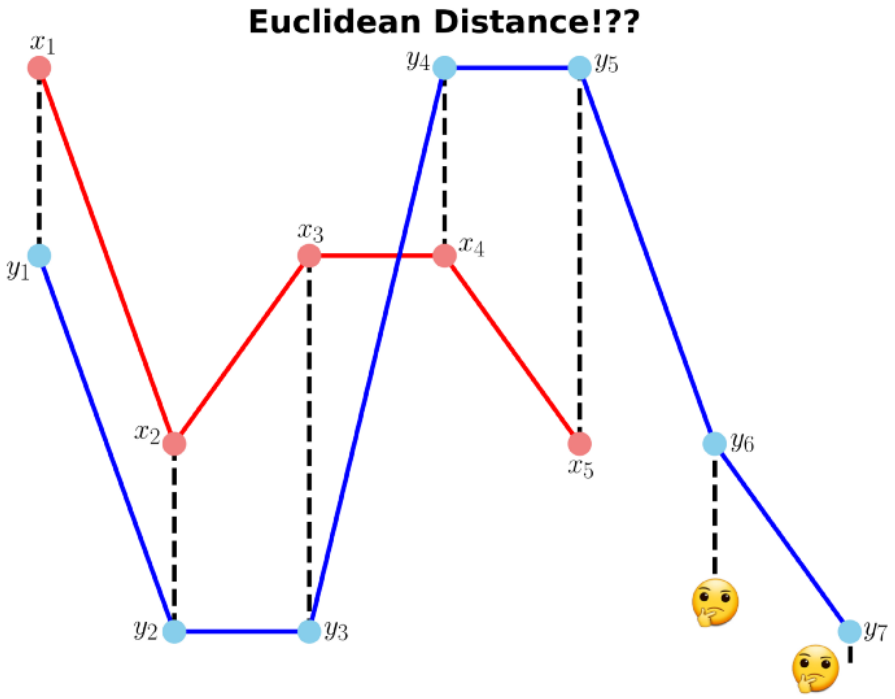


Figure 2.7 *Two exemplary time-series (blue and red). The black lines show which points would be compared by a basic similarity measure. Note that this is not possible for time-series of different lengths. (image similar to [17])*

series of different lengths (compare Figure 2.8).

Taking a closer look at k-NN, the idea of the algorithm can be described like the following: The simplest idea for classification is to compare the new example of an unknown class with the sample-points that have a class label (training set) and use the label of the most similar stored instance as the new prediction. That is the basic idea behind a Nearest Neighbor classifier. The drawback of looking at only the most similar training-example is a severe probability of falling for a misleading one. Thus, usually a number k of most similar stored instances is taken into account. This leads to a better generalization of the classifier and is called k-Nearest-Neighbours. In this case the prediction is extracted e.g. by majority voting beyond the k "nearest" labels.

But how do these two elements collaborate? We assume a dataset with labeled examples. When a new example arrives, it should be classified. This is done by comparing the new example to all known (labeled) examples by computing the DTW-similarity. Subsequently, the k most similar examples are known and their

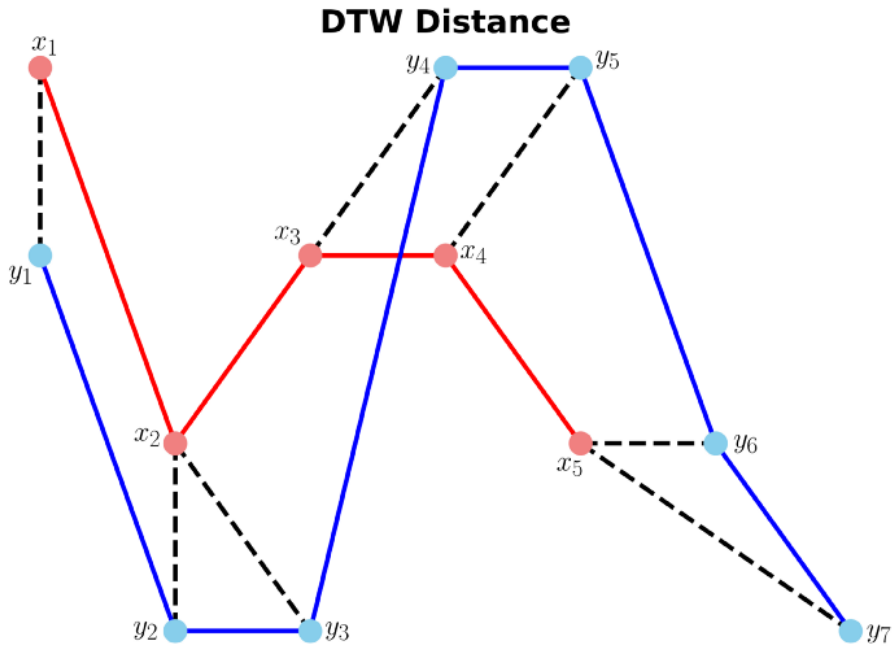


Figure 2.8 *Illustration of the assignment of points between two time series by dynamic time warping. The black lines show which points are be compared. Due to this assignment, similarity between time series with similar shape but e.g. delay or warping of patterns can be determined. Note that even time-sequences with different lengths can be compared. (image similar to [17])*

corresponding labels are extracted from the dataset. Finally, the prediction is selected by a majority voting beyond the extracted labels. In other words: If three of the five most similar examples carry the label 'Filler', the Filler is predicted as causer-machine.

For a detailed insight into dynamic time warping, we look at an example and compare two time-sequences $\mathbf{x} = [x_1, x_2, ..., x_m]$ and $\mathbf{y} = [y_1, y_2, ..., y_n]$. The distance between x_i and y_j is written as d(i, j) where $1 \leq i \leq m$ and $1 \leq j \leq n$. The algorithm fills a $m \times n$-matrix $\mathbf{D}$ by using the recursive formula

$$\mathbf{D}(i,j) = d(i,j) + \min \begin{cases} D(i, j-1) \\ D(i-1, j) \\ D(i-1, j-1) \end{cases} \tag{2.8}$$

and the initial condition $D(1,1) = d(1,1)$.

After finishing $\mathbf{D}$, the best warping is extracted by starting at (m, n) and from every position (i, j) progressing to one of $(i-1, j)$, $(i, j-1)$ or $(i-1, j-1)$ depending on which corresponding value of $\mathbf{D}$ is the lowest. In this way the ideal path through the cost-matrix $\mathbf{D}$ is found and provides the ideal warping from x to y.

To further improve the performance of DTW, constraints regarding the path like a maximum difference between i and j can be given:

> *"As we can see, relatively tight warping constraints produce more accurate classifiers."*[18]

One big advantage of this approach is the ability to process data with different lengths in time-dimension especially since Machine Learning algorithms often are dependent on equally sized input.

As a major drawback, contrary to the usually hard to beat performance, the combination of k-NN and DTW is computationally costly while classifying and therefore not suitable for live computation or other fields that need fast inference. This is based on the fact that Dynamic Time Warping itself is (compare [19]) computationally expensive although already heavily optimized and has to be calculated for every combination of sample-point that should be classified, and every example in the training set.

2.3.3 Manual feature extraction

If there is the possibility to extract a number of features from the multivariate time-series that preserve most important information for the classification, a time-series problem can be converted to a normal Machine Learning problem with one-dimensional data as input. Thus the model no longer has to learn a function $\mathbb{R}^{N \times T \times D} \to \mathcal{C}$ but the way less complex function $\mathbb{R}^{F} \to \mathcal{C}$, where N denotes the number of sample-points, T the number of time-steps, D the number of dimensions of the time-series and F the number of features. The resulting problem can be solved with every kind of classifier that has a 1d-input per example. This includes all "classic" classifiers like Random Forest, Support Vector Machine and also Multilayer Perceptrons.

2.3.4 Random Forest classifier

Random Forests are built up using an ensemble of Decision Trees that take 1d-input like e.g. features from previous chapter as input. The output of the Random Forest is usually given by a majority voting over all Trees.

Each Tree consists of a hierarchical structure of nodes and edges, each node possesses two children. The final nodes are called leaves. Training a Decision Tree means iteratively splitting the training-data into two branches, the children. The goal of training is to set up a decision for each node such that(s.t.) each leave consists of training samples of the same class. This is called purity and usually *gini index* G (Equation 2.9)

$$G = \sum_{i=1}^{C} p_i(1 - p_i) \tag{2.9}$$

is used as purity measure where p_i describes the ratio of class i members in the current split.

But Decision Trees are known for one drawback: They are prone to overfitting. Besides of using ensembles of Decision Trees to increase generalizability, *bagging* is used. The original idea [20] was randomly selecting subsets of the dataset and training each tree on only a subset of the data. This idea evolved to *feature bagging*. This means that a random subset of features is selected for each split. In today's implementations of Random Forest classifiers, the usage of the introduced ideas can be selected when initializing the classifier.

2.3.5 Neural Network approaches

Over the last decades of improvements in Neural Networks research, types of Neural Network Layers heavily evolved. After a long time of research using Multilayer-Perceptrons, also called Feedforward Artificial Neural Network (consisting of Fully Connected Layers, compare Figure 2.9a), the introduction of Convolutions for Neural Networks [21] marked a breakthrough for e.g. processing images (compare Figure 2.9b). Their great advantages are e.g., the sharing of weights and thus their ability to overcome the "curse of dimensionality", the recognition of patterns independently to local shifts in the input, and processing of multidimensional input, like images, under preservation of the original two-dimensional structure. But Convo-

lutional Neural Networks need input that is arranged on a lattice which implies a constant number of neighbors. In contrast, many structures in nature belong to non-Euclidean domains and can be represented as graphs ([22], [23], [24]). Therefore, Graph Neural Networks [23] (compare Figure 2.9c) and many kinds of Graph Convolutional Neural Networks (e.g. [25], [26]) were invented.

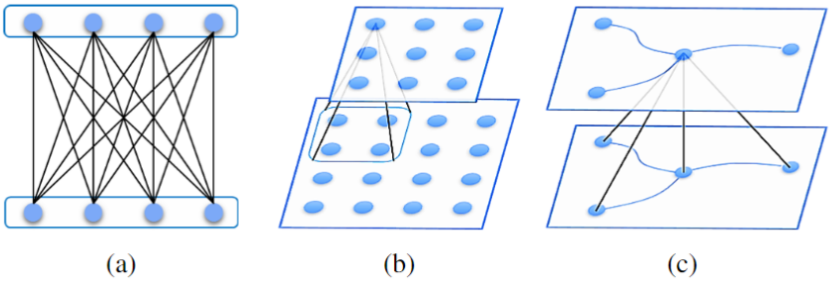


Figure 2.9 *Three different types of layers. (a) fully connected layer. (b) convolutional layer (2-dimensional). (c) graph convolutional layer. [27]*

In this work, Graph Convolutional Neural Networks will be used to classify (compare Equation 2.7) the root cause machine in a filling and packaging line.

Additionally, we want to cover requirements that originate from Multi-Task Learning and Transfer Learning. Therefore, instead of the common architecture that combines Convolutional Layers for feature extraction and fully connected *decision-layers*, that output class probabilities, we use Fully Convolutional architectures, where class probabilities are output of the last Convolution Layer without additional fully-connected layers on top. By using Graph Convolution layers, we can build up a network that automatically adapts its output-size, the size of the class probability vector, to the number of machines that appear in the input data.

Thus, one model can solve the root cause classification task for all available bottling lines!

In this chapter Convolutional Layers and afterwards Graph Convolutional Layers, in particular Relational Graph Convolutional Layers, will be introduced. Additionally, a combination of Convolutional Layers and Graph Convolutional Layers, called spatio-temporal Graph Convolutions that use 1D Convolutional Layers for temporal and Graph Convolutional Layers for spatial pattern recognition, are explained.

This theory is later used to build up two types of Graph Convolution Networks: On the one hand we built up a Relational Graph Convolution Neural Network that takes

manually preprocessed data (time-dimension is collapsed to size 1 by preprocessing) and does spatial analysis to identify the root-cause machines. On the other hand we use spatio-temporal Graph Convolutions to process the full multivariate time-series data by performing temporal convolutions as well as spatial convolutions.

Both models will be built up according to the mentioned fully convolutional architecture to meet the requirements given by Multi-Task Learning and Transfer Learning (section 3.10).

Learning process of Neural Network for classification

A Neural Network consists of layers that apply some linear function $\ell(\theta, \mathbf{x})$ that is dependent on the model parameters θ, on the input $\mathbf{x}$ and add a nonlinear activation function σ on the result. By stacking e.g. L layers, a neural network $f = \ell_1 \circ \ell_2 \circ ... \circ \ell_L(\theta, \mathbf{x})$ arises.

In this work, we use the activation functions $\tanh(x)$, $\text{sigmoid}(x) = (1 + e^{-x})^{-1}$, $\text{ReLU}(x) = \max(0, x)$ and $\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_{c=1}^{C} e^{x_c}}$. Since softmax function is used for converting activations into class probabilities, C is the number of output nodes and simultaneously classes.

To train a Neural Network f, a loss function $\mathcal{L}$ has to be defined to use it as input for an Optimizer that adapts the networks parameters by using backpropagation. For deeper details into this sections topics the reader is referred to e.g. [28].

When using Neural Networks for classification tasks usually categorical cross-entropy

$$\mathcal{L}(\mathbf{x}, \mathbf{y}) = \mathbf{y} \log \hat{\mathbf{y}} = \sum_{c=1}^{C} y_c \log \hat{y}_c \tag{2.10}$$

is used as loss function. Here $\mathbf{x}$ denotes one example of the dataset, $\mathbf{y}$ the corresponding label in one-hot notation and $\hat{\mathbf{y}}$ the model's output which is a vector of class probabilities. Finally, C represents the number of classes.

To smooth the trajectory of net parameters in their parameter space while optimizing them, the examples of the dataset are divided into batches with size B. Each batch consists of a set of examples $\mathbf{X} = \mathbf{x}_1, ..., \mathbf{x}_B$ and a set of corresponding labels

$\mathbf{Y} = \mathbf{y}_1, ..., \mathbf{y}_B$. For each batch the loss

$$\mathcal{L}(\mathbf{X}, \mathbf{Y}) = \sum_{b=1}^{B} \sum_{c=1}^{C} y_{b,c} \log \hat{y}_{b,c} \tag{2.11}$$

and corresponding backpropagation with parameter adaptions are computed.

Convolutional Layer

In this work, we mainly use convolutional layers in our Neural Networks. To introduce these, we start with the standard definition for a discrete convolution

$$(\mathbf{x} \star \mathbf{f})(t) = \sum_{s=0}^{K-1} \mathbf{f}(s)\mathbf{x}(t-s) \tag{2.12}$$

where $\mathbf{f} \in \mathbb{R}^K$ denotes the convolution filter, $\star$ the convolution operation and $\mathbf{x} \in \mathbb{R}^T$ a 1d series of data with length T.

In Machine Learning, the filter consists of learnable parameters (weights), thus written as $\mathbf{W}^{(l)}$ for layer l (compare Equation 2.13), which are adapted by backpropagation. These learned filters are responsible for detecting the corresponding learned patterns in the input $h^{(0)}$ or hidden dimension $h^{(l)}$. In this case the convolution operation between weights and input nodes is represented by the term $\sum_j h_j^{(l)} W_{ij}^{(l)}$. From now on, we use superscript (l) notation for the corresponding layer. Thus $h_i^{(l)}$ denotes the i-th neuron of layer l.

As usual in Neural Network implementation (compare [29]), a bias b is added to the convolutions result. To enable the network to learn non-linear dependencies, an activation function σ is applied to the sum of convolution result and bias.

$$h_i^{(l+1)} = \sigma\left(\sum_{j \in \mathbf{M}_i} W_{ij}^{(l)} h_j^{(l)} + b^{(l)}\right) \tag{2.13}$$

Additionally, the summation is conducted over the indices j, provided by $\mathbf{M}_i$, which thus is adaptable to different selections from input h. In other words, one can choose which nodes' information (from layer l) should be respected when computing $h_i^{(l+1)}$. An example in which this is needed, is dilatation. This concept is introduced in the following section. To introduce another common notation for layers, we rewrite

Equation 2.13 to

$$\mathbf{H}^{(l+1)} = \sigma(\mathbf{W}^{(l)} \star \mathbf{H}^{(l)} + \mathbf{b}^{(l)}). \tag{2.14}$$

Equations 2.13 and 2.14 intend the same calculation specification.

Temporal Convolutional Layer

We adopt the gated temporal convolutional layer according to Wu et al. [30]. Therefore Equation 2.12 is extended to a dilated convolution operation

$$\mathbf{x} \star \mathbf{f}(t) = \sum_{s=0}^{K-1} \mathbf{f}(s)\mathbf{x}(t - d \times s) \tag{2.15}$$

by using d as dilation factor. This results in the kernel only respecting every d-th step in $\mathbf{x}$. In this way the layers' receptive field grows linearly by increasing d. When increasing the dilatation factor from top layer towards the input like done in [30], the receptive field of the network grows exponentially with layer depth what enables effective handling of long-range dependencies in data.

Additionally, gating is introduced by using

$$\mathbf{H}^{(l+1)} = g(\hat{\mathbf{W}}^{(l)} \star \mathbf{H}^{(l)} + \hat{\mathbf{b}}) \odot \sigma(\mathbf{W}^{(l)} \star \mathbf{H}^{(l)} + \mathbf{b}) \tag{2.16}$$

where $\mathbf{W}$, $\hat{\mathbf{W}}$, $\mathbf{b}$ and $\hat{\mathbf{b}}$ are model parameters, and $\odot$ is the element-wise product. Furthermore $\sigma(\cdot)$ is an activation function of the outputs, usually a hyperbolic tangent, while $g(\cdot)$ is the sigmoid function which determines the ratio of information passed to the next layer and thus represents the gating functionality (compare [30], [31]). In this approach, the gating mechanism enables the model to decide if a computed feature should pass into the next layer, based on the input that was also used to compute the feature in question.

Graph Convolutional Layers

Convolution operation on graphs follows the same idea like 'classic' CNNs (like described in section 2.3.5) but mathematical formulation has to be adapted to the number of neighbors per node being dynamic, since this changes the number of sources which provide information for a given node.

Note that M_i in Equation 2.13 consists of the same number of elements for each convolution operation. More intuitively spoken, the area of data that is processed by a kernel will always have the same shape. In contrast, graph data, as it is not distributed evenly on a grid like e.g. images, consist of a variable number of neighbors for each node (compare Figure 2.9). But information of each neighbor should be respected during the convolution operation on the current node.

Thus M_i from Equation 2.13 is replaced by the neighborhood of node i, denoted as $\mathcal{N}_i$. Furthermore, for indistinguishable edges there is no reasonable way to associate different weights to them and thus the same weight W is used to propagate information from all neighbors. In addition to the neighbors' information, also information of the current node is respected by adding a so-called self-loop with weight W_0.

Introducing the described changes results in Equation 2.17.

$$h_i^{(l+1)} = \sigma\left(\sum_{j \in \mathcal{N}_i} W^{(l)} h_j^{(l)} + W_0^{(l)} h_i^{(l)}\right) \tag{2.17}$$

For efficient implementation, Equation 2.17 is translated to Equation 2.18

$$\mathbf{H}^{(l+1)} = \sigma\left(\tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}} \mathbf{H}^{(l)} \mathbf{W}^{(l)}\right) \tag{2.18}$$

with $\mathbf{H}^{(l)}$ denoting the input, $\mathbf{W}^{(l)}$ the weight matrix and the term $\tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}}$ representing the dependencies between nodes. This term is combined from $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}_N$ with $\mathbf{I}_N$ as $N \times N$ identity matrix and $\mathbf{A}$ denoting the adjacency matrix, and the diagonal matrix $D_{ii} = \sum_j \tilde{\mathbf{A}}_{ij}$ [25]. In this notation $\tilde{\mathbf{A}}$ represents the adjacency matrix of the undirected graph with added self-connections.

For the case that edges can be distinguished into different kinds (in contrast to Equation 2.17), which means that now there are different relations between the nodes, a weight can be learned for each kind of relation. For this case Schlichtkrull et al. [26] introduce Equation 2.19 which extends Equation 2.17 by adding a summation over the relations $r \in \mathcal{R}$ and an additional learnable weight $c_{i,r}$ that evaluates the influence of each relation for the given node.

$$h_i^{(l+1)} = \sigma\left(\sum_{r\in\mathcal{R}}\sum_{j\in\mathcal{N}_i^r}\frac{1}{c_{i,r}}W_r^{(l)}h_j^{(l)} + W_0^{(l)}h_i^{(l)}\right) \tag{2.19}$$

Examples for different kinds of relations between nodes of a knowledge graph are e.g. a person 'is citizen of' some country, 'educated at' some university or 'awarded' some prize [26]. In this thesis we distinguish between error propagation by 'tailback' and by 'lack', respectively upstream and downstream propagation of the error.

Transferring the multiple kinds of relations into Equation 2.18 results in

$$\mathbf{H}^{(l+1)} = \sigma\left(\sum_{r\in\mathcal{R}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}_r\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}^{(l)}\mathbf{W}_r^{(l)}\right) \tag{2.20}$$

with $\mathcal{R} = \{\text{'upstream'}, \text{'downstream'}, \text{'self-loop'}\}$. Thus each graph convolution layer is a combination of three graph convolutions with a separate adjacency matrix. $\mathbf{A}_{downstream}$ describes a directed graph equal to the directed graph of material flow through the line. $\mathbf{A}_{upstream}$ accordingly corresponds to the same nodes but edges point in the opposite direction. Finally $\mathbf{A}_{self-loop}$ is the identity matrix $\mathbf{I}_N$ and describes connections of each node to itself.

Blocks of Layers

In this work, we adopt the combination of spatial and temporal convolutions like proposed in [30]. Their approach is combining a gated TCN and a GCN with residual connection to one layer.

In the next step the mentioned layers are combined to blocks like shown in Figure 2.10. Each block consists of K layers. The outputs of all layers are concatenated and two convolutional layers with ReLU activations are applied before the block outputs the information. The last mentioned layers are depicted as 'Linear' which is an easily misleading nomenclature. Thus we go a little deeper into detail at this point: Since concatenation of layer outputs is done in the dimension of channels, a new number of channels, which is the product from number of layers and number of channels before concatenation, exists. To reduce this amount of channels two convolutions with 1x1-kernels are applied on the data. Effectively these 1x1 convolutions act as if a fully connected (linear) layer was applied on the channels of one node at

one timestep. Thus the naming 'Linear' occurs.

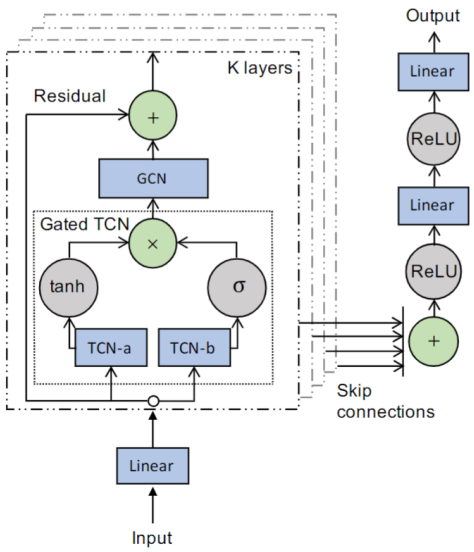


Figure 2.10 *Visualization of one block of layers used in spatio-temporal GCN [30]. "Linear" layers describe a fully connected layer without activation function, the grey circles containing "tanh", "σ" or "ReLU" represent activation functions while the green circle containing "+" depicts a concatenation operation and the green circle containing "×" describes the element-wise product.*

Usually layers are stacked inside a Neural Network. In this case a block of layers takes the function that usually one layer fulfills. That means a number of blocks will be stacked and after the final block some *decision layer* that converts the information into class probabilities will be added to create a Neural Network for solving classification tasks.

2.4 Weak Supervision

Machine Learning has achieved great results in solving various problems over the last decades. Typically, a Deep Neural Network is trained on a huge amount of training examples, each consisting of a tensor $\boldsymbol{x}$ containing the features, and the corresponding ground truth label y. In classification, the label represents the class in which the example belongs and the model f should learn the mapping $y \approx f(\boldsymbol{x})$ like described in Equation 2.7.

Gathering a huge amount of feature representations is usually an automatized task and thus does not need too much manpower. But creating a ground-truth label for every example is a time-consuming and difficult job. Thus, it can be difficult to attain strong supervision information for a Machine Learning problem with *Strong* Supervision meaning that there is a correct (ground-truth) label, representing exactly the desired model output, available for every training example.

Due to usually high labeling costs for strong supervision, especially in an industrial context perfectly supervised datasets are seldomly available. But the corresponding use-cases should be solved anyways. Thus, all data that carries some valuable information has to be used. This field is, in contrast to strong supervision, called *Weak Supervision*. A definition is:

> *"We define WSRL [weakly-supervised representation learning] as the collection of representation learning problem settings and algorithms that share the same goals as supervised representation learning but can only access to less supervised information than supervised representation learning"*[32]

But how does weakly supervised data look and how can it be of use for Machine Learning?

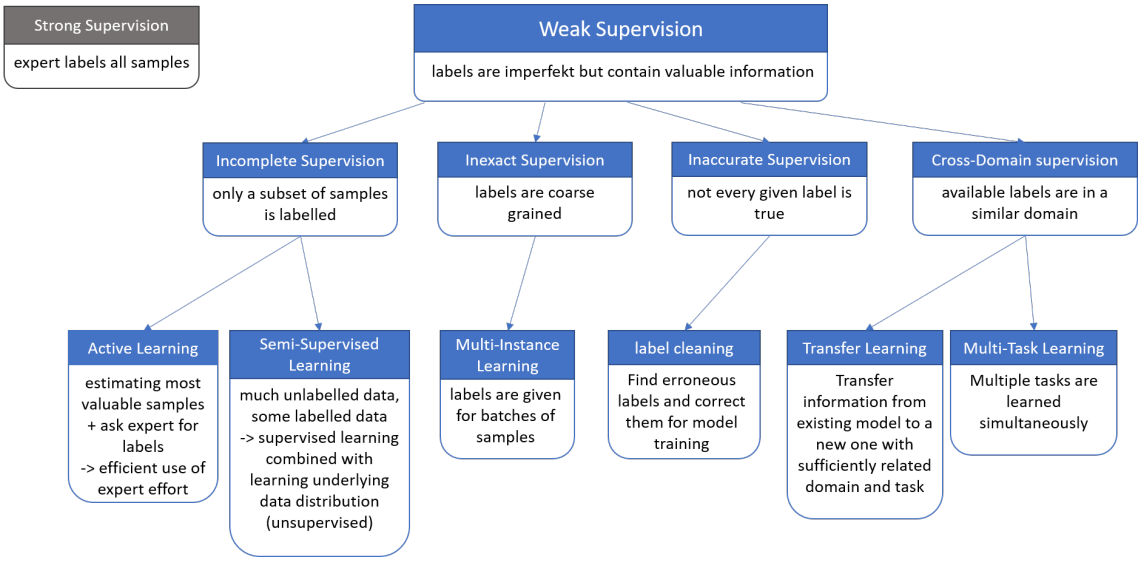


Figure 2.11 *Strong vs. Weak Supervision with subcategories of Weak Supervision and approaches for Neural Network training with data corresponding to one subcategory.*

In the following, an overview over the four main types of Weak Supervision will be given and approaches to utilize the corresponding data are provided [33]. Also, a

graphical overview provided in Figure 2.11.

Incomplete Supervision describes the subcategory when only a subset of training examples is labeled. Formally the model $f\colon \mathcal{X} \to \mathcal{Y}$ has to be learned from a training set $\mathbf{X} = \{(\boldsymbol{x}_1, y_1), ..., (\boldsymbol{x}_L, y_L), \boldsymbol{x}_{L+1}, ..., \boldsymbol{x}_N\}$ with L (strongly) labeled sample-points and $U = N - L$ unlabeled examples. $\mathcal{Y}$ denotes the space of predictions of the model. At this point, two common techniques are used: On the one hand *Active Learning* can be used to get ground-truth labels for the most informative unlabeled examples. Afterwards, Supervised Learning is conducted on the built up labeled data. On the other hand, *Semi-Supervised Learning* can be used to learn from labeled and unlabeled data simultaneously. Thus a mix of supervised and unsupervised learning has to be used. In this case, in addition to learning the classification, the unlabeled data is used to better adapt the model to the underlying data distribution and thus improve generalization of the model.

Inexact Supervision is the field, in which labels are not as detailed as needed for the problem task but coarsely-grained. Formally the model $f\colon \mathcal{X} \to \mathcal{Y}$ has to be learned from a training set $\mathbf{X} = \{(\boldsymbol{x}_1, \tilde{y_1}), ..., (\boldsymbol{x}_N, \tilde{y_N})\}$ with $\tilde{y_i} \in \tilde{\mathcal{Y}}$ and $\tilde{\mathcal{Y}}$ an inexact label space compared to $\mathcal{Y}$. For example learning object localization from a dataset that only has class labels without a local information. This is e.g. done by using *Multiple Instance Learning* (MIL).

Inaccurate Supervision describes the setting that every sample-point has a label describing the desired model-output but the labels are not always ground-truth. Again $f\colon \mathcal{X} \to \mathcal{Y}$ has to be learned from a training set $\mathbf{X} = \{(\boldsymbol{x}_1, y_1), ..., (\boldsymbol{x}_N, y_N)\}$ but a percentage of ys is erroneous. To solve this e.g. label cleaning can be used where erroneous labels are detected and corrected.

Cross Domain Supervision describes the scenario that there are no or only few labels in the original domain but supervision information can be sufficiently derived from a related domain by using e.g. Multi-Task Learning or Transfer Learning.

In practice, the presented types of Weak Supervision often appear in a mixed style. This work faces a combination of incomplete, inaccurate and cross-domain supervision. In the following, solutions for these types are presented. Firstly, Active

Learning will be discussed as a tool to extract best possible information, given a defined amount of labeling-effort by an expert. As additional label-source, algorithmically created inaccurate labels in combination with a label-cleaning approach is presented. To combine classification knowledge over multiple plants, Multi-Task Learning and Transfer Learning for the field of Cross Domain Supervision is introduced.

2.4.1 Active Learning

While in many problems data that can be used as features for a Machine Learning problem is available in sufficient amount, the corresponding labels are usually created by an expert and therefore costly.

> *"How do we select instances from the underlying data to label, so as to achieve the most effective training for a given level of effort?"*[34]

The goal of *Active Learning* is to exploit a labeling expert's (called oracle) work as much as possible by requesting (querying) the labels for the most promising sample-points. This can be either done for an existing set of examples (pool-based Active Learning) or for every new incoming example (online Active Learning). Since the further course of this work will use the mechanisms of pool-based Active Learning, these will be elaborated in the following.

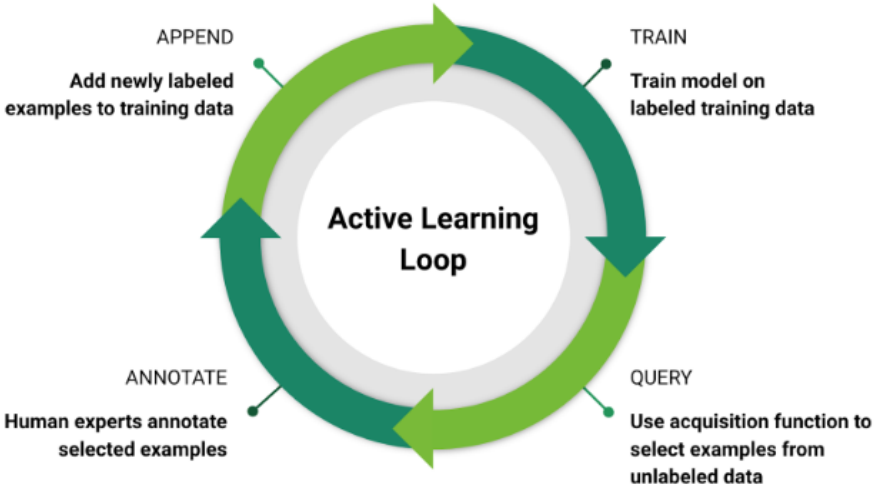


Figure 2.12 *Loop describing one iteration of the active learning process [35]*

The workflow includes the following steps (compare Figure 2.12): From a set of

unlabeled data $\mathbf{U}$ the most promising examples $\tilde{\boldsymbol{x}}$ are extracted by the *query strategy* $\mathcal{Q}(f, \mathbf{U}, \mathbf{L})$. It usually also uses the set of labeled examples $\mathbf{L}$ and the model f (e.g. when using the uncertainty of the model) as input.

Under the hood, an utility score representing how promising the gained information is, is calculated for every unlabeled example. According to the score a number N of sample-points $\{\mathbf{x}_1, ..., \mathbf{x}_N\} = \tilde{\mathbf{U}}$ is extracted and sent to the oracle as a query. The oracle returns the sample-points with corresponding labels $\tilde{y} = \mathcal{O}(\tilde{\boldsymbol{x}})$ such that new labeled pairs $\{(\tilde{\boldsymbol{x}}_1, \tilde{y}_1), ..., \tilde{\boldsymbol{x}_N}, \tilde{y}_N)\} = \tilde{\mathbf{L}}$ are created and thus expands the set of labeled examples $\hat{\mathbf{L}} = \mathbf{L} \cup \tilde{\mathbf{L}}$. Now the model is retrained and the loop starts again until the set of unlabeled examples is empty or e.g. the loss of the model is below a threshold δ. For a pseudo-code description of pool-based Active Learning also compare algorithm 1.

Algorithm 1 Pool-based Active Learning

Input: Classifier f trained on $\mathbf{L}$, unlabeled examples $\mathbf{U}$, labeled examples $\mathbf{L}$
Output: f with $\mathcal{L}(f) < \delta$
while $\mathbf{U} \neq \{\}$ & $\mathcal{L}(f) > \delta$ **do**
  $\boldsymbol{x}_s = \mathcal{Q}(f, \mathbf{U}, \mathbf{L})$
  $y_s = \mathcal{O}(\boldsymbol{x}_s)$
  $\mathbf{L}$, $\mathbf{U} = \mathbf{L} \cup \{(\tilde{\boldsymbol{x}}, \tilde{y})\}$, $\mathbf{U} \setminus \tilde{\boldsymbol{x}}$
  retrain f on $\mathbf{L}$
  update $\mathcal{L}(f)$
end while
return f

To get more into detail we have to discuss the question: How does the query strategy decide? One can use different approaches to assign a utility score to a sample-point:

Informativeness describes the ability of an example to reduce the uncertainty of a model, e.g. uncertainty sampling (UC). In uncertainty sampling, the model's predicted class probabilities are used to derive the classifiers confidence for this prediction. The example, for which the model is least confident, is assumed to carry most information for improving the model. This sample-point is usually located at a decision boundary of the classifier. Two approaches for converting the class probability $\mathbf{p}$ which is a discrete distribution into an uncertainty score are:

1. Each example get assigned the uncertainty

$$\mathcal{S}(\mathbf{p}) = 1 - \max_{c \in \mathcal{C}} p_c, \tag{2.21}$$

that consists of the difference between the most probable class membership and the highest possible probability to be in a certain class, which is one. As in the previous chapters, $\mathcal{C}$ denotes the space of class-probabilities.

2. As a second option, the margin between highest class probability and the second highest one is extracted as (un)certainty

$$\mathcal{S}(\mathbf{p}) = \max_1 p_c - \max_2 p_c \tag{2.22}$$

where $\max_n p_c$ denotes the n-th largest element of $\mathbf{p}$.

Note that a value of zero corresponds to highest uncertainty in Equation 2.22 while it corresponds to being fully certain in Equation 2.21. The query strategy finally returns the example that corresponds to the models maximum uncertainty.

Representativeness measures the quality of the representation of the dataset's underlying distribution $\mathcal{D}$ by the distribution of labeled sample-points like e.g. done by Nearest neighbor criterion (NNC).
Basis of NNC is the nearest neighbor distance

$$\mathcal{N}(\mathbf{L}, \mathbf{U}) = \sum_{\mathbf{x}_u \in \mathbf{U}} \min_{\mathbf{x}_l \in \mathbf{L}} \|\mathbf{x}_l - \mathbf{x}_u\|^2 \tag{2.23}$$

where $\|\cdot\|^2$ denotes the Euclidean norm. Since it sums up the Euclidean distance of each example in the unlabeled pool to its nearest neighbor in the labeled pool, the distance $\mathcal{N}$describes the dissimilarity between the example distributions of the labeled set $\mathbf{L}$ and the unlabeled set $\mathbf{U}$.
This query strategy returns the unlabeled example that decreases the nearest neighbor distance most by converting to the labeled pool. Expressed differently: The example $\mathbf{x}_u$ for which

$$\mathcal{N}(\mathbf{L} \cup \{\mathbf{x}_u\}, \mathbf{U} \setminus \{\mathbf{x}_u\}) \tag{2.24}$$

is minimal is chosen by the NNC.

After taking a closer look on different approaches for querying an example, we have to care about the consequences of manipulating the training-data by Active Learning.

By selecting the sample-points in a defined way, the query strategy, of course, changes the training set for the model due to its characteristics. Thus the distribution of drawn examples $\tilde{\mathcal{D}}$ deviates from the original distribution $\mathcal{D}$. This can lead to a poor generalization ability of the model since it did not have sufficient information about certain regions of the sample space while training. This phenomenon is called *sampling bias* and usually occurs when using exploit-heavy query strategies. The term exploit-heavy stems from the dilemma of balancing exploration (expanding your knowledge) versus exploitation (using your knowledge). The so called exploration/exploitation dilemma is a common problem in Machine Learning and has to be solved e.g. in Reinforcement Learning when the actor on the one hand has to explore the available state- and action-space but on the other hand exploit its gathered knowledge by choosing actions that are already known for their positive reward.

Informativeness-based query strategies, for example, tend to query sample-points with a distribution $\tilde{\mathcal{D}}$ that can strongly deviate from the datasets distribution $\mathcal{D}$. More intuitively spoken, querying can lead to regions without examples in the example space within the training set. Taking classification as an example: Areas of high informativeness between certain classes can lead to a concentration of examples in that region, and even to border-regions between other classes missing entirely in the dataset.

To benefit from advantages of different query strategies, it is also possible to combine multiple strategies. This allows to get over the problem of sampling bias and also decreases the variance between training runs [36]. One example for combining different query strategies is Dynamic Ensemble Active Learning (DEAL) [37]. DEAL is an approach to combine multiple query strategies which are in the following called base query strategies. Furthermore, it is able to weight the base query strategies dynamically over the active learning process. In this way, a base strategy that works well for small training sets can be exploited by DEAL for choosing the first examples, while DEAL can later (with a bigger labeled set) decide to put its trust in another

base query strategy. This is based on the theory of multi-armed bandit (MAB) problems and their extension to multi-armed bandit with expert advice [37]. In this extension the problem converts from finding the arm of a slot machine that gives best reward to detecting the expert that gives the best advice. In case of active learning, the same approach is followed by using the query strategies analogously to the experts and the sample-points instead of the arms. Thus, in every active learning loop (compare Figure 2.12) DEAL chooses an example to query. Dependent on the estimated accuracy increase of the model, a reward is returned to DEAL and DEAL adapts its weights between the base query strategies accordingly. Thus, it evaluates its strategy after every Active Learning loop along the Active Learning process and continuously adapts which base query strategy is best to use at the moment. Since explaining DEAL in detail would go beyond the scope of this work, the reader is referred to [37] and [36] for in-depth explanation.

2.4.2 Transfer Learning

Transfer Learning aims to use knowledge of a source-problem for a target problem.

To be able to compare different Machine Learning problems and determine if a knowledge transfer between them is possible two terms for exact description are introduced: A Domain $\mathcal{D}$ and a task $\mathcal{T}$ ([38], [39]).

The Domain is composed of the feature space $\mathcal{X}$ (space of all possible feature tensors) and the joint probability distribution $P(X)$, where X describes the dataset. Thus the domain $\mathcal{D} = \{\mathcal{X}, P(X)\}$ describes properties of the model's input. The task is defined by the pair of label space $\mathcal{Y}$ and the predictive function $f(.)$ that is learned by the model. Assuming the output of the predictive function f being class probabilities, it can be rewritten as $P(y|x)$ and thus we get $\mathcal{T} = \{\mathcal{Y}, P(y|x)\}$ with $x \in \mathcal{X}$ and $y \in \mathcal{Y}$ denoting one sample-point or label respectively.

One possibility of transferring knowledge from source to target problem is Feature-Representation Transfer. In the case of a Neural Network model the lower layers are responsible for extracting features from input. Under the assumption that error propagation over conveyor segments is similar over all filling and bottling lines, transferability of low level feature extraction (e.g. detecting causality between the stops of two neighbouring machines) can be expected. This can be conveniently used in the case of feature extraction by Convolutional Layers in a Neural Network. In this case, for the target model only the last layers, called decision layers, have to be

trained since the feature extracting layers already exist.

Transferring knowledge from the source (or base) problem to the target problem works the better, the more general the transferred knowledge is. Yosinski et al. mention this for the case of Feature-Representation Transfer:

> *"In transfer learning, we first train a base network on a base dataset and task, and then we repurpose the learned features, or transfer them, to a second target network to be trained on a target dataset and task. This process will tend to work if the features are general, meaning suitable to both base and target tasks, instead of specific to the base task."*[40]

In other words, Transfer Learning gets more promising if source and target problems are as similar as possible:

$$\mathcal{D}_S \approx \mathcal{D}_T \tag{2.25}$$

$$\mathcal{T}_S \approx \mathcal{T}_T \tag{2.26}$$

In the extreme case, when $\mathcal{D}_S = \mathcal{D}_T$ and $\mathcal{T}_S = \mathcal{T}_T$ no Transfer Learning is needed but one model can be generalized to solve source and target problem simultaneously.

2.4.3 Multi-Task Learning

Especially in competitive sports, a new kind of learning and teaching increasingly introduced itself over the last decades named differential learning:

> *"[...] instead of following a direct linear path towards the target of a 'to-be-learned' movement technique by means of numerous repetitions and corrections, a differential approach is more beneficial because it perturbs learners towards more functional movement patterns during practice."*[41]

A very similar and thus probably from human learning borrowed idea was also pursued in Machine Learning. So Yu Zhang and Qiang Yang describe the analogue effect in different words:

> *"Based on an assumption that all the tasks, or at least a subset of them, are related, jointly learning multiple tasks is empirically and theoretically found to lead to better performance than learning them independently."*[42]

So accuracy of a learned model can be improved by multi-task learning which means, from another point of view, that the set of existing labeled sample-points is exploited more efficiently by adding supplementary learning tasks.

In the present work we will take a deeper look into Multi-Task Supervised Learning (MTSL) which can be grouped into three kinds of relatedness between the tasks: feature-based, parameter-based, and instance-based models. Instance based relatedness uses data instances from different tasks in a weighted manner to construct a learner for each task. Parameter-based MTSL models see the relatedness between tasks in the regularization or prior on model parameters while feature-based models assume that tasks can share feature-representations.

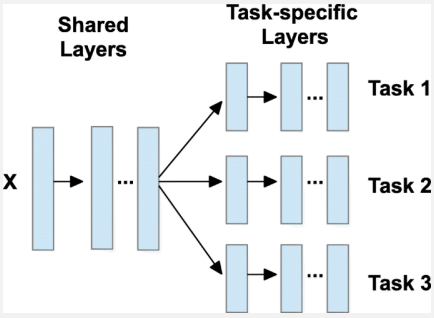
Multi-task learning

Multi-task learning aims to improve the learning of a model for $\mathcal{T}_i$ by using the knowledge from m related tasks $\{\mathcal{T}_i\}_{i=1}^m$.

In supervised multi-task learning (MTSL) each task is a supervised learning task associated with a training dataset $\mathcal{D}_i = \{(\boldsymbol{x}_j^i, y_j^i)\}_{j=1}^{n_i}$ consisting of n_i examples.

Feature-based MTSL

The feature extraction layers of a Neural Network are shared among the models of each task. Thus the feature representation is shared and only the subsequent decision layers are learned exclusively on the tasks data. graphic from [43]



In the current problem, multiple tasks are represented by the different lines that should be analyzed. Thus, the relation between the tasks lies in the patterns that machine states of connected machines show during error propagation which leads to the use of a feature-based approach. Since the combination of Multi-Task Learning in weakly supervised environments will be used in this work, it is also interesting to mention that Multi-Task Learning is shown to decrease the generalization error in weakly supervised environments by Ratner et al. [44].

2.5 Related work

The starting point of the work around this thesis is the so-called 'Avalanche Algorithm' that is used at Syskron GmbH since many years to detect the root-cause machine. This is a rule based system that is based on expert knowledge and takes machine states as input. It is dependent on a time-consuming manual configuration and thus roll-out costs per filling line are high. Since Syskrons Share2Act platform should be provided for around 100 Krones lines per year, one of the biggest challenges is to decrease the roll-out costs heavily.

Voigt [2], Flad [45] and Kather [46] describe model based approaches that show accuracies in a wide range between 40% and 88.7% depending on the plant and the model. Multiple evaluations are based on very few examples and thus limited informative. In 2014, Voigt et al. [47] published a paper proposing model based analysis with a mean accuracy of 95.4% over two plants and this time using 460 test examples for evaluation. One advantage of this model is the possibility to also cover secondary object flows like crates and pallets besides the primary flow. The high accuracies were achieved by heavily adapting the proposed model to the given plant. In terms of roll-out costs, the model based approach is also labor-intensive and expensive since every line that should be analyzed has to be modeled by an expert.

In addition to the model-based approach, Voigt [2] used a model-free approach (Multilayer Perceptron) for his PhD-thesis in 2004. As input, data of all machines in the time window inside 1000s before the lead-machine stop with a sampling rate of 5Hz was used. This data contained machine states reduced to productive, lack, tailback and own-fault. In his approach, firstly the machine state of the lead-machine was analyzed. If the lead-machine showed an own-fault it was returned as the error causer. For the cases lack or tailback two independent Multilayer Perceptrons were trained. Thus error-propagation via auxiliary flows cannot be resolved. Overall only 177 training examples were available. The algorithm could be trained to a high accuracy (98,3% to 100%). But due to missing information about evaluation mechanisms these results are difficult to compare to others.

3 Methods

This chapter leads through the methods that are used in the present work.

First of all, a new approach to unsupervisedly extract characteristic densities for temporal error propagations between pairs of machines is introduced. This information will subsequently be used for automatically configuring rule based algorithms for root cause detection like Syskrons currently productive "Avalanche-algorithm". As next step, an improved rule based algorithm, processing the filling line as a graph-structure and using the extracted characteristic temporal densities is proposed.

To be able to process the data by using Machine Learning algorithms on reasonably sized samples, different approaches for compressing the data are presented. On the one hand, the time-dimension can by collapsed by manual feature extraction, on the other hand, it can be downsampled.

As baseline approach for the time-series classification task at hand, a combination of Dynamic Time Warping and k-Nearest Neighbours is introduced. Furthermore, usage of Random Forest Classifier and the newly introduced Relational Graph Convolutional Neural Network, which both are trained by using data that is preprocessed by manual feature extraction, is explained. Finally, the newly developed spatio-temporal Relational Graph Convolutional Neural Network approach that is based on the Graph WaveNet [30] architecture and processes the temporal error propagation as well as the spatial one, is proposed.

To train the Machine Learning algorithms, a supervised dataset is needed. But expert labels, and consequently supervised data, is expensive and thus sparse. So, additionally, methods to effectively use the available supervised data have to be used. We use Weak Supervision, which includes Algorithmic Labeling, Active Learning, Multi-Task Learning and Transfer Learning for efficiently converting labeling effort into capable models. Thereby, Algorithmic Labeling means using an algorithm, like Random Forest, to create a huge amount of inaccurate labels (compare 2.11). Active Learning decreases the amount of needed supervised training examples while Multi-Task Learning and Transfer Learning, in combination with the introduced

fully convolutional network architectures, enable us to find the error causer at any filling and packaging line by using one single model.

3.1 Probabilistic formulation of temporal error propagation

Due to better explainability and simplicity compared to Machine Learning models, rule-based systems are still an appreciated solution for automatized analysis. Their great drawback usually is the need for an expert configuration which is expensive. In this chapter a solution is proposed that is able to extract the temporal error propagation behaviour of a line from 1-2 weeks of production data and thus only needs configuration regarding the machines and their connections. Since this information has to be available in any system that analyzes a filling and packaging line, the algorithm-specific configuration effort is very low.

3.1.1 Extracting the temporal characteristics of error propagation from data

Let's assume two machines m_i and m_j that are directly connected by e.g. a bottle conveyor. The goal now is to extract the temporal coupling of the machines in case of one stopping the other. E.g.: m_i stops and thus does not bring any more bottles onto the conveyor. Consequently m_j only can continue working as long as there are bottles left on the conveyor using the conveyors buffer nature. The given example is depicted as gantt chart in Figure 3.1. This figure additionally visualizes the used variables τ and t for temporal problem description.

For the described example, the duration between stoppages of m_i and m_j is given by

$$\tau_{ij}^{stop} = N_{ij}/v_{cur,j} \tag{3.1}$$

where the number of bottles on the conveyor connecting m_i and m_j is denoted as N_{ij} while $v_{cur,j}$ represents the current speed of m_j.

The contrary effect in the line is restarting the machines. In this case m_i has to start producing first to provide bottles on the conveyor which have to be transported

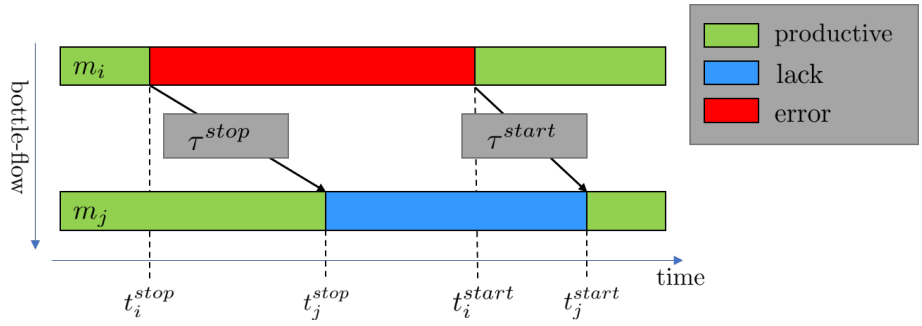


Figure 3.1 *time delay between starts and stops of two machines that are connected by a conveyor*

to m_j to be processed. We call this time delay τ_{ij}^{start}.

But both delays are not constant over multiple occurrences for many reasons. The responsible operator is always an influence. E.g. he often has to accept a message from the machine before the restart. Besides of the operator, machines can be 'blocked' with another machine what means that they have a direct signal exchange and thus e.g. one of the two machines of our example has to wait for some signal of a third machine. Additionally the state of the conveyor before the error propagation heavily affects the temporal delay. For the case of machines stopping the buffer-nature of the conveyor was mentioned. Depending on the filling state (linearly dependent on number of bottles N_{ij}) of this buffer, the subsequent machine has different amounts of bottles left to process which is directly proportional to the remaining productive time.

Thus we interprete every measured τ as a random event of the random variable $\boldsymbol{\tau}$. For better readability let σ be either 'start' or 'stop'. In the following e.g. τ^{σ} is used in formulations that are valid for the stopping as well as the starting process.

To extract the probability distribution $p_{\boldsymbol{\tau}}(\tau)$ over τ^{start} and τ^{stop} for each connection (machine i to machine j) in the line, that means extracting each $p_{\boldsymbol{\tau}_{ij}^{\sigma}}(\tau)$ the following steps are conducted: Gather data of 10 productive days of a line and extract random events τ_{ij}^{σ}. Build up a discrete probability distribution like shown in Figure 3.2. Approximate the discrete distribution by a continuous probability density function $p_{\boldsymbol{\tau}_{ij}^{\sigma}}(\tau)$.

Since the stochastic processes corresponding to the random variable $\boldsymbol{\tau}$ show stationary and independent influences the underlying process is a Levy-process. The inverse gamma function as a member of the family of Levy-distributions proved empirically to provide the best fit results.

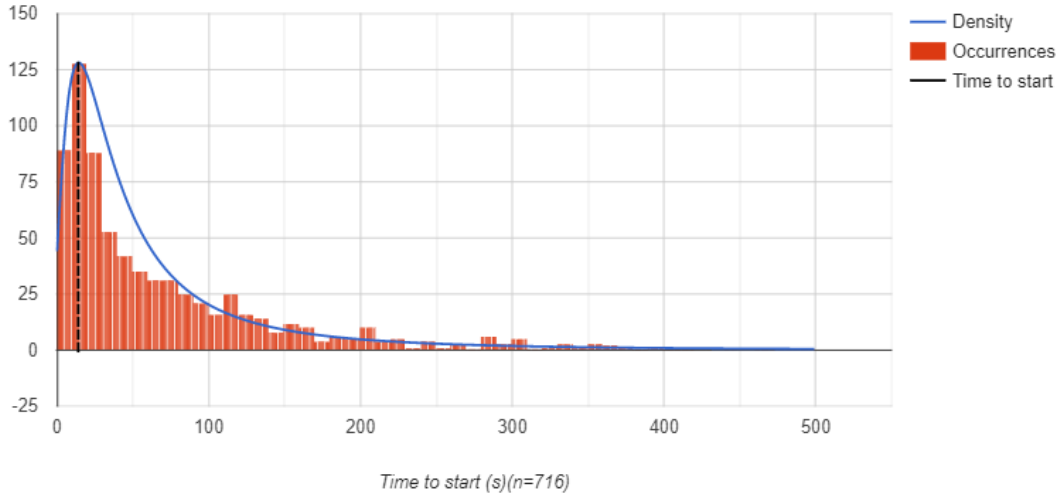


Figure 3.2 *Histogram of τ_{start} over 716 events. Blue line shows fitted inverse gamma function and black dashed line marks the characteristic restart duration*

The inverse gamma distribution is defined as

$$f(x, \alpha, \beta) = \frac{\beta^{\alpha}}{\Gamma(\alpha)} x^{-\alpha-1} \, e^{\left(-\frac{\beta}{x}\right)} \tag{3.2}$$

with corresponding cumulative distribution function

$$F(x; \alpha, \beta) = \frac{\Gamma\left(\alpha, \frac{\beta}{x}\right)}{\Gamma(\alpha)} = Q\left(\alpha, \frac{\beta}{x}\right) \tag{3.3}$$

where $\Gamma(\alpha, x)$ is the upper incomplete gamma function

$$\Gamma(\alpha, x) = \int_{x}^{\infty} t^{\alpha-1} \, e^{-t} \, \mathrm{d}t, \tag{3.4}$$

$\Gamma(\alpha)$ is the gamma function

$$\Gamma(\alpha) = \int_{-\infty}^{\infty} t^{\alpha-1} \, e^{-t} \, \mathrm{d}t \tag{3.5}$$

and Q is the regularized gamma function

$$Q\left(\alpha, x\right) = \frac{\Gamma\left(\alpha, x\right)}{\Gamma(\alpha)}. \tag{3.6}$$

By approximating the given discrete density function by the inverse gamma function we get

$$f(x, \alpha_{ij}^{\sigma}, \beta_{ij}^{\sigma}) \approx p_{\tau_{ij}^{\sigma}}(\tau) \tag{3.7}$$

like exemplary shown in Figure 3.3 and can directly derive

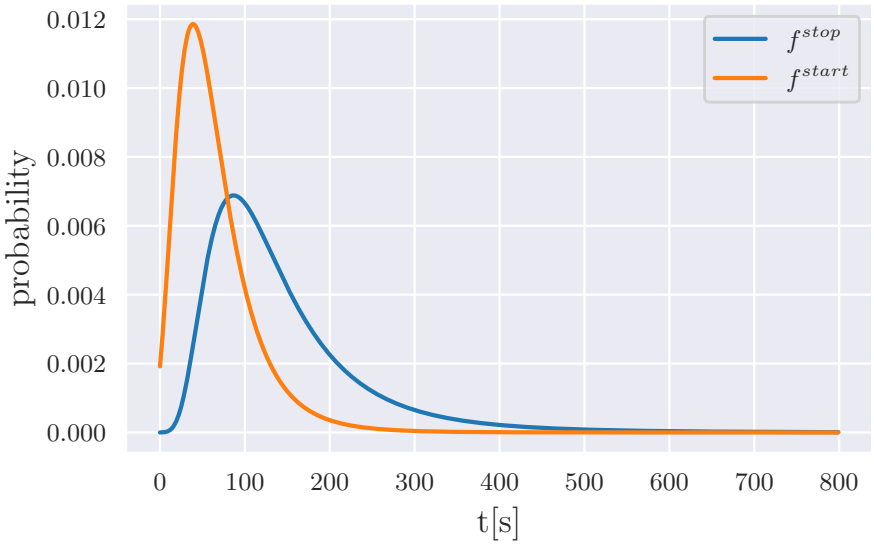


Figure 3.3 *continuous probability density function for temporal delay between causally connected stops and restarts of two neighboring machines. τ^{stop} should be larger than τ^{start} as a result of the conveyors buffer mechanism.*

$$P_{\tau_{ij}^{\sigma}}\left(\tau \leq x\right) = F(x, \alpha_{ij}^{\sigma}, \beta_{ij}^{\sigma}) \tag{3.8}$$

(compare Figure 3.4). For better readability

$$P_{ij}^{\sigma} := P_{\tau_{ij}^{\sigma}} \tag{3.9}$$

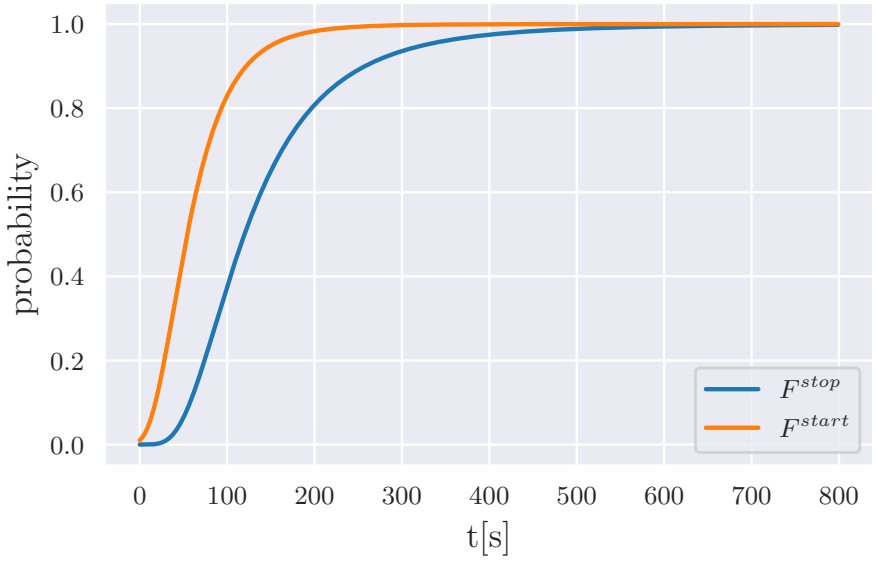


Figure 3.4 *cumulated probability density function for temporal delay between causally connected stops and restarts of two neighbouring machines*

will be used further on.

3.1.2 Causality between stoppages

Using the assumption that f_{ij}^{start} and f_{ij}^{stop} describe characteristics of causally connected stoppages, we can compute a causality score

$$C_{ij}(S) : \mathcal{T}, \mathcal{S} \rightarrow \mathbb{R}_{+} \tag{3.10}$$

describing if machine i has caused a certain victim stoppage S_v of machine j.

$\mathcal{T}$ describes the space of all possible time-series T_i of machine states at machine i and $\mathcal{S}$ the space of stoppages containing start-time and end-time information. For every $t \in T_i$, we compute the probability s that

1. a stop of m_i before t caused m_j to stop

$$\tau_{ij}^{stop} > t_j^{stop} - t \tag{3.11}$$

2. a restart of m_i after t made m_j restart

$$\tau_{ij}^{start} < t_j^{start} - t \tag{3.12}$$

Compare Figure 3.1 for visualization of temporal conditions. The probability of an error at m_i at time t is equal to the probability that both listed dependencies are fulfilled.

Thus

$$\begin{aligned} s_{ij}(t, S_v) &= P_{ij}^{stop}(t_j^{stop} - t < \tau^{stop}) \times P_{ij}^{start}(\tau^{start} < t_j^{start} - t) \\ &= \left(1 - P_{ij}^{stop}(\tau^{stop} < t_j^{stop} - t)\right) \times P_{ij}^{start}(\tau^{start} < t_j^{start} - t) \\ &= \left(1 - F(t_j^{stop} - t, \alpha_{ij}^{stop}, \beta_{ij}^{stop})\right) \times F(t_j^{start} - t, \alpha_{ij}^{start}, \beta_{ij}^{start}) \end{aligned} \tag{3.13}$$

like depicted in Figure 3.5. The probability s can also be expressed using f_{ij}^{σ} as follows:

$$s_{ij}(t, S_v) = \left(1 - \int_{-\infty}^{t_j^{stop}-t} f_{ij}^{stop}(t')\, dt'\right) \times \int_{-\infty}^{t_j^{start}-t} f_{ij}^{start}(t')\, dt' \tag{3.14}$$

Integrating $s_{ij}(t, S)$ over the duration of a potentially causing stoppage S_c at m_i, the causality score c for this stoppage (S_c) is

$$c_{ij}(S_c, S_v) = \int_{\mathrm{dur}(S_c)} s_{ij}(t, S_v)\, dt. \tag{3.15}$$

This score has the potentially causing interval S_c and the victim interval S_v as input.

By summing up $s_{ij}(t, S)$ over all periods where m_i is in an self-induced error $\mathcal{E} \in \mathcal{S}$ the causality score for m_i being the causer of the stop S at m_j can be finally found.

$$C_{ij}(S) = \sum_{\mathcal{E}} s_{ij}(t, S)\, dt \tag{3.16}$$

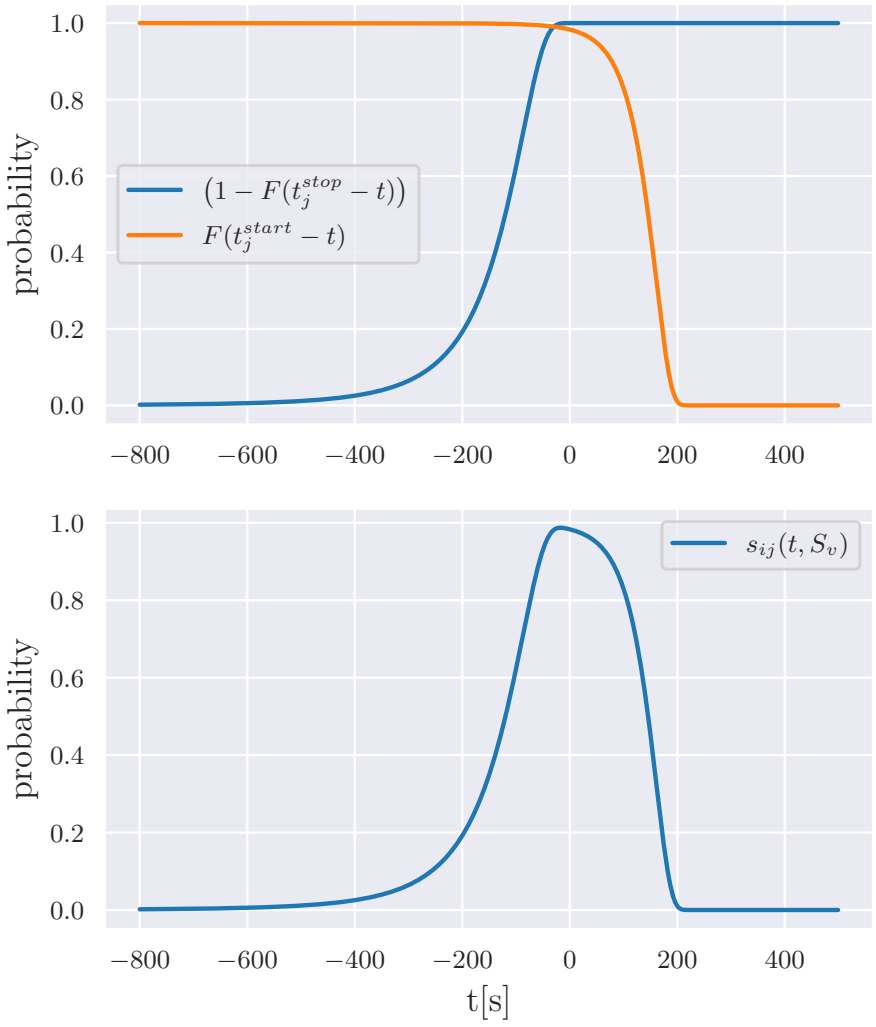


Figure 3.5 *upper: factors of causality score $c_{ij}(t, S)$; lower: causality score $c_{ij}(t, S)$. Both are computed for a stoppage S of m_j from 0s to 200s*

3.1.3 Long range causality

The result of the previous section result Equation 3.15 is derived from f_{ij}^{σ}. But so far f_{ij}^{σ} is fitted to data that should be gathered under the assumption of causality between the stop/start events and thus can only be extracted for direct neighbors.

But under this constraint also the causality of stops can only be computed for directly connected machines. To avoid this limitation and also cover higher order connections, f_{ij}^{σ} can be built up by convolution of all $f_{i'j'}^{\sigma}$ with $|i' - j'| = 1$ along the direct connection between machines i and j:
By using the convolution formula

$$(f * g)(t) := \int_{-\infty}^{\infty} f(\tau)g(t - \tau)\, d\tau \tag{3.17}$$

the distributions for computing the causing score for e.g. two machines i and j which are chosen s.t. $j > i$, can be expressed by

$$f_{ij}^{\sigma} = (f_{i,i+1}^{\sigma} * ... * f_{j-1,j}^{\sigma}). \tag{3.18}$$

3.2 Data handling

This section will lead through the data preprocessing and display all different approaches of preprocessing that are used in this thesis. Additionally, data-preparation steps for the different algorithms are explained.

First of all, we have to define the time-window that is cut from the full time-series of a plant for a given lead-machine stoppage. It should contain all information for analyzing the corresponding error propagation. After preprocessing, the data of this time-window will be used as input for an algorithm. For most customers the error propagation usually does not take more than 10 minutes from causer-machine to lead-machine. For larger filling lines, the error propagation can last up to 15 minutes. Thus we extract data in the time-window starting 10 minutes or 15 minutes respectively before the lead-machine stoppage and ending with the end of the lead-machine stoppage. Since lead-machine stoppages have a variable length, also the time-windows do have variable lengths and can only be used in this shape by algorithms that are able to handle data of dynamic length in time-dimension. And that is usually not the case. The usual approach for this problem is to cut time-

windows of constant lengths. By cutting the time-windows 10s after the beginning of the lead-machine stoppage (Figure 3.6 a) we initially also followed this approach. This leads to fixed time-lengths and provides the highly important information about the machine state during the lead-machine's stop. Although the mentioned approach fulfills the given requirements, valuable information during the lead-machine stop is lost. We thus also use the second option of extracting time-windows of data from usually 10 minutes before the filler stop until its end (Figure 3.6 b).

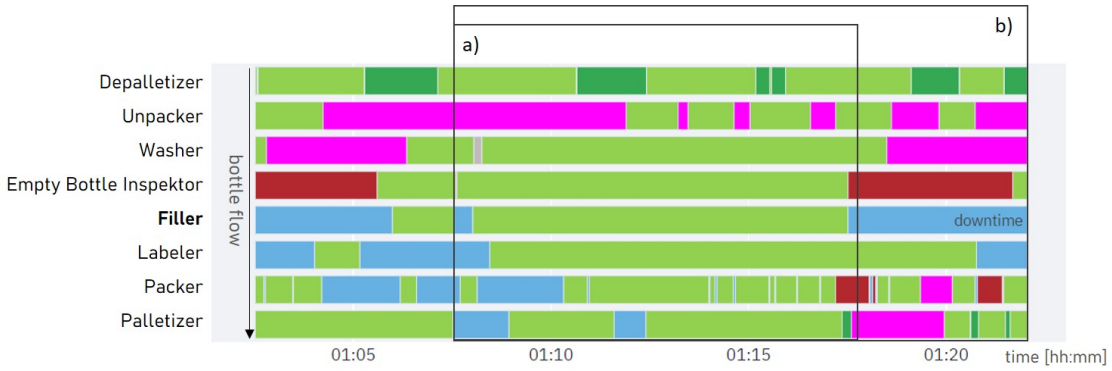


Figure 3.6 *Visualization of time-window of data that is cut per lead-machine downtime. Time in minutes is given on the x-axis, machines sorted along the bottle stream are aligned on the y-axis. The downtime in question is marked by 'downtime' in the chart. Cut time-window usually lasts from 10 minutes before the lead-machine (Filler) stop until a) 10s after the lead-machine stop (static window-size) or b) until the downtime ends (dynamic window size).*

If an algorithm is able to process data of dynamic temporal length this can be used as input. For other algorithms like e.g. Neural Networks that use convolutions for the temporal dimension, we introduce a different approach by converting the full time-window by downsampling the dynamic temporal length to a fixed number of time-steps. This approach will be described in detail in subsection 3.2.1.

After having defined the temporal extraction of data-snippets for each lead-machine downtime, now the different kinds of preprocessing are presented.

First of all, the data arrives in a format that is written 'on-change'. Thus it only possesses entries when a state-change event took place. For every change, information about the new machine-state and the time of the change is provided. We can directly use this kind of data for the rule-based algorithms. But ML-algorithms need a different kind of input.

For ML models either a set of features or time-series data with fixed frequency is needed as input. We achieve that by, first of all, resampling the given data to a fixed frequency (we use 1 Hz) and successively encode the machine states into one-hot representation (compare subsection 2.2.3). In an example, this looks like the following: Let an exemplary machine be in state 'productive' from second 0. At second 3 it stops as a result of an own-fault. We want to look at the data from second 0 to second 8. After resampling to 1 Hz this looks like Table 3.1.

time	machine-state
1	'productive'
2	'productive'
3	'productive'
4	'own-fault'
5	'own-fault'
6	'own-fault'
7	'own-fault'
8	'own-fault'

Table 3.1 *exemplary machine data in resampled style*

By converting the given example to one-hot style we get an array of $(T \times S)$ where T describes the number of timesteps and S describes the number of machine-states (compare Table 3.2).

time	'productive'	'lack'	'tailback'	'own-fault'	'planned-downtime'
1	1	0	0	0	0
2	1	0	0	0	0
3	1	0	0	0	0
4	0	0	0	1	0
5	0	0	0	1	0
6	0	0	0	1	0
7	0	0	0	1	0
8	0	0	0	1	0

Table 3.2 *exemplary machine data in one-hot style*

To cover the entire line the data of machines has to be stacked. Thus we finally get data with shape $(T \times M \times S)$, where M is the number of machines, for every lead-machine stoppage.

For the given example (Figure 3.6 a), 610 seconds of data for a line consisting of 8 machines, the size of one sample-point is $(610 \times 8 \times 5)$ and thus a large number of timesteps. To decrease the size and therefore the complexity of the data and convert it into a less sparse format, we introduce two approaches: Temporal Data Downsampling and Manual Feature extraction.

3.2.1 Temporal data downsampling

To reduce the amount of time-steps an algorithm has to process, downsampling is used on the temporal data dimension. This is done by cutting the time-dimension of one-hot encoded data into equally-sized, non-overlapping slices of length Δ_{slice} and computing the mean value for each slice along the temporal dimension. Thus, downsampling the data of a machine that was productive for 3s and in own-fault for 5s (example like above) in 4s slices looks as following:

$$\mathbf{x}_m = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \rightarrow \begin{pmatrix} 0.75 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0.25 & 1 \\ 0 & 0 \end{pmatrix} \tag{3.19}$$

For real data choosing e.g. $\Delta_{slice} = 60$s means that the time-window before lead-machine stop (usually 600s) is downsampled into 10 slices. The resulting data, thus, consists of these 10 slices plus those that describe the lead-machines stoppage. In case of a 5 min stop this results in an overall number of 15 time-steps after Temporal Data Downsampling.

A second approach is to separate the given time-series into a fixed amount of slices. Then, the duration of one slice is variable but the resulting preprocessed data is of fixed size, independent of the incoming number of temporal steps. This is an important property for most ML algorithms and particularly Neural Network architectures.

3.2.2 Manual feature extraction

This approach fully covers the time dependencies in error propagation and thus the time dimension is collapsed.

We again use one-hot data and thus the given data is of shape $N \times T \times M \times S$ where N denotes the number of sample-points, T the number of time-steps, M the number of machines and S the number of machine-states.

After extracting the features, the shape will be reduced to $N \times M \times S$. Thus, the time dimension with e.g. length 300 is reduced to 1 which yields a reduction of data-size and -complexity by factor 300.

To achieve this, the temporal behavior of error propagation from subsection 3.1.3 is used. Based on the extracted probability density functions characteristic propagation times between start/stop of two connected m_i and m_j are set as T_{ij}^{start} and T_{ij}^{stop}. These characteristic values are set from Syskron-operations staff by using automatically created suggestions like Figure 3.2 that are derived from the fitted density functions.

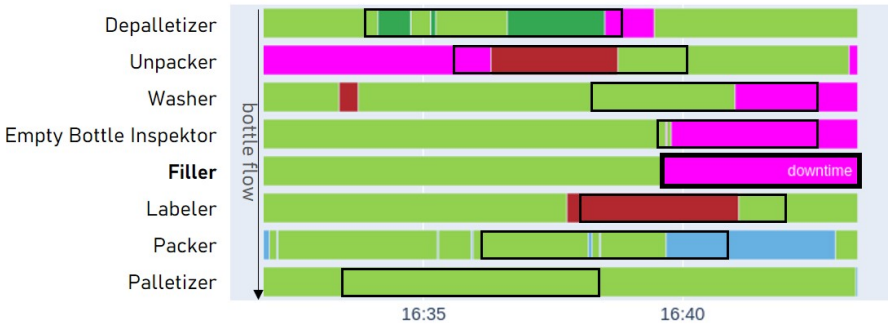


Figure 3.7 *Visualization of manually defined time-regions of interest for the downtime in question. A temporal ratio is computed for each state at a machine inside the given time-window as feature vector.*

By using Equation 3.18, the characteristic stop and start durations T_{ij}^{start} and T_{ij}^{stop} between m_i and m_j can be used to calculate T_{iv}^{start} and T_{iv}^{stop}, the characteristic temporal delays for m_i causing to stop or restart the lead-machine (also called victim machine) m_v. Based on this, time windows $\mathbf{w}_i = [w_i^{start}, w_i^{end}]$ of interest can be derived for every machine m_i in the line with respect to the current victim downtime S_v:

$$w_i^{start} = S_v^{start} - \mathrm{T}_{iv}^{start} \tag{3.20}$$

$$w_i^{stop} = S_v^{stop} - \mathrm{T}_{iv}^{stop} \tag{3.21}$$

Afterwards, for each example, the temporal ratio **r** per machine and state is computed from data inside the given time-window:

$$r(m_i, s) = \frac{\sum_{\mathbf{w}_i} \mathbf{X}_{m,s}}{\sum_{\mathbf{w}_i} 1} \tag{3.22}$$

3.3 Rule based analysis approaches

The goal of root cause analysis is finding the causer for the victim machines (usually lead-machines) stop. A good starting point is using an experts approach when tracing back an error during production. Starting at the victim machine while it is stopped, he/she will study the line for lack and tailback on the conveyors and machines showing own-faults. Usually, the error propagation path is followed backwards step by step through machines and conveyors by deciding firstly, if the causer was found and if not, where the error, that caused the currently analyzed machine to stop, came from.

This basic idea is denoted in algorithm 2 and consists of many steps that need an experts experience to make it work properly. A few examples:

- The error does not only propagate locally but also temporally (compare subsection 2.1.4 for greater detail). Thus extracting the stoppage-causing time-interval at a machine is not trivial! In fact the temporal aspect of error propagation has considerably higher volatility over different line stoppages. Thus powerful algorithmic solutions have to be found to reach the accuracy of an expert's view in this task.

- What is about multiple stops at one machine? Which of them contributed to the stoppage of the connected machine? An expert intuitively groups non-productive time-spans to erroneous phases that act like one interval with insufficient machine performance.

Algorithm 2 Basic error tracing

Input: sample-point **x**, adjacency matrix **A**,
victim machine m_v, stoppage of victim machine downtime S_v
Output: class probability distribution **c**

```
m (machine under investigation) = lead-machine
S (stoppage at m) = S_v
while m is not None do
    s_m = x(m, t)                          ▷ machine state of m
    if s_m ≠ 'own-fault' then
        m = A(m, s_m)                      ▷ find next machine to look at
        extract S_m (causer of S)          ▷ extract stoppage of m
        S = S_m
    else
        c(m) = 1                           ▷ causer was found
        m = None
    end if
end while
```

To translate this into an algorithm we use the results of section 3.1 and take a closer look into finding the stoppages of m_i that caused a certain stoppage S_{m_j} at m_j. This step has to be repeatedly done for every station on the trace to find the error source.

Like denoted in algorithm 3, when 'sitting' on the stoppage interval S_{m_j} and searching for its source at m_i, firstly all stoppages of m_i have to be extracted from the given data. Now we use Equation 3.15 to assign a causality score $c_{ij}(S, S_{m_j})$ to every stoppage S in the set of stoppages at m_i ($\mathbf{S}_i$) and sort them by this score. In that way, we can now split up the caused stoppage duration beyond the stoppages of m_i like described in 3. Since conveyors are built with buffering capability, the duration of an error that propagates from one machine to another, should always shrink. Thus, we maximally assign the own dur(S) as the caused-stoppage-duration δ_c to any causing stoppage.

In this way, the caused nonproductive time is spread beyond stoppages due to the computed causality score and consequently scenarios with more than only one root cause can be resolved.

Algorithm 3 Temporal tracing

Input: sample-point $\mathbf{X}$, machine under investigation m_i, last stoppage on trace S_{m_j}
Output: Set of stoppages that caused S_{m_j}

```
extract S_i                                          ▷ set of stoppages of m_i
compute causing score c_ij(S, S_mj) for every S ∈ S_i
sort S_i by causing score
set victim-stop-duration δ_v = dur(S_mj)
causing-intervals C = []                  ▷ Initialize list of found causing intervals
for S in S_i do
    set caused duration δ_c = min(δ_v, dur(S))
    δ_v = max(δ_v − δ_c, 0)
    if δ_c > 0 then
        append (S, δ_c) to C
    end if
end for
    return C
```

3.4 General methods for Machine Learning approaches

In this section, the methodical approaches that are common beyond all kinds of used Machine Learning are discussed. This starts with filtering the dataset. Contrary to the above mentioned rule-base approach that is also able to classify areas between two machines, the ML-models will be trained to select the causer from the set of machines that provide data. Additionally, stoppages that are labeled as 'else', implying that the causer was not within the available classes or the expert was not able to detect the root cause himself, are filtered.

3.5 Dynamic Time Warping and k-Nearest Neighbors

Since DTW has the ability to compare Time Series data of different lengths, we could use "raw" one-hot encoded data in this approach. But the greatest drawback of DTW are high computation times. To decrease these we reduce complexity of the input data by using Temporal Data Downsampling according to subsection 3.2.1 with fixed downsampling window-sizes Δ_{slice}. Thus, time-series data is decreased in size by a factor of $\Delta_{slice}/1\text{s}$ since original data is sampled in seconds. But examples still differ in their temporal length. For this, the capability of computing similarities

between time-series of different lengths is utilized.

We use training- and test-sets with amounts of labels like given in Table 3.3 for each customer.

	# training examples	# test examples
customer a	439	100
customer b	439	100
customer h	328	100
customer w	102	100

Table 3.3 *Number of training- and test-examples for DTW and k-NN training*

3.6 Random Forest

As one of the simpler classification approaches, Random Forest can also be trained on significantly smaller datasets than Neural Networks. Thus we train it using the experts labels.

	# training examples	# test examples
customer a	424	80
customer b	525	80
customer h	80	80
customer w	112	80

Table 3.4 *Number of training- and test-examples for Random Forest training*

The minimal amount of labeled examples per customer is 160. To leave a sufficient amount of examples in the training set for every customer we choose 80 as size for the test-set per customer. This results in sizes for test- and train-set like denoted in Table 3.4.

As model input, we use data preprocessed by manual feature extraction (compare subsection 3.2.2). To cover the problem complexity, the amount of used trees has to be increased to 500 and the maximal tree depth is set to 7.

3.7 Algorithmic labeling

Since manually labeled, high-quality labels are very rare, training of complex Machine Learning models like large Neural Networks tends to heavy overfitting with these

labels. A significantly larger amount of labeled data is likely to overcome this problem. Thus we are using algorithmic labeling in this work.

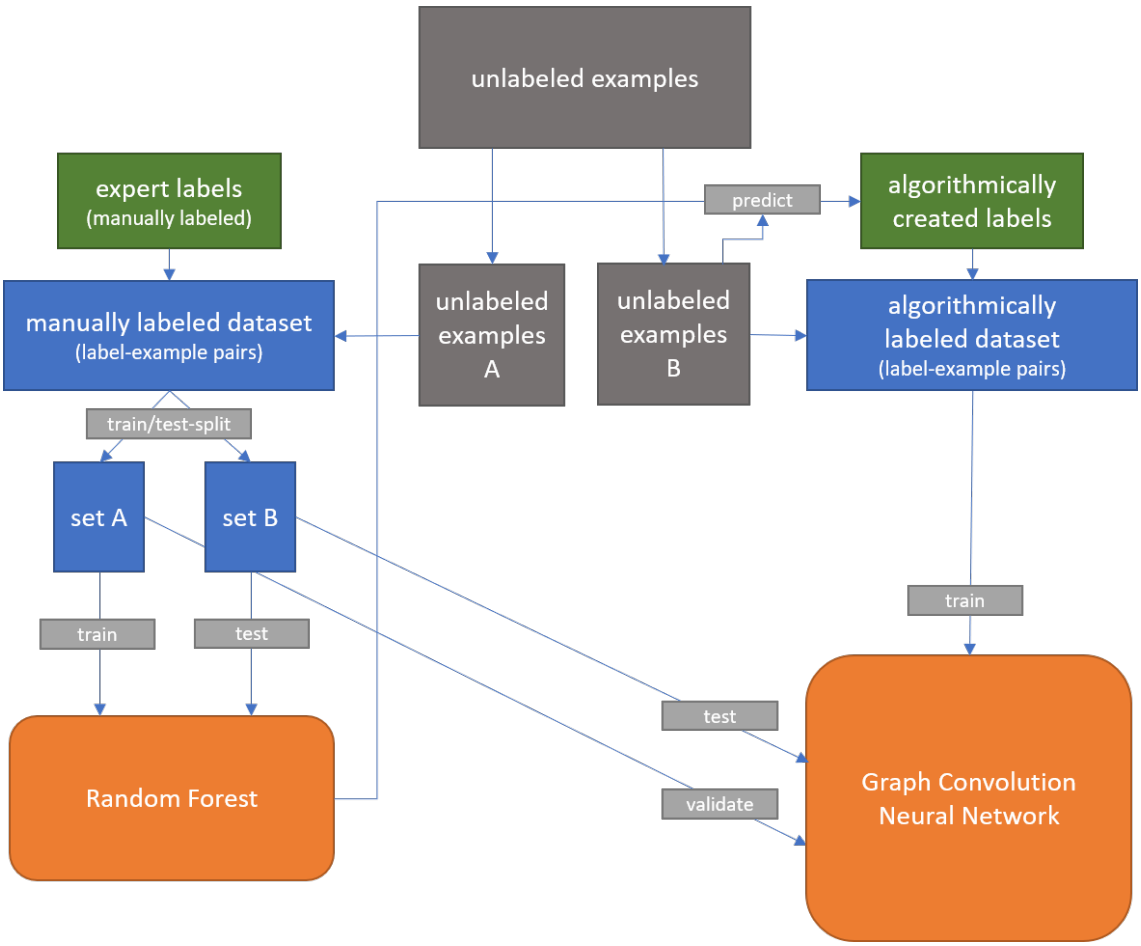


Figure 3.8 *Algorithmic labeling pipeline: Low amount (starting from ≈ 100) of manually created labels is used to train a Random Forest classifier. This afterwards provides predictions for the remaining unlabeled examples which are used as algorithmically created labels. The resulting, significantly larger dataset enables improved Neural Network training.*

Like depicted in Figure 3.8, a Random Forest model is trained on manually labeled data. Therefore the available expert labels are combined with their corresponding examples to form the so-called manually labeled dataset. This set is split in subsets A and B. In the first step, these two sets are used as train- and test-set for Random Forest training. At this step, we can use the Random Forest model to algorithmically create labels for the examples that do not possess a manually created label.

The combination of algorithmically created labels and their corresponding examples is the desired, large dataset. This will be used for Neural Network training.

For a clean evaluation of the GCNs we also need a validation-set in addition to the test-set. It is crucial that the test set consists of examples which never contributed information to any step inside the training pipeline. Since set A was used to train the Random Forest which created the training labels for the Neural Network, this set cannot be used as test-set for evaluation of GCN-training. Contrary, set B was only used to evaluate the performance of the trained Random Forest model and thus its contained information was only used for evaluation and never for training. Thus we use set B, the test-set corresponding to Random Forest training, as test-set for analyzing the GCN performance. During training the GCN, we use the validation-set for estimating the models generalization ability. For this task, also label-example pairs that already contributed their information during the training pipeline can be used. Thus we use set A as validation-set for Neural Network training.

Since Random Forest training is possible with a minimum amount of 100 to 200 examples, heavily depending on the complexity of the filling line, this approach enables us to create large, algorithmically created datasets and thus also train Neural Networks after collecting only a small amount of manually labeled examples.

We expect this approach to provide significantly higher accuracies than rule-based algorithms but lower accuracies than a human expert.

3.8 Active Learning

We want to use Active Learning to achieve the best information content out of a given amount of time/effort of an expert on site to train a model on the elaborated information. The usual approach is pool-based active learning. Following this approach would lead to maintaining a large pool of unlabeled data that is constantly updated with every stop of the lead-machine. From time to time an expert would decide to label some sample-points that are selected from the huge pool for him. But how does the expert decide which label to give? He can take the data as his information for the error. But then the big advantage of an expert on site, who is knowing what really happened beyond data inaccuracies etc., is lost. Thus, our oracle (expert) needs to have the respective stoppage in mind. And this is only the case if the stoppage happened in the ongoing shift.

For that reason, classic pool based learning is not the perfect option in this case. Instead, we introduce a new form of Active Learning. We build up a pool of unlabeled examples for every shift. Since the lead-machine stops around 20-50 times per shift, there are enough data-instances to provide a proper pool. Examples from this pool are selected by the query strategy and directly addressed to the oracle (expert). He has the knowledge about the errors of this shift and thus can exactly label the line-stoppage. In this way, fast model training combined with heavily reduced expert work can be achieved.

During detailed comparison of many different query strategies ([36]), the combination of Uncertainty Sampling (UC) and Nearest Neighbour Criterion (NNC) by using Dynamic Ensemble Active Learning (DEAL) proved best on the given data. For comparability to training on the full manually labeled dataset we use Random Forest with 500 estimators and a maximal depth of 7 as classifier.

3.9 Multi-Task Learning and Transfer Learning

In this work, we try to decrease the effort per roll-out of a Machine Learning algorithm for one filling and packaging line. The optimal solution for the case of a large number of roll-outs is to have an initial effort (e.g. training a ML-model) and then zero additional effort per customer. This can be achieved if the trained model is able to detect the root-cause machine for any line. Since each new line corresponds to a new classification task, the model has to be able to solve multiple tasks (Multi-Task Learning) when being trained on all available training examples from different lines. Additionally, also tasks(lines) that have not been represented in the training (Transfer Learning) must be solved. Combination of both approaches is a manageable task since both approaches are highly similar as Caruana et al.

> *"Multitask Learning is an approach to inductive transfer that improves generalization by using the domain information contained in the training signals of related tasks as an inductive bias."*[48]

and also Ruder et al.

> *"We can view multi-task learning as a form of inductive transfer."*[49]

state.

Practically spoken: We want to train a Neural Network on all tasks/filling lines that provide labeled data and use the trained model to classify the root cause machine

at all lines that provide sufficient data. Even those without any labels.

Each task consists of converting data of a line downtime $\mathbf{x} \in (M \times T \times S)$ into a vector of class probabilities $\boldsymbol{y}$ where the set of classes $\mathbf{C}$ is equal to the set of machines $\mathbf{M}$ (comapre Equation 2.7). Thus we allocate a class probability to every machine that is available in the input data $\mathbf{x}$.

This leads to challenging requirements to solve for one single model: The model has to be capable of

- processing input of variable shape (different number of machines per line)
- providing a variable shape of class probabilities (different number of machines per line)
- using the same set of network-parameters(weights) for all tasks

In the following section, we introduce a new Neural Network architecture that was build up to meet these requirements.

3.10 Fully Convolutional Graph Network architectures

In this work, all Neural Networks are built up and trained using PyTorch and PyTorchGeometric. As Optimizer, Adam [50] was used on default parameters: learning-rate = 0.001, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-08}$, weight-decay = 0.

We stack Graph Convolutional Layers to build up a Fully Convolutional Graph Neural Network which satisfies the above (section 3.9) given requirements towards solving Multi-Task Learning and Transfer Learning. Our structure shows one major difference (which enables us to conduct Multi-Task Learning and Transfer Learning) compared to the usual Fully Convolutional Classification architecture (e.g. [51]). In our case the common approach would look like shown in Figure 3.9).

In the last layer, the number of channels is equal to the number of classes and a global pooling operation delivers one real number for every channel by collapsing the graph (calculate a mean value over the nodes of each channel). The resulting vector is finally converted to class probabilities by a softmax-layer.

But, in our case we want to know which node of the graph was the error causing machine. Thus, we choose only one output channel for the last convolution layer and correspondingly we only compute one feature for each node in the final convolution

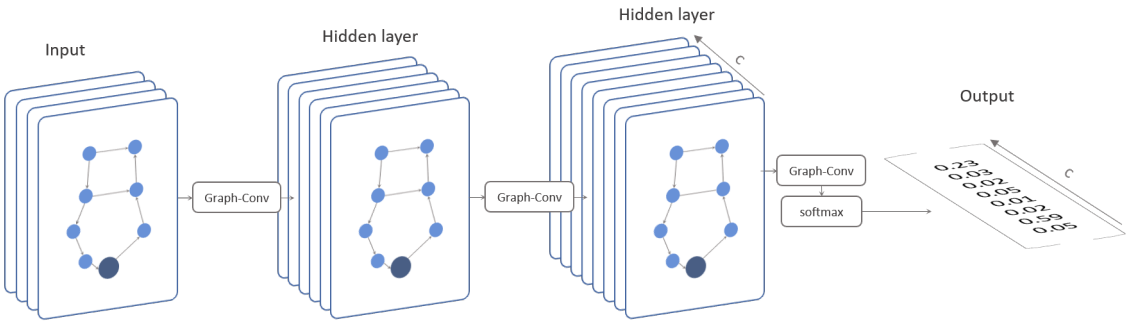


Figure 3.9 *Common fully convolutional Graph Neural Network architecture. This architecture uses Global Pooling on each channel after the final Convolution Layer. In this case, the length C of the resulting class probability vector is equal to number of channels of the final Convolutional Layer*

layer (compare Figure 3.10). Finally, this is converted to class probabilities by using softmax-activation.

For better understanding the difference in the final layer, Figure 3.9 and Figure 3.10 visualize the dimension that provides the class probabilities in the output layer. Usually, these are given along the dimension of channels (features per node). In the proposed architecture, the probabilities are returned as a graph of nodes with one feature, that is trained to represent the class probability.

Since the architecture of combining the layers is set, we now take a closer look at the used graph information. Looking at a filling and packaging line, one intuitively depicts the graphs edges as directed connections representing the material flow. Contrary to the directed connections, errors can propagate in both directions along the conveyors.

The algorithm tries to reconstruct the path of error propagation before a lead-machine stop from machine state data and therefore from lack and tailback information. But to convert the information of lack or tailback to knowing where the error came from, one needs to know the direction of material flow.

To provide this information to the GCN we use multiple relations (compare Equation 2.19) between the nodes. One for downstream (compared to material flow) propagating information which means we provide the adjacency matrix corresponding to the graph of material flow to the model. The second relation represents the upstream propagating information while the third relation connects each node to itself and thus enables the model to 'remember' information through the layers. The

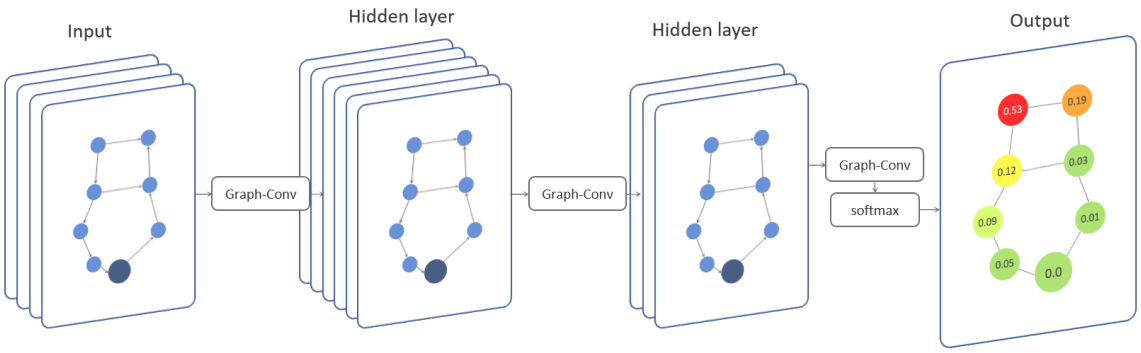


Figure 3.10 *Newly introduced fully convolutional Graph Neural Network architecture. In this case, a class probability is provided for each node by reducing the channels to one and applying the softmax activation on the graph's nodes. This enables us to use Multi-Task and Transfer Learning over multiple lines.*

last mentioned connections are usually called self-loops.

To evaluate the models, in each run a validation-set is used to monitor validation accuracy after each training epoch. This, on the one hand, enables to detect overfitting. On the other hand we select the model, showing highest validation accuracy during the training process, as the best trained model and compute the test accuracy for this. For greater insight we do not only compute one test accuracy for the full Multi-Task dataset but also for each customer. The test-sets are built up in such way, that every customer's test-set is used for the individual test-accuracy. For the average test-accuracy of Multi-Task Learning the test-examples are combined to one test-set. To avoid one customer being overweighted in the combined test-set the number of test examples per customer are equal over all customers.

3.10.1 Relational Graph Convolutional Neural Network (RGCN)

We build up the RGCN as a three or four layer architecture consisting of Relational Graph Convolutional Layers like introduced in Equation 2.19 and Equation 2.20. Therefore, the layer-implementation provided in pytorch-geometric [52] is used. To provide the nonlinearity in the model, ReLU activation in all layers except the last, which possesses a softmax activation, was used. The network is trained and evaluated with data preprocessed by Manual Feature Extraction (compare subsection 3.2.2).

When using algorithmically labeled data, test-sets contain 64 examples per customer while size of validation- and train-sets vary since the train-set consists of the

algorithmically labeled examples and validation set corresponds to the train set of Random Forest which was used for algorithmic labeling.

If manually labeled data is used, the examples of one customer are split into a test-set of size 32, an equally sized validation-set and the remaining examples are used as training-set. In this case, the validation- and test-sets are very small and thus the corresponding quality measures are highly volatile beyond different splits of data into train-, validation- and test-sets. Due to using at least three customers for Multi-Task-Learning and Transfer Learning, the accumulated test- and validation-sets have a size of 96 (three customers) or 128 (four customers) which is a sufficient size for proper testing.

3.10.2 spatio-temporal Relational Graph Convolutional Neural Network (stRGCN)

In this architecture, we use three blocks containing three layers each, like described in section 2.3.5. After the last block, two 1x1 convolutional layers with ReLU activations (compare section 2.3.5) are added (similar to the final layers of each block) to reduce the number of channels (features) per node to one. Finally, the class probabilities are computed by a softmax activation over the nodes. Thus, we again achive the proposed fully convolution architecture.

Furthermore, the number of channels is set to 16 for all convolution operations and dropout is set to 0.2 for the output of the graph convolutions.

3.11 Adding lead-machine information to data

Knowledge about the filling and packaging line is crucial for correctly classifying the root-cause of a lead-machine. Particularly, knowing which machine is the lead-machine inside the line is needed to backtrace the error-propagation starting from the correct machine. Thus, an approach to provide this information to the stRGCN is presented in this section.

Reminder: The data array of one lead-machine stoppage is of shape $(T \times M \times S)$ where T represents the number of time-steps, M is the number of machines and S displays the number of machine-states.

To provide the information to the Neural Network, we add one more input-channel to the data. That means, we expand the 'machine states' dimension of the data by

one and add a matrix of shape $(M \times T)$. The matrix is binary and thus consists of ones and zeros only. In this way we mark the lead-machine with ones and all other machines with zeros. Thus e.g. data of a stop at a line consisting of four machines (lead-machine is the third machine) with five time-steps of data additionally gets the matrix

$$\mathbf{x}_m = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{3.23}$$

appended.

4 Results

In this chapter, rule-based approaches, Dynamic-Time-Warping and k-Nearest Neighbors, Random Forest and Graph Convolutional Neural Networks (GCN) are evaluated by comparing their accuracies when classifying the correct root cause for a lead-machine stoppage. Additionally, results of the proposed label-efficient approaches Multi-Task Learning, Transfer Learning and Active Learning are analyzed.

For Multi-Task Learning, the special property of our fully convolutional GCNs is exploited and each model is trained on all customer's data at once. An extension of this approach is done by additionally combining it with Transfer Learning. For that purpose, we train the model on all but one customer's data using Multi-Task Learning, and afterwards evaluate it on the left-out customer by using the Transfer Learning approach. If this combination reaches sufficient accuracy, the model can be used at a new customer without initial training effort and thus "out of the box".

Last but not least, savings in labeling-effort by using Active Learning in combination with a Random Forest Classifier are quantified.

4.1 Data characteristics

This section contains background knowledge about the customers' filling and packaging lines and peculiarities of some machine's data. Their effects will show up in multiple results.

First of all, a data-example corresponding to customer-b (Figure 4.1) shows an extreme example of bad data quality. In reality, an error occured somewhere near Unpacker, thus this machine should show some non-productive status. Afterwards, the Depalettizer stopped in 'tailback' and the error also propagated downstream towards the Filler with stopping the Washer on its way. Finally, a sensor before the Empty Bottle Inspector (EBI) made the Filler duo stop due to missing bottles in the entry. Furthermore the EBI gets a stop-command via electronic data exchange from

Filler and thus shows the status 'tailback'. Due to the direct information exchange between Filler and EBI, those two machines are called *blocked*.

Looking at Figure 4.1 shows data that is suspicious for three machines. As already mentioned, the Unpacker should show some non-productive status. Additionally, the Labeler constantly shows an own-fault status although the machines before and after it are productive. Since buffer-capacity of conveyors equals roughly the amount of bottles that can be produced in three minutes, data of Labeler must be erroneous. In the same time Packer shows "productive" state which has to be erroneous by the same argumentation. At customer-b, these data inconsistencies only rarely occur since machine-state output at all machines was checked and, in case of errors, corrected after connecting the industrial data acquisition. At this point, it is to mention that the data at the investigated customers was monitored very well and thus it can be assumed that the corresponding data-accuracies are above average.

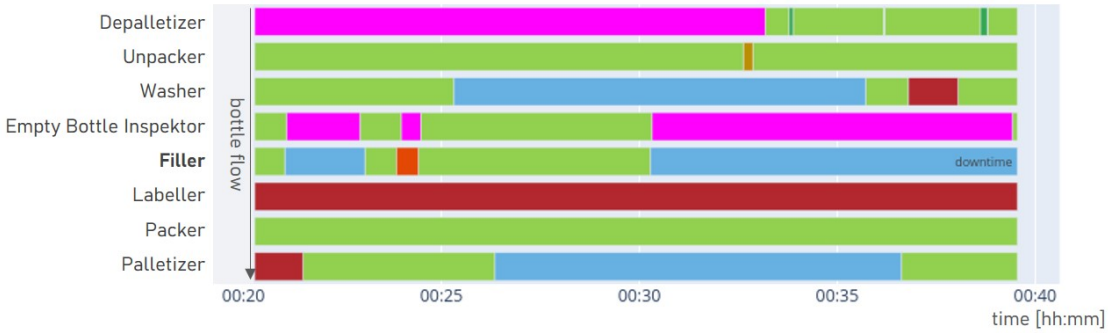


Figure 4.1 *Gantt chart showing data inconsistencies at customer-b. Production is displayed in green, tailback in pink, lack in blue and equipment-failure in red. Incorrect status for Unpacker, Packer and Labeler since in reality all machines stopped caused by a lack of crates/bottles from line's beginning.*

Based on data evaluation we assume, that an algorithm usually has to classify about 10% to 20% of downtimes based on data with bad quality which heavily increases the problem complexity for these cases. If an important time-window of a machine under investigation carries an erroneous machine-state, this almost certainly causes a wrong classification when using rule-based algorithms. In case of Machine Learning approaches and especially Neural Networks, it is possible to overcome this problem for part of the low quality examples.

In the following, we look deeper into the data of each customer and elaborate some characteristics.

Unpacker of customer-b regularly shows state "tailback" when its speed is decreased by the line-control. In this scenario, the buffer between Unpacker and Washer gets filled during production as a result of the V-graph (Figure 2.2) and corresponding higher nominal speed of Unpacker compared to Washer. When the buffer is filled, the line-control decreases the Unpacker's speed to the Washer's speed. Thus, although Unpacker runs with decreased speed as a result of tailback, it is in normal production and should show state "productive".

Empty Bottle Inspektor of customer-a was changed during data collection. The new machine often returns state "held" in case it was stopped by a signal from Filler as shown in Figure 4.2.

Figure 4.2 *Gantt chart showing wrong machine state of EBI at customer-a in case of lack from Depalletizer. Thus EBI should show machine-state "lack" when stopping as a cause of Depalletizer error. Production is displayed in green, "tailback" in pink, "lack" in blue, and "held" and "off" in grey. "held" and "off" are own-fault states of the machines.*

Since a machine should be in "held" if it was manually stopped by an operator, this state is an own-fault state. Consequently, the EBI is shown as having stopped due to an own-fault in the example (Figure 4.2). When analyzing this example, one has to trace back the error path from filler upstream and finds the EBI in own fault. Thus it is reasonable for an algorithm to select the EBI as causer. Looking deep into detail, one can get a hint what really happened when looking at the exact moments of stopping for Filler and EBI. Since EBI stops slightly after the Filler the information flow probably went from Filler to EBI. And this leads to the assumption,

that EBI was not the real causer. But this tiny pattern is hard to evaluate for an algorithm and would not be needed with correct machine states.

Packer at customer-w shows "intermediate" state in tailback situation. For all examples, that are labeled with Palletizer as causer-machine the Packer shows machine state "intermediate" although "tailback" was the correct one. During data-preprocessing for Machine Learning "intermediate" is converted to an "own-fault" state (Table 2.1). Thus, classifying the corresponding examples as caused by the Packer is a reasonable behavior of an algorithm but will be evaluated as wrong classification later on, since the stoppage of the Palletizer is the root cause.

Figure 4.3 *Gantt chart showing erroneous machine state of Packer at customer-w in case of tailback from Palletizer. Thus Packer should show machine-state 'tailback' when stopping as a cause of Palletizer's error. Production is displayed in green, "tailback" in pink, "intermediate" in grey and "equipment-failure" in red. "intermediate" and "equipment-failure" are own-fault states of the machines.*

Blocked machines at customer-w are directly connected to each other without bottle transport on conveyors. Due to producing and filling non-returnable plastic bottles, this line is equipped with a so-called ErgoBlock which is a combination of Blower, Labeler and Filler. Thus there is no buffer between the machines and they stop as one when an error appears. Note the simultaneous stopping and starting behavior in Figure 4.3.

Labeling experts differ across the customers. For customer-a, the labels were given from line-experts without looking at the corresponding data. That opens up problems that are based on different standards used by the expert and the algorithm for determining the causer machine.

Besides the mentioned data-inconsistencies at the EBI that lead to predicting the EBI as causer when the Depalletizer is labeled, there are additional examples that are labeled with Depalletizer as the causer although EBI was correct from a data perspective. In this case the expert sees a quality problem of returned bottles (returnable bottles that are already used and input to the filling line). Then, often the washer is not able to clean old labels off the bottles. Those bottles are afterwards detected and rejected by the EBI. If there are too many rejects, the EBI stops with an appropriate message that informs the user about the rejection cause. For the expert, the problem in glass quality is not connected to a machine and thus the convention is to choose the very first machine in the line, the Depalletizer, as causer. From an algorithm's point of view, the causer-machine is the EBI since it stopped the Filler. And due to the EBI's message, the information about problems with cleaning and thus glass quality is also worked out. Thus expert's label and algorithm's prediction carry the same information about the very root-cause of the problem in the filling line, the insufficient quality of returned bottles. But different root-cause machines are selected which leads to evaluating these examples as wrong classifications.

At customer-a about 30% of the data consist of examples with the described problem of different analysis standards. Discussions with the customer regarding results of the newly introduced rule-based system revealed that most of the predictions in question are sufficient for their analysis and only another point of view on the problem. Thus we can assume an increase of accuracies for customer-a by around 20% to 30% for customer-a given the fact that they adapt the bigger part of our problem-interpretation standards.

For customer-b, the customer's expert labeled the examples based on a data visulization without information from inside the line but with using the algorithms prediction as proposed result. Thus, the approach is the contrary to that of customer-a. The basis of root-cause machine classification, the data, is equal for algorithm and expert. Additionally, the standards of interpretation were aligned during the labeling work by repeatedly comparing the algorithms result with the experts estimation and thus understanding and adopting the algorithms standards. This results in comparably high accuracies for customer-b.

In the following results, we will witness severely different accuracies between customer-a and customer-b as a result of the described labeling characteristics although their lines are very similar and thus also the classification tasks should be of similar difficulty.

The labels for customers h and w are created by Syskron and Krones experts. Thus, the interpretation standards are, similar to customer-b, well adjusted with those of the algorithm.

4.2 Rule-based approaches

In this chapter, at the one hand, we look at accuracies of Avalanche-algorithm which is the currently active approach at Syskron GmbH, one the other hand we evaluate results of the newly introduced rule-based system, introduced in section 3.3.

4.2.1 Avalanche algorithm

This algorithm is currently used in Syskron's productive systems and therefore sets the general baseline for results.

The algorithm is a rule-based system and analyses temporal correlations in machine states over the line to estimate probabilities for each machine for being the causer-machine. Due to compliance issues, the approach can not be described in deeper detail.

	test accuracy [%]
customer-a	47.4
customer-b	67.6
customer-h	48.1
customer-w	63.1

Table 4.1 *Accuracies of Avalanche algorithm which is currently active in Syskron's productive system. The goal of this thesis is to clearly outperform the given results.*

4.2.2 New rule-based approach

This approach uses the representation of the line as a graph and backtraces each error propagation from the lead-machine stoppage to its source. We will firstly analyze accuracies and confusion matrices. Additionally, the false classifications are analyzed in detail and therefore divided into three error-categories: Errors that were caused by miss-classification of the algorithm itself, those that ware caused by erroneous data and finally differing interpretation standards between expert and algorithm.

For customer-h, an accuracy of 70% shows a respectable result considering the complexity of the line. The improvement of 21% compared to Avalanche algorithm

is huge and can be explained by the utilization of the graph structure of a filling line. Since the corresponding line is a large one with many inter-dependencies, the information about these dependencies is crucial for an algorithm. In contrast to the new rule-based algorithm, Avalanche-algorithm is not able to exploit this information.

	test accuracy [%]
customer-a	53.5
customer-b	92.1
customer-h	69.5
customer-w	68.5

Table 4.2 *Accuracies of the new rule-based approach.*

Since customer-w is a very simple line architecture, the accuracy of 69% is lower than expected. Additionally an improvement of only 5% compared to Avalanche is significantly lower than at customer-h. In this case, the factor that mainly limits the accuracy is not the complexity of the line, which is very simple, but the data-quality.

To get a better insight, will investigate different influences which lead to miss-classifications for customers a and b in the following.

For customer-a, a comparably low accuracy of 53.5% was achieved. By looking deeper into detail of the miss-classified examples, 11.8% of examples were falsely classified due to erroneous data, 5.2% because of a wrong classification by the algorithm and the largest part, namely 29.5% due to differing interpretation of data by the algorithm and expert.

At this step, the expert's approach has to be explained. At customer-a, an expert labeled the examples for calculation of internal key performance indicators (KPIs) while working at the line. Those estimations are not primarily done for the used dataset. And thus, in contrast to the algorithm which selects the causing machine, the expert often directly goes one layer of analysis deeper and primarily extracts the general problem, that occurred in the line, like bad glass quality or manpower shortage. Since these problems are not directly corresponding to a machine and the main focus of the expert for the analysis is satisfied, he simply chooses a machine to allocate the error. On the other hand, it is very important to firstly find the correct causer machine from an algorithms perspective. Proceeding from this result, the analysis can find the cause for the causer-machine's stoppage which is provided by the error-message of the machine. Based on these different approaches, in 29.5% of the examples, neither the expert's, nor the algorithm's result are wrong but still they provided different causer machines. Thus the evaluation counts them as wrong

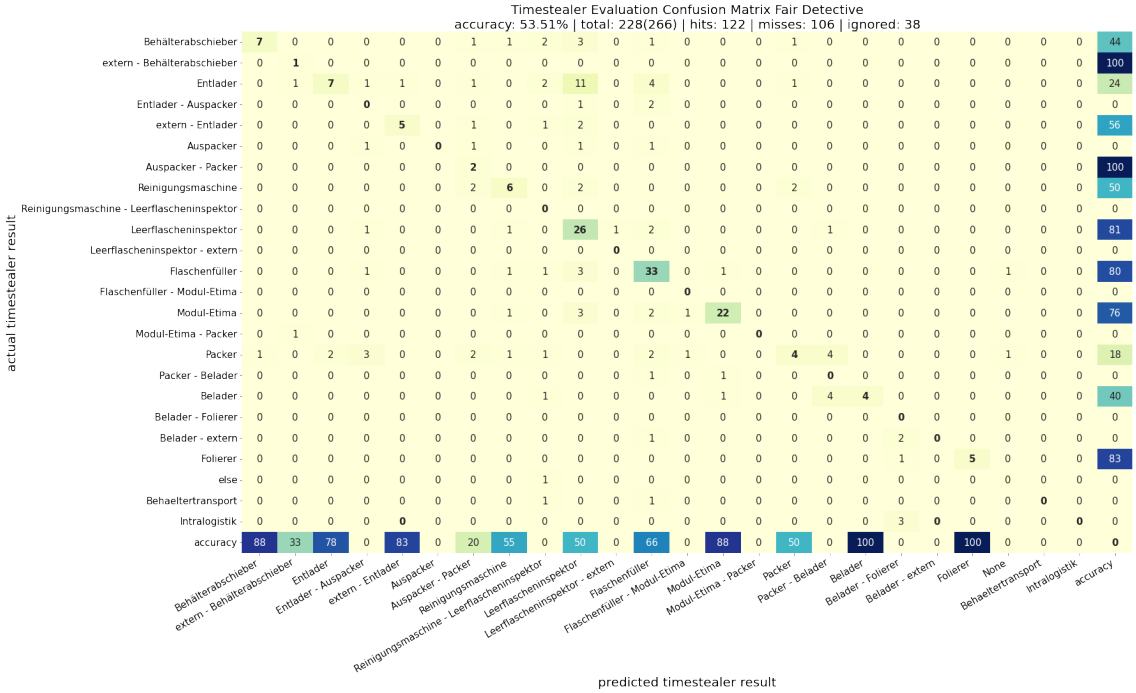


Figure 4.4 *Confusion matrix for rule based algorithm at customer-a. Labels and results of the form "m_1 - m_2" are to be read as "causer is located between "m_1 and m_2". Low accuracy since interpretation of stoppages between customer and algorithm heavily deviates.*

classifications. During discussions with customer-a's expert, it turned out that he accepts the results of the algorithm as also valid results for most examples in question. Thus, we can effectively assume a significantly higher effective accuracy of up to 83.0% for this customer.

At customer-b, a different picture appears for the classification results. With a classification accuracy of 92.1%, the rule based approach reached unexpectedly good results. Again taking a deeper look into reasons for miss-classifications shows that 4.0% of the data was wrongly classified by the algorithm, in only 2.5% of cases the interpretation of the situation was different between expert and algorithm and a very low ratio of 1.4% were caused be erroneous data. In contrast to the labeling process at customer-a, the expert at customer-b did his analysis based on the data with seeing the algorithms result. In this way, he on the one hand had the same knowledge-input for each downtime as the algorithm also had, and on the other hand he adapted his interpretation of stoppages to the algorithms approach. Thus the ratio of errors arising from different analysis-approaches is very small at this

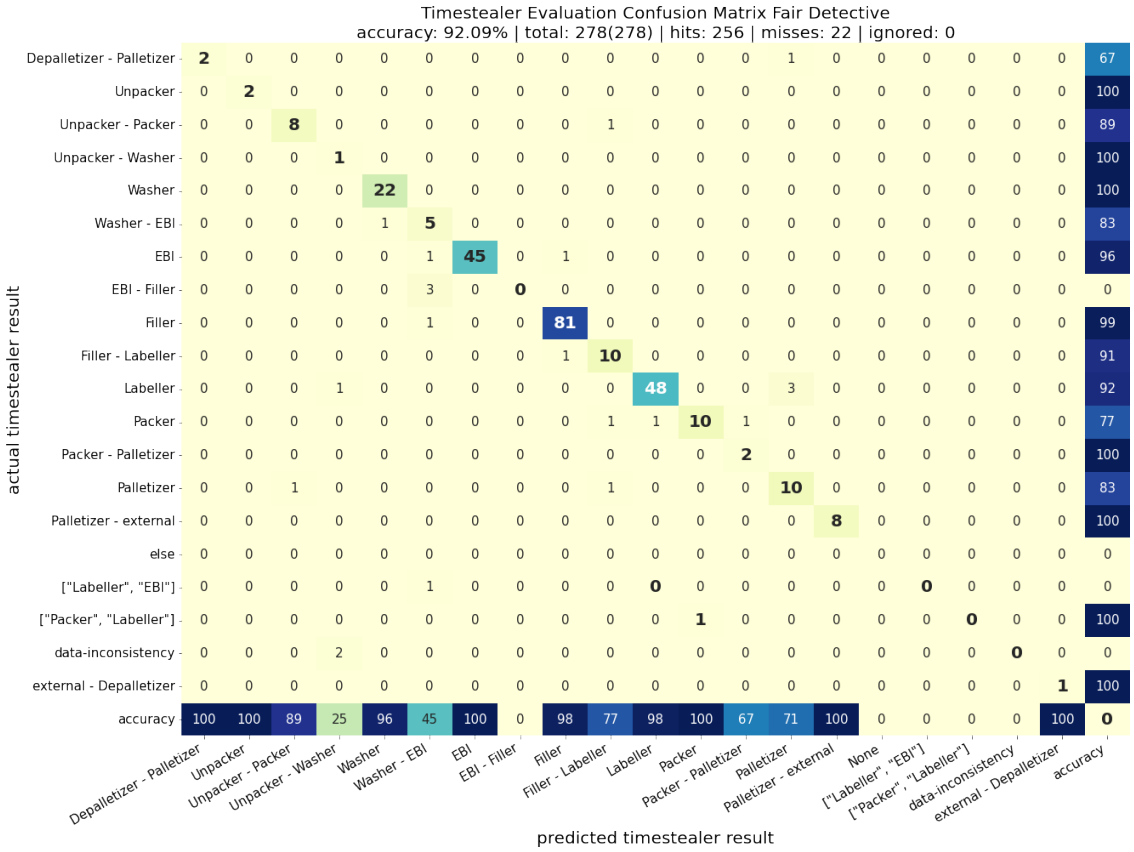


Figure 4.5 *Confusion matrix for rule based algorithm at customer-b. Labels and results of the form "m_1 - m_2" are to be read as "causer is located between "m_1 and m_2". Very good result due to reworking data output of multiple machines and highly consistent interpretation approaches between customer and algorithm.*

customer. A further difference between these customers is the data-quality. Before recording the data at customer-b, severe effort was done to improve the machines data-output. This gets reflected in the corresponding ratio of caused classification errors which are significantly smaller at customer-b. Contrary, the ratio of errors caused by the algorithm itself was quite similar with 4.0% to customer-a's 5.2%. To put things into a nutshell, the newly developed rule-based algorithm provides results with high accuracy for the case of good data-quality. Additionally, synchronizing the users understanding of root-cause detection with the approach of the algorithm has a huge potential. We showed this potential at customer-a: When comparing the algorithmic results to the customer's own estimations an accuracy of about 50% resulted. Contrarily, when asking the customer if he agrees to the result of the

algorithm he agrees for about 80% of the examples.

Overall, the new rule based algorithm shows multiple improvements in direct comparison to the Avalanche algorithm: First of all, its accuracies are in average 13% higher than those of the Avalanche algorithm. One factor for this improvement is the ability of resolving error propagations via auxiliary streams by using the graph structure of a line. Additionally, since its basic idea imitates the common approach of experts when searching for the root cause machine, the results of Fair Detective are always intuitive for experts. Even in case of a wrong classification the expert can understand why this result occurred. Thus acceptance of Fair Detective beyond experts is very good.

4.3 Dynamic Time Warping and k-Nearest Neighbors

In this section, we first of all optimize parameters, afterwards evaluate classification results per customer, and finally discuss if inference durations are sufficient. We use manually labeled data for training and evaluations. Table 4.3 provides the amounts of label-example pairs per customer and per class.

customer a		customer b		customer h		customer w	
Depalletizer	72	Depalletizer	22	Depalletizer	16	**Blower**	116
Unpacker	27	Unpacker	26	Unpacker	0	Labeler	18
Washer	39	Washer	45	Sekamat 2	4	Filler	40
EBI	96	EBI	76	Sekamat 1	0	Packer	23
Filler	127	**Filler**	181	Washer	9	Palletizer	5
Labeler	85	Labeler	153	EBI	91		
Packer	51	Packer	50	**Filler**	94		
Palletizer	32	Palletizer	58	Labeler	69		
Cargo Safeguarding	0			Packer	73		
Wrapper	10			Palletizer	65		
				Crate Washer	1		
				Sweep-off Depalletizer	6		
$\sum$	539		611		428		202

Table 4.3 *Available manually labeled examples per customer for DTW and k-NN training. Overall number and amount per class(root-causer) are given. Lead-machine is written in bold letters.*

These are split into 100 test-examples and the remaining ones for training.

4.3.1 Optimal parametrization

First of all, parameters Δ_{slice} and the number of neighbors k of k-Nearest Neighbors are sweeped on customer-b data. Customer b is a reasonable choice since its line-characteristic represents an average between all four customers. For k, values 2, 3, 4, 5, 6 and 8, and for Δ_{slice} the parametrizations 10s, 20s, 30s, 60s, 120s, 240s are evaluated over five repetitions each.

For each run, train/test split is done randomly before the run. This results in mean accuracies like displayed in 4.4.

Δ_{slice} \ k	2	3	4	5	6	8
10	60.2 ± 3.2	69.4 ± 3.1	73.0 ± 3.6	69.6 ± 3.2	70.4 ± 3.2	71.2 ± 1.6
20	63.2 ± 2.6	71.6 ± 2.9	72.4 ± 2.6	75.8 ± 5.5	75.8 ± 3.7	72.0 ± 1.1
30	65.8 ± 4.9	68.2 ± 2.1	74.4 ± 3.8	71.8 ± 1.9	75.8 ± 4.5	74.8 ± 4.0
60	66.8 ± 2.5	72.0 ± 2.1	70.8 ± 5.6	75.6 ± 3.7	71.4 ± 4.6	76.2 ± 3.2
120	67.8 ± 2.2	71.8 ± 2.9	72.0 ± 5.6	74.4 ± 3.6	74.4 ± 2.2	74.2 ± 5.0
240	62.8 ± 5.4	68.0 ± 2.1	70.4 ± 2.9	71.6 ± 6.3	69.4 ± 2.2	75.0 ± 3.0

Table 4.4 *Mean classification accuracies over five runs for dataset of customer-b using DTW and k-NN with hyperparameter-sweep over k of k-NN classifier and Δ_{slice} (subsection 3.2.1).*

Accuracies show a significant growth along k-values until k=5. When choosing k as 5, 6 or 8 and Δ_{slice} as 30s, 60s or 120s equally good results near the maximum reachable ones are achieved. With growing k usually overfitting is prevented, since multiple training-examples have to be found near the example that should be predicted. In this way, the model generalizes its prediction policy. On the other hand, choosing a larger k means prefering classes that are represented over-proportionally often in the training set. As an extreme example: If k is chosen equally to the size of the training-set, always the class with most examples will be predicted. Since multiple classes consist of only a few samples (≈ 10) in our datasets (compare Table 4.3) it is favourable to choose a small k-value to also regard those classes. Thus we decide to choose k=5 and Δ_{slice}=60s. This will be used for the upcoming evaluation.

4.3.2 Results per customer

An overview over the mean accuracies of each customer is given in Table 4.5.

We can see quite large dependencies of the accuracy to the customer. On the one

	test accuracy [%]
customer a	67.7 ± 2.1
customer b	70.0 ± 2.8
customer h	63.3 ± 3.9
customer w	87.0 ± 2.8

Table 4.5 *Mean classification accuracies and standard deviation over three runs for each customer using DTW and k-NN*

hand, the complexity of the line is an important factor. The more machines a line consists of, the more classes the classifier has to resolve. Additionally, the patterns in the data are more complex and have to be found over a larger local and temporal space. Thus, it is reasonable that customer h, as the largest line, has the lowest accuracy while customer w, as the simplest line, has the highest accuracy.

In the following, we are going to look at the confusion matrices of for each customer. These are created by comparing the test-sets labels to the models predictions. For customer-a (Figure 4.6), we can see the main diagonal(correct classifications) being better occupied for the central machines than for the outer ones. Especially the first part of the line (Depalletizer to Washer), seems to be hard to resolve for the algorithm. This can result from greater variance in temporal error propagation due to the longer spatial propagation path for machines at the very start or end of the line.

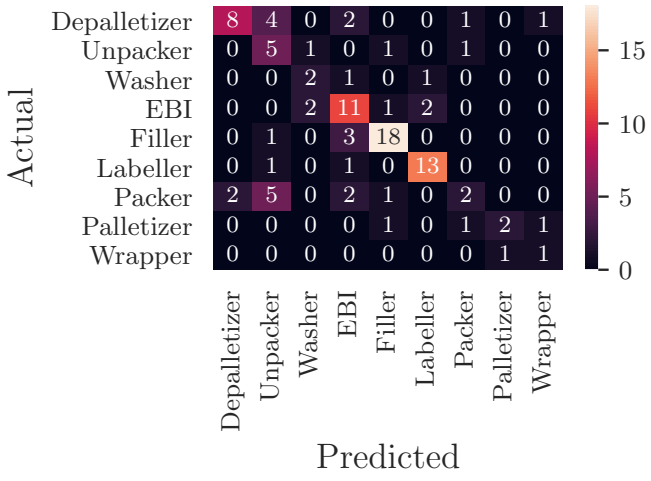


Figure 4.6 *Confusion matix for DTW and k-NN on data of customer a.*

Customer-a (Figure 4.6) as well as customer-b (Figure 4.7) have the filler as lead-machine. And in both cases the accuracy of detecting the lead-machine correctly is not high, although this should be the easiest class to predict since the lead-machine is the causer itself if it stopped as a result of an own-fault.

The following is an example for multiple wrong classifications in case of own-

faults at the lead-machine: At customer-b the Labeler is chosen as causer by the algorithm six times although the Filler itself was chosen as causer by the expert. Most probably, this is a result from DTW equally weighting all inputs since it is a similarity measure and not a learning algorithm. Thus it is not able to emphasize the importance of the lead-machine's state like any expert intuitively does. This results in classification results that feel counter-intuitive for experts. And missing the classifications that seem to be easy from an expert's view quickly diminish the trust in the algorithm and should be avoided at all costs.

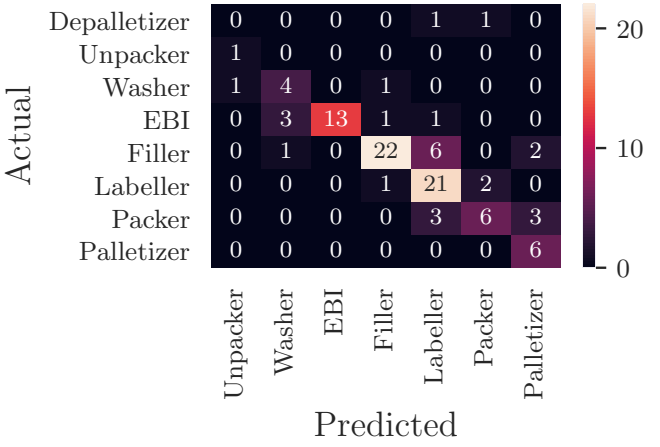


Figure 4.7 *Confusion matix for DTW and k-NN on data of customer b.*

Confusion matrix of customer-h (Figure 4.8) looks unfamiliar since the class distribution in this dataset is very uneven (compare Table 4.3). Most examples are representing the area from EBI to Palletizer. Also in this case, the Filler is the lead-machine and especially resolving EBI and Filler as root cause is a hard task for the algorithm. As mentioned in section 4.1, EBI and Filler are electronically connected. That leads to missleading machine-states for some cases and makes it hard to interprete the status of EBI for an algorithm.

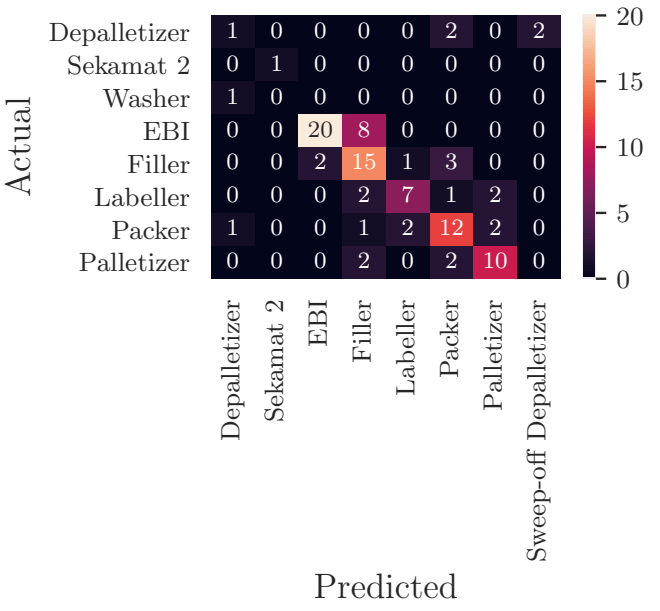


Figure 4.8 *Confusion matix for DTW and k-NN on data of customer h.*

In the case of customer-w (Figure 4.9), Palletizer is not predicted once. This matches with the problem of choosing k too large like mentioned in subsection 4.3.1. In case of Palletizer at customer-w we only have 5 examples in the dataset. Thus, it is very unlikely to find a spot in the space of input data which is close to more than one of these 5 sample-points.

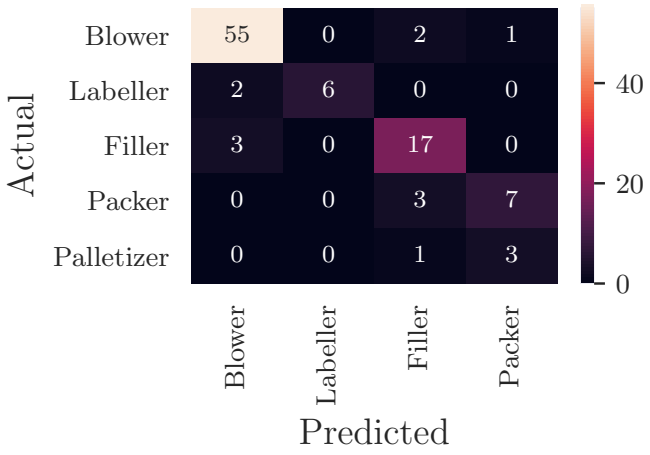


Figure 4.9 *Confusion matix for DTW and k-NN on data of customer w.*

All in all DTW and k-NN reaches solid results but, since it is not able to stronger weight important regions of the data like the lead-machine's state during its stoppage, consistently miss-classifications that are not explainable towards a line expert come up.

4.3.3 Inference durations

For a statistic over detected root causers, usually time spans starting from one shift (8h) to a week are selected. With roughly 20 to 50 stoppages per shift, this results in hundreds of root-cause analyses for one statistical view. And latency should not exceed a few seconds. That is a demanding requirement for the computational duration of the algorithm.

Δ_{slice}	inference duration [s]
10	3.9 ± 0.2
20	1.8 ± 0.13
30	1.2 ± 0.08
60	0.6 ± 0.04
120	0.3 ± 0.03
240	0.2 ± 0.01

Table 4.6 *Mean duration with standard deviation for one inference of DTW and k-NN dependent on* Δ_{slice}

Thus, we compute the mean duration per inference (classifying one example) dependend on Δ_{slice} since this changes the number of temporal steps in each sample-point and therefore heavily affects the processing efford inside DTW. In the results of Figure 4.10, we can see the exponential dependency of computation durations to Δ_{slice}.

As expected (compare subsection 2.3.2), dura-

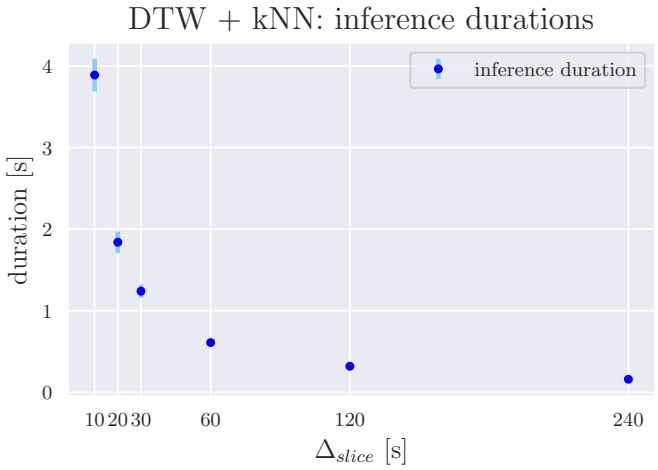


Figure 4.10 *Average and standard deviation of duration per single inference dependent on Δ_{slice} in preprocessing which is directly dependent to the number of temporal steps in input-data. Statistics are computed over five repetitions.*

tions are very high even for the largest $\Delta_{slice} = 240$s with 0.16s (compare also Table 4.6). Assumed a batch of 100 line-stoppages this results in a computation duration of 16s even without respecting loading times from the database and further environmental tasks like api requests. Thus, DTW+k-NN is not suited for usage in a productive environment since latencies of this magnitude are not acceptable. Most probably, significant speedups in comparison to the used implementation are possible but it is also highly probable that these will not be able to solve the problem satisfactory.

With this result, we are able to eliminate DTW+k-NN from the list of candidates for usage in a productive system. Contrary to the current, the upcoming approaches are known for sufficient speed of their predictions. Thus, their computation durations will not be evaluated in this work but have to be checked in detail, in case one of these approaches should be used in the productive environment.

4.4 Random Forest

When training a random forest classifier, data is preprocessed using Manual Feature Extraction (subsection 3.2.2). The distribution of labeled examples beyond customers and classes can be seen in Table 4.7. Due to a minimum amount of 160 examples in

customer a		customer b		customer h		customer w	
Depalletizer	76	Depalletizer	22	Depalletizer	7	**Blower**	108
Unpacker	27	Unpacker	26	Unpacker	0	Labeler	18
Washer	39	Washer	46	Sekamat 2	0	Filler	39
EBI	87	EBI	76	Sekamat 1	0	Packer	22
Filler	110	**Filler**	181	Washer	1	Palletizer	5
Labeler	82	Labeler	145	EBI	43		
Packer	48	Packer	51	**Filler**	34		
Palletizer	25	Palletizer	58	Labeler	19		
Cargo Safeguarding	0			Packer	24		
Wrapper	10			Palletizer	29		
				Crate Washer	0		
				Sweep-off Depalletizer	3		
$\sum$	504		605		160		192

Table 4.7 *Available manually labeled examples per customer for random forest training. Overall number and amount per class(root-causer) are given. Lead-machine is written in bold letters.*

the dataset of customer-h we separate the data into 80 test-examples and use the remaining ones for training.

The resulting train-accuracies are close to 100% for all customers except customer-a, while results close to 100% are usual for a Random Forest. The worse ability of learning customer-a's data can be explained by the fact that labels of customer-a were provided by experts from inside the filling and packaging line without looking at the corresponding data

	train accuracy [%]	test accuracy [%]
customer a	92.2 ± 0.7	72.8 ± 4.0
customer b	99.0 ± 0.3	90.7 ± 1.6
customer h	100.0 ± 0.0	82.5 ± 5.0
customer w	100.0 ± 0.0	93.9 ± 1.3

Table 4.8 *Accuracy of test- and train-set over ten runs for each customer using random forest (500 trees with maximal depth of 7)*

(compare section 4.1), whereas all other labels were provided from experts by taking data to the aid. Thus the labels of customer-a are likely to contain information that is not part of the data and consequently the model can not be able to learn the corresponding dependencies.

All test-accuracies are significantly higher than those of DTW+kNN. Especially at customer-b and customer-h, the performances were increased by a huge amount of about 20%. For the other two lines the increase is approximately 6%.

Comparing the quality of the results to each other, the same picture as in the last section arises: Customer-w seems to be the easiest task to solve, followed by customer-b. Customer-h is the second hardest task to learn and customer-a the most difficult.

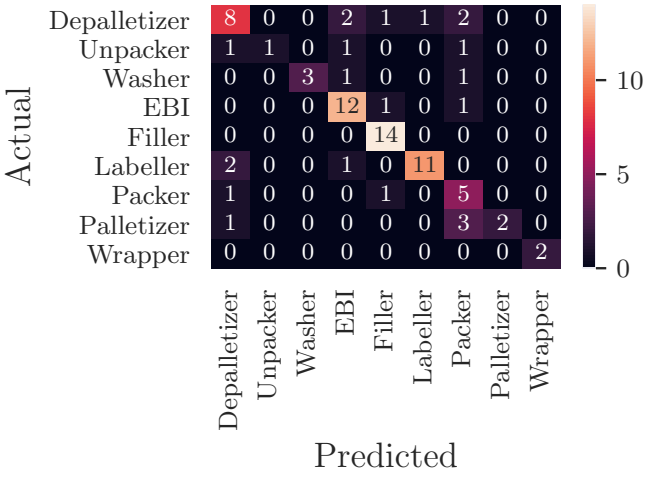


Figure 4.11 *Confusion matix for random forest on data of customer a. High accuracy when classifying the central machines (Washer to Labeler). In the outer area of the line classification seems to be more difficult.*

When looking at results of each single customer more closely, we can mainly see two problems that cannot be solved properly by Random Forest model at customer-a (compare Figure 4.11). On the one hand, the Depalettizer is severely confused with various other classes. On the other hand, the model confuses Packer and Palettizer by predicting the Packer as root causing machine for examples that are labeled as corresponding to Palettizer.

A similar picture emerges for customer-b (compare Figure 4.12): Also in this case, distinction of Depalletizer's class to the rest provides significantly worse results than for the remaining classes. This time also three stoppages that are labeled as "caused by the Palettizer" are predicted by the model as related to the Labeler. Both cases are concerning the first and last machines in the line and thus the error propagation between the causer and the lead-machine happens locally as well as temporally on the longest path.

On the one hand, the local error propagation can differ by "using" the secondary streams like empty crate or empty palette conveyors. On the other hand, the temporal differences of error propagation between the incidents grow with the distance between causer machine and lead-machine. E.g.: If the Palettizer has already had a problem which made it stop and the filling and packaging line currently is restarting when again a problem at the Palettizer occurs, then the temporal error propagation through the line is very fast since all conveyors still are full. In contrast to that, if the buffers that are positioned downstream of the lead-machine have been empty (due to an error caused by the lead-machine itself or any machine upstream of it) the buffers can gather thousands of bottles in case of a Palettizer problem and thus it will take up to 15 or 20 minutes until the lead-machine has to stop.

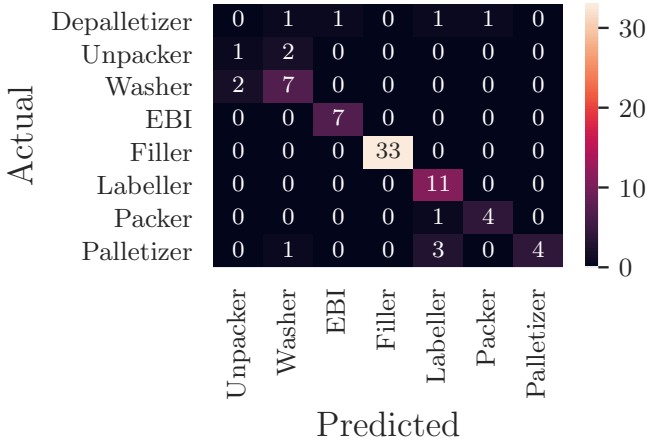


Figure 4.12 *Confusion matix for random forest on data of customer b. Most miss-classifications are connected to machines that handle crates or pallets. This can be caused by the additional complexity introduced by branch-streams (empty crate/pallet transport) between these machines.*

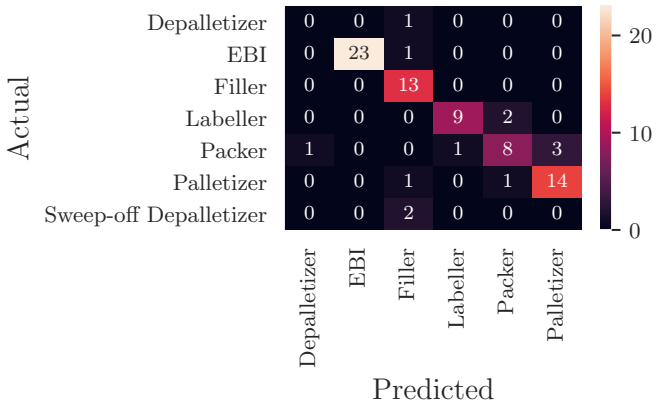


Figure 4.13 *Confusion matix for random forest on data of customer h.*

In customer-h's confusion matrix (compare Figure 4.13), most wrong classifications are positioned on the secondary diagonal. Thus, the classification result only missed by one machine (along the main stream). This is to be rated as positive, since almost all classifications showed the correct region for the error cause. At customer-w, the classification accuracy is very high (compare Figure 4.14). Still, Palettizer

was never chosen as the root causer by the algorithm. Correspondingly, one has to note Palettizer being only represented five times in the dataset and only three times in the training set (compare Table 4.7). Although Random Forest does a good job in learning underrepresented classes, this level of imbalance seems to be too high.

In contrast to DTW + kNN, this approach shows significantly increased accuracies for the machines being positioned near the lead-machine. The Random Forest classifier shows to solve the simpler examples with very high accuracy. But also for the outer machines, the classifications are good. That leads to results ranging from sufficient accuracy in case of the hard task at customer-a to high accuracy at customer-w. Compared to results of previous studies (section 2.5), these results are at least on state-of-the-art level and likely higher. Thus, Manual Feature Extraction seems to extract meaningful features which can be learned effectively by a comparably basic classifier like Random Forest.

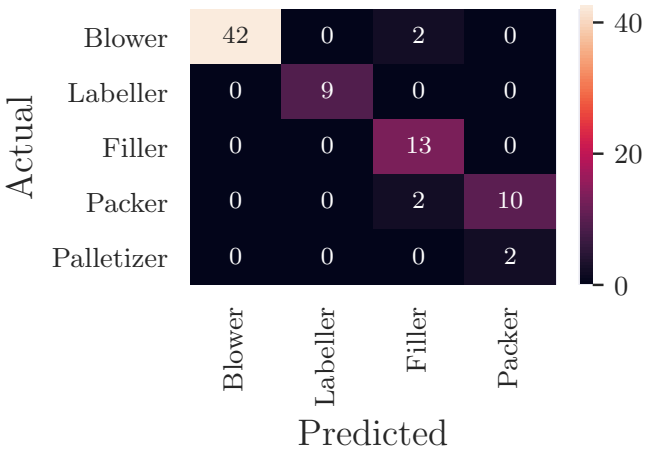


Figure 4.14 *Confusion matix for random forest on data of customer w. Palletizer is missing in the predictions.*

Since this approach provides the highest accuracies so far, it will be used as algorithmic labeler for building up larger datasets for Neural Network training (section 3.7).

4.5 Relational Graph Convolutional Neural Network

In this section, we will discuss the training of a fully convolutional Relational Graph Neural Network (RGCN) architecture on algorithmically labeled data on the one hand, and manually labeled data on the other hand. For this Neural Network, data is preprocessed by using manual feature extraction (subsection 3.2.2) like also done for Random Forest. The Neural Network architectures were built up in such way

that one model can afterwards be used to classify the root causer on any filling and packaging line with the same set of trained parameters. The only configuration needed, is the graph-representation of each filling line. This is the basis to firstly use Multi-Task Learning to fit the model to a preferably wide range of different lines simultaneously, and afterwards predict causer machines at lines that were not contained in the training set, by using Transfer Learning.

For training, validation and testing we used two different approaches. On the one hand, we go the usual way and simply divide all available expert labels up into train-, validation- and test-set. This results in some hundred labeled examples. For more detailed distribution compare Table 4.9.

	train set	validation set	test set
customer a	440	32	32
customer b	541	32	32
customer h	96	32	32
customer w	128	32	32

Table 4.9 *Available **manually labeled** examples per dataset and customer.)*

These expert labels are created with much effort and still the resulting datasets are comparably small for Neural Network training. Thus we use additionally built up a algorithmically labeled dataset (compare section 3.7) with about 9 times as many samples (compare Table 4.10).

	train set	validation set	test set
customer a	2429	160	160
customer b	2475	200	200
customer h	4603	60	60
customer w	1053	60	60

Table 4.10 *Available **algorithmically (Random Forest) labeled** examples per dataset and customer.)*

Distributions of labels at each customer are similar to those given in Table 4.7. Thus this dependency is not again displayed here.

4.5.1 Training on single lines using RGCN

In the following, the results of training one RGCN per customer (in contrast to Multi-Task Learning) are evaluated. First of all, manually labeled datasets are used. Despite the comparably small size of training sets for a Neural Network, results show similar accuracies compared to Random Forest (compare Table 4.11).

While customer-a and -b achieve around 3% less accuracy than Random Forest, customer-h provides about 3% higher accuracy and customer-w is similar. But on

	train accuracy	test accuracy
customer-a	78.4 ± 4.9	69.4 ± 9.4
customer-b	92.9 ± 2.2	86.9 ± 3.1
customer-h	91.7 ± 2.5	85.6 ± 3.2
customer-w	95.6 ± 2.1	92.5 ± 2.5

Table 4.11 *Mean train- and test- accuracy with standard deviation computed over five repetitions for RGCN model trained on* ***manually labeled*** *data per customer*

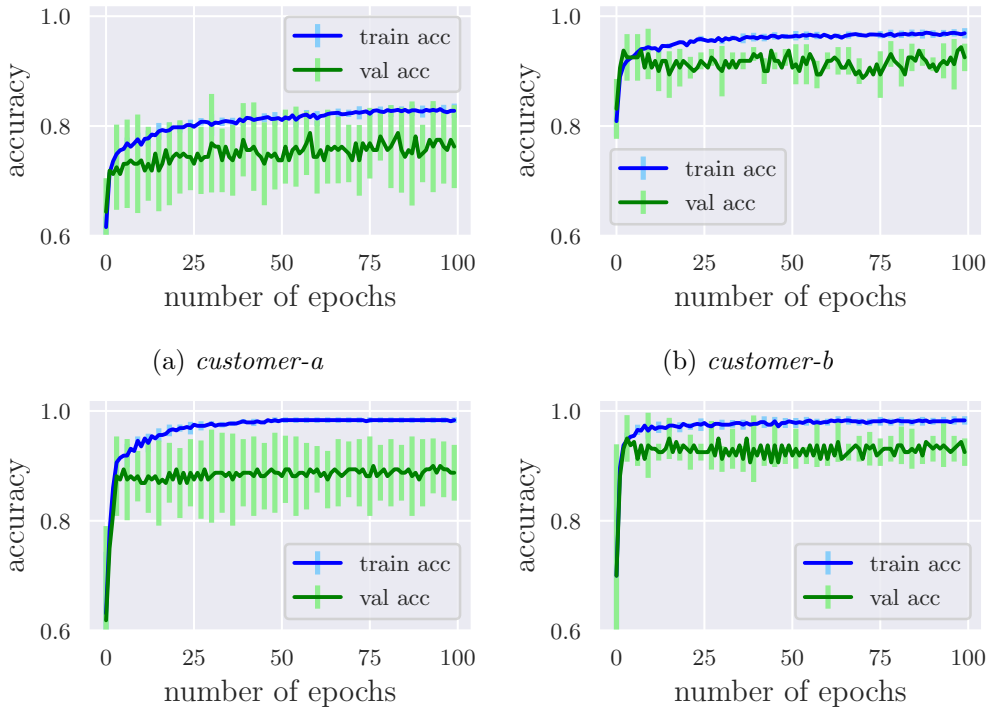
the other hand, the standard deviation in accuracies over multiple training runs is really high. Especially for customer-a, with a statistical error of almost 10%, results seem to be highly dependent on the training run. But most probably, the point that raises variance most, is the size of the test-set. With only 32 examples, the sample distribution inside this dataset is far less stable than a larger set due to probabilistic train/test-splitting. And since one test example contributes about 3% of accuracy the test-accuracy is highly volatile. If there are e.g. two additional examples of a machine that is simple to classify correctly, like the lead-machine, instead of a difficult one, we can expect the test accuracy to change by 3% or 6%. Based on this knowledge, the variances at customers b, h and w are not too large.

	train accuracy	test accuracy (at best validation acc)	test accuracy (final)
customer-a	95.1 ± 1.2	82.8 ± 1.0	83.0 ± 1.2
customer-b	96.9 ± 0.4	93.4 ± 0.6	92.7 ± 1.8
customer-h	91.4 ± 0.4	86.0 ± 1.3	85.7 ± 0.8
customer-w	97.4 ± 0.4	83.3 ± 3.3	88.3 ± 0.0

Table 4.12 *Mean train- and test- accuracy with standard deviation computed over five repetitions for RGCN model trained on* ***algorithmically*** *(Random Forest)* ***labeled*** *data per customer*

Comparing results of training the same Neural Network on algorithmically labeled data to those resulting from training on manually labeled ones, the mean test accuracies are significantly higher for customer-a and -b. Customer-h shows very

similar results in both cases while customer-w is significantly worse. The result at customer-w was not to expect, since the Random Forest model, which was used for labeling, achieved a test-accuracy of 96% and also the RGCN-model shows a mean train accuracy of 97.4%. This means that is was able to learn, what the Random Forest model itself had learned before, almost perfectly.



(a) *customer-a* (b) *customer-b*

(c) *customer-h* (d) *customer-w*

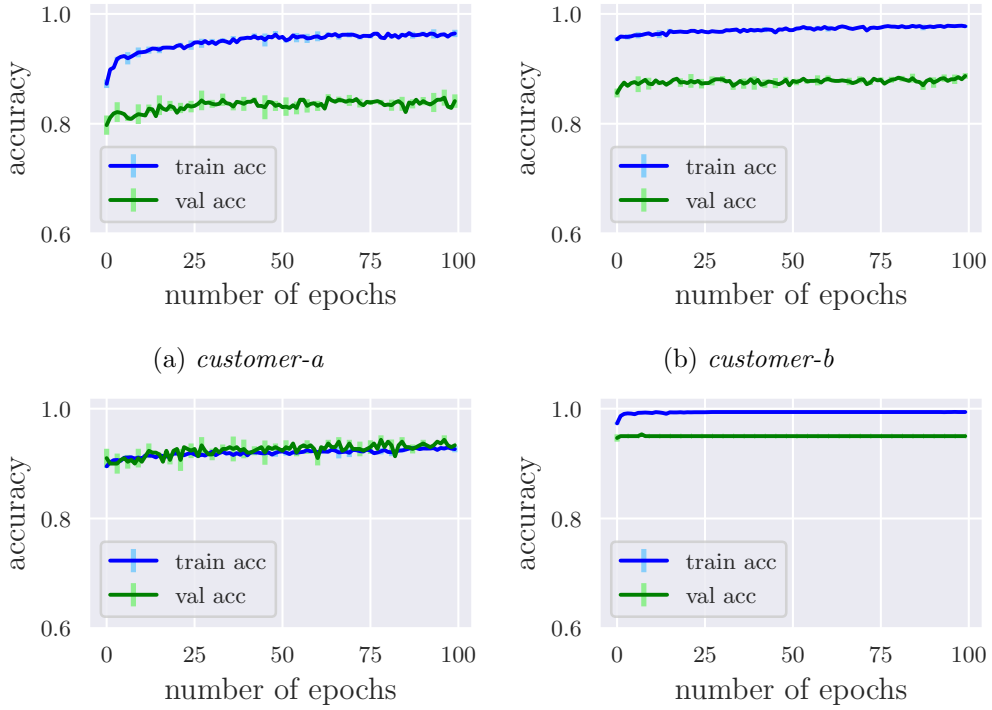
Figure 4.15 *Training and validation accuracy for RGCN on each customers manually labeled data. Mean values with standard deviation as error bar are computed over five repetitions.*

Looking deeper into detail, test accuracy after finishing the training (100 epochs) is with 88.3% significantly higher than after the epoch with best validation accuracy (83.3%) for customer-w (compare Table 4.12). In this case, the models' ability of generalization is increasing during the training although the validation accuracy starts to decrease after a few epochs. Usually validation- and test-accuracy show similar values and especially tendencies. This is why validation is used to determine e.g. overfitting.

In our case, the difference between validation- and test-results are probably connected with the way of building up these datasets. Since train- and validation-dataset

are created by algorithmic labeling, the validation set is sampled from Random Forest labeled examples while test-set consists of manually labeled examples. This can result in slightly different distribution of data in the datasets and thus explains the decreased ability of generalization for one of four customers.

Since Neural Networks in general possess a huge number of learnable parameters, they are prone to overfitting. Over the years, many mechanisms have been invented to decrease this tendency. But independently of the type of ML-model, every model learns better generalized patterns or decision processes by having a broader, better distributed and thus usually also quantitatively larger dataset.



(a) *customer-a* (b) *customer-b*

(c) *customer-h* (d) *customer-w*

Figure 4.16 *Training and validation accuracy for RGCN on each customers Random Forest labeled data. Mean values with standard deviation as error bar are computed over five repetitions.*

We can witness this effect by comparing training progress when using manually labeled data (compare Figure 4.15) with the progress when using of algorithmically labeled(compare Figure 4.16) data which possesses about nine-fold amount of training-examples.

Validation accuracy on manually labeled data shows very high results after the first few epochs and afterwards does not further improve (except of customer-a). Instead it decreases slightly, especially for customer-b and -w. Additionally, the shapes of training- and validation-results are different. While training-accuracy shows the expected saturation behavior, validation-accuracy follows the training-results for a few epochs but afterwards detaches and stays at the same level or slightly under it. Starting from the mentioned knee in the validation curve one usually talks about overfitting: The model loses its ability of generalizing the learned patterns to unknown examples as a result of adapting too heavily to the training-set.

In the case of algorithmically labeled data, the model reaches its best results significantly later in the training process. Additionally, validation- and training-graphs possess very similar shapes. Thus there is no overfitting when using the larger dataset.

4.5.2 Multi-Task Learning using RGCN

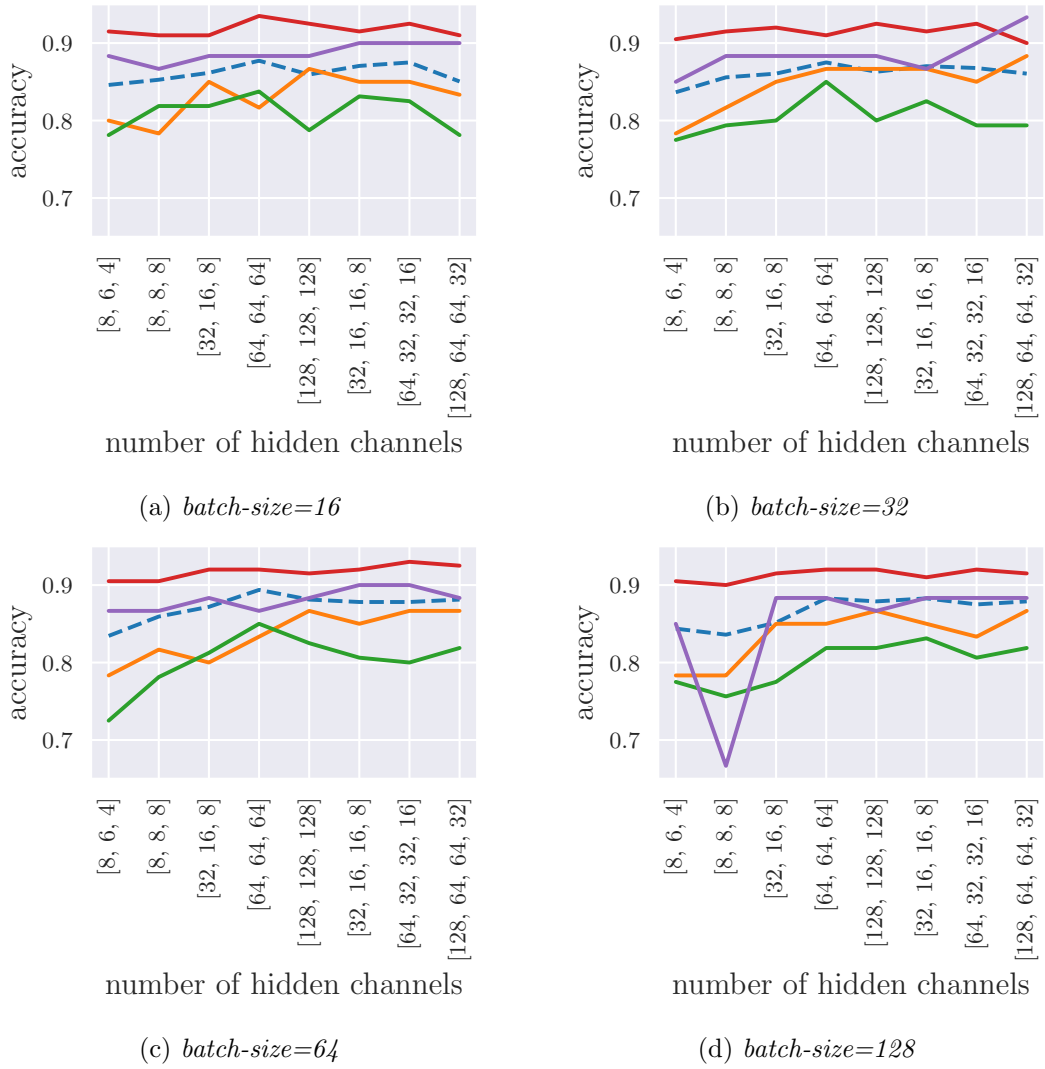
During this section, we train the Relational Graph Convolutional Neural Network on data of multiple filling and packaging lines at once. This way, we expect the model to learn a generalized ability of detecting the root-cause machine over different filling line architectures and characteristics. Here we exploit the ability of our built up networks: They are adaptable to any possible line by providing the related graph structure. In other words: Instead of training one Neural Network per customer we now train one Neural Network for all four customers.

In the following, we will compare Multi-Task Learning on algorithmically labeled data to manually labeled data, extract the best hyperparameters and compare the results to training one Neural Network per filling line. As before, test-accuracies are computed for each network when it shows its best validation-accuracy. But since we now train on different customers simultaneously, the output looks different. An accuracy for each customer is extracted and additionally the network is evaluated against a combined dataset over all customers. This average over the different filling and packaging lines is also used for all measures during training, e.g. calculating the best validation score.

Hyperparameter tuning

To determine hyperparamters for RGCN training, influence of batch-size and size of the Neural Network architecture are evaluated. We have run RGCN Multi Task training on algorithmically labeled data with Neural Network architectures ranging from small three-layer with only eight, six and four hidden dimensions to four-layer shapes with 128, 64, 64 and 32 latent dimensions. Every architecture was combined with batch-sizes 16, 32, 64 and 128. The results of this experiment are displayed in Figure 4.17.

All four plots show that larger networks, like the 4-layer architectures, lead to best performing results and best stability beyond the customers' accuracies. Thereby, neither varying the number of hidden channels, nor different batch-sizes do change the accuracies for 4-layer architectures significantly. At this point we set the batch-size to 64 and the hidden dimensions to 128 in the first, 64 in the second and third, and 32 in the fourth layer for upcoming trainings.



(a) *batch-size=16*

(b) *batch-size=32*

(c) *batch-size=64*

(d) *batch-size=128*

Figure 4.17 *Comparing test accuracies of multi-task training for different sizes of the RGCN architecture and different batch-sizes on data that was algorithmically labeled by Random Forest. Each combination is evaluated on a customer-specific test set and additionally on a overall test-set (average). The average accuracy over five runs is plotted in dashed blue, customer-h in orange, customer-w in lila, customer-a in green and customer-b in red.*

Manually- vs. algorithmically-labeled

Like done before, we evaluate the influence of using manually labeled data in comparison to algorithmically labeled data also for Multi-Task Learning performance. We use a four layer architecture with 128 hidden dimensions in the first layer, 64 in the second and third, and 32 in the fourth layer and train it five times. For every run, we extract the test-accuracy for the model after the epoch showing highest validation accuracy while training for 200 epochs. Afterwards, we compute the mean accuracy and standard deviation as statistical error for the average-accuracy and every customer over the set of five repetitions. Table 4.13 shows the results for training on algorithmically labeled data. The trained network reaches accuracies that are marginally beyond (customer-a and -b) or very similar to those of training the RGCN model on single customers (compare subsection 4.5.1).

	train accuracy [%]	test accuracy [%]
average	95.4 ± 0.8	87.0 ± 0.7
customer-a		81.3 ± 1.4
customer-b		91.3 ± 1.0
customer-h		86.3 ± 0.7
customer-w		88.3 ± 1.1

Table 4.13 *Mean train- and test- accuracy with standard deviation computed over five repetitions for RGCN model training on Random Forest labeled data using Multi Task Learning*

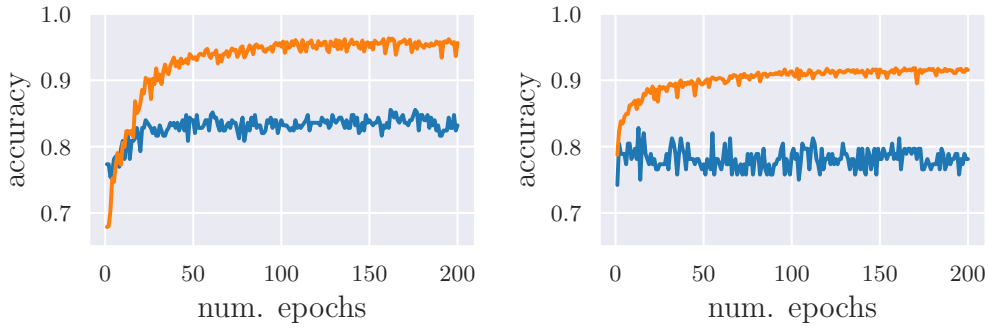
But contrary to the previous results, also for customer-w the maximum accuracy of about 88% is available at the epoch of best validation accuracy. Thus expressiveness of validation concerning the abilities of the model is given in this case. In case of a productive usage of a Neural Network, this correlation is very important to automatically determine the optimal stop of training progress for receiving the best possible parameters of the model.

Like in subsection 4.5.1, variances over multiple training runs are way (three to four times) larger for training on manually labeled data compared to training on algorithmically labeled data. Additionally, the achieved accuracies are worse in general.

Figure 4.18 also shows the same behavior of training- and validation-accuracy like in the previous chapter: For algorithmically labeled data train- and validation-curve

	train accuracy [%]	test accuracy [%]
average	91.5 ± 0.8	82.0 ± 2.4
customer-a		66.3 ± 8.8
customer-b		86.1 ± 4.2
customer-h		84.0 ± 5.8
customer-w		91.7 ± 4.2

Table 4.14 *Mean train- and test- accuracy with standard deviation computed over five repetitions for RGCN model training on manually labeled data using Multi Task Learning*



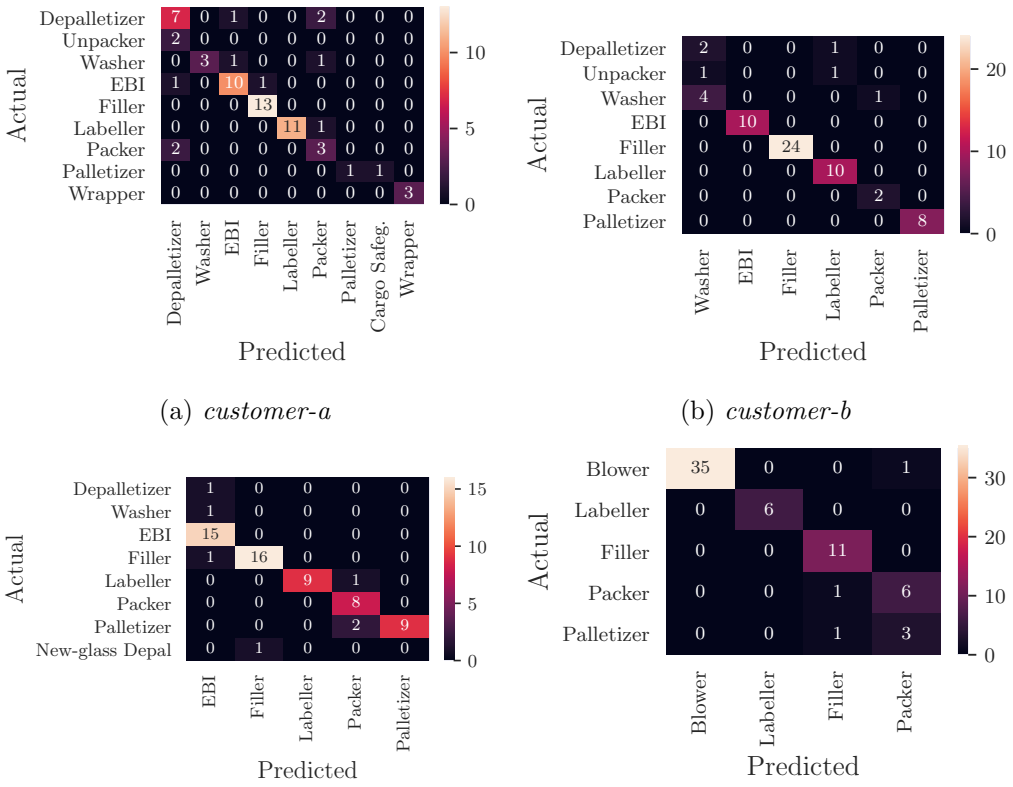
(a) *Random Forest labeled* (b) *manually labeled*

Figure 4.18 *Training (orange) and validation (blue) accuracies of a RGCN over 200 epochs*

show similar shapes while a early detaching of the validation-accuracies from training progress can be witnessed for manually labeled data.

Comparing the average test-accuracies, we can state an performance increase of 5% by using algorithmically-labeled data over manually labeled data!

Looking deeper into details of the training results, very structured confusion matrices for training on algorithmically labeled data (compare Figure 4.19) can be witnessed. The main diagonals, which are showing correctly classified results, are clearly pronounced. And many of the falsely classified examples appear next to the main diagonal and thus classify a neighboring machine of the correct one. This has to be considered a "good miss", since the rough area of the error cause still is correct. Especially in cases that show a superposition of error causes leading to a lead-machine stoppage, it is hard to choose exactly one causer machine. A frequent scenario is e.g. a fault at machine A that makes the neighboring machine B stop.



(a) *customer-a* (b) *customer-b*

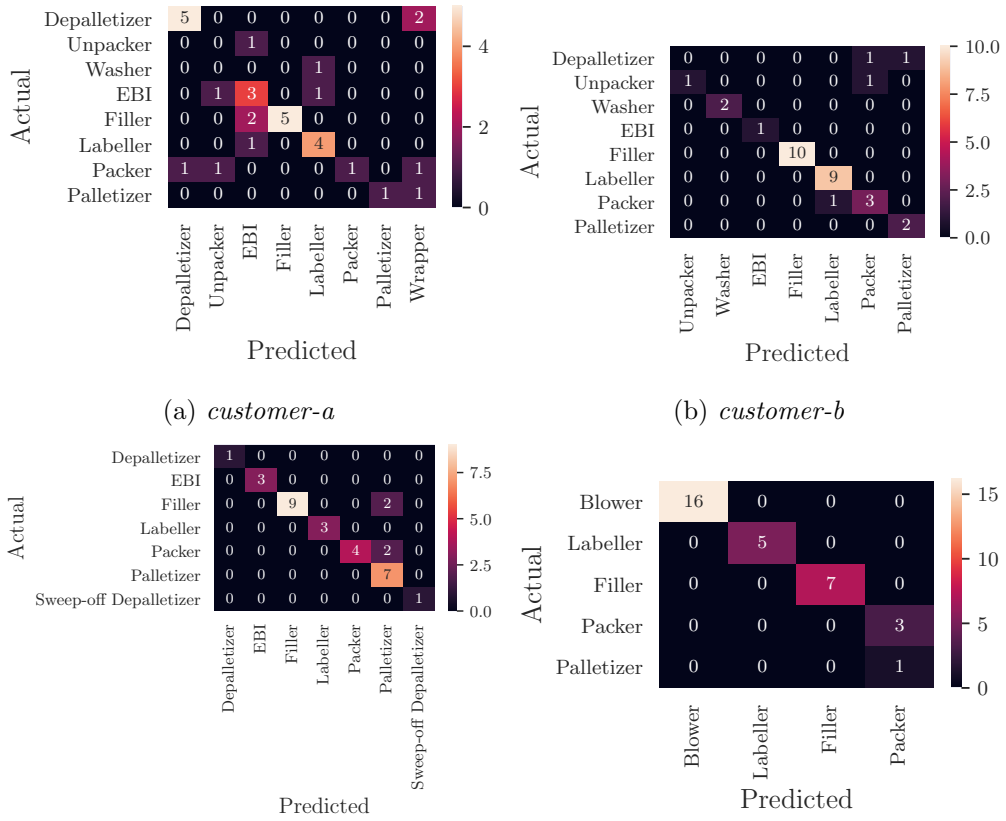
(c) *customer-h* (d) *customer-w*

Figure 4.19 *Confusion matrices for Multi-Task Learning on algorithmically labeled data. The model is trained on all customers' data at once and tested using test-sets of each customer individually.*

The operators use the time to fix some previously known but not severe issue at machine B. While they are still working on machine B, machine A already restarted and now has to stop caused by machine B. In this case, line experts usually choose the temporally first causer as the main causer since the line would not have stopped without this failure. But both machines contributed to the overall line downtime. In this case it would not be exactly wrong to classify machine B as causer. Thus, model predictions that miss the correct machine in favor of its neighbor have better chance of being reasonable than those that completely miss the correct area within the line.

Figure 4.19 additionally shows some machines getting ignored by the Neural Network. E.g. Unpacker and Cargo Safeguard at customer-a, Depalletizer and Unpacker at customer-b, Depalletizer, Washer and Sweep-off Depalletizer at customer-h and

Palletizer at customer-w. All of these have in common that they are severely underrepesented in the dataset. On the long term, that means with increasing amount of data and customers, it is to expect that the potential of the known graph structure is fully exploited, accordingly the generalization of the model increases and thus also the poorly represented classes can be correctly classified. But at this stage the model seems to lack some of the needed ability of generalization to expand its space of predictions to cover all input machines.



(a) *customer-a* (b) *customer-b*

(c) *customer-h* (d) *customer-w*

Figure 4.20 *Confusion matrices for Multi-Task Learning on manually labeled data. The model is trained on all customers' data at once and tested using test-sets of each customer individually.*

In contrast to the above, the confusion matrices related to manually labeled data show a significantly more confused picture, especially for customer-a. Obviously, it is very hard for the model to share the learned features of the other customers with those of customer-a, what, as seen before, decreases the classification accuracy

for this filling line by about 15%. In Multi Task Learning, the model will adapt to recognize those patterns that are most common beyond the tasks. Thus from a Multi Task Learning point of view, the task corresponding to customer-a (task-a) differs from the others in such degree, that training the network on all tasks at once influences the training result for task-a negatively.

Summarizing the given results, we encountered accuracies within Multi Task Learning that replicate the top level of the available approaches in this work. A great advantage when using this approach is that only one model has to be trained and monitored in its life-cycle. Additionally, we expect the model to increase its accuracy and ability of generalization by adding more customers and more examples to the training.

As training on manually labeled data again leads to inferior results compared to algorithmically labeled data, we will proceed the evaluation of Transfer Learning by only using the latter.

4.5.3 Transfer Learning using RGCN

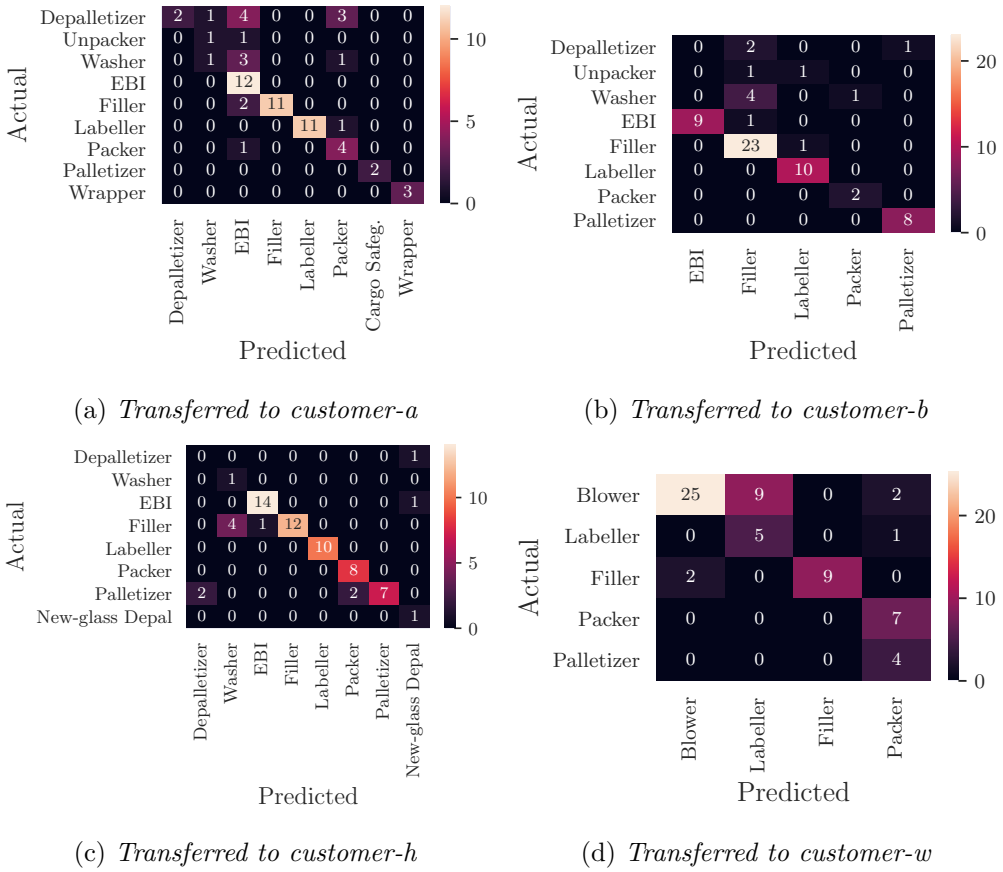
In this chapter, we use the models transferability over customers to train it on three of the four customers using Multi Task Learning, transfer the network to the remaining customer and evaluate its performance on the unknown filling and packaging line.

Since Multi Task Learning on the larger, algorithmically labeled dataset showed to be superior to training on manually labeled data, the combination of Multi Task Learning and Transfer Learning will be exclusively evaluated on algorithmically labeled data.

transferred to customer	average MTL test-accuracy [%]	transfer accuracy [%]
customer-a	89.4 ± 0.3	66.9 ± 2.5
customer-b	86.4 ± 1.1	80.9 ± 1.5
customer-h	86.2 ± 1.2	82.5 ± 2.3
customer-w	86.1 ± 1.1	70.3 ± 5.5

Table 4.15 *Accuracies with standard deviation for MTL on three customers and transferring the model to fourth customer. Computed over five repetitions for RGCN model training on algorithmically labeled data using Multi Task Learning*

When transferring the models, we achieve accuracies that are between 4% and 18% lower than in Multi-Task Learning when using all four customers. This wide range



(a) *Transferred to customer-a*

(b) *Transferred to customer-b*

(c) *Transferred to customer-h*

(d) *Transferred to customer-w*

Figure 4.21 *Confusion matrices for Transfer Learning using RGCN, trained with Multi-Task Learning on three customers and transferred to the fourth.*

of performance decrease can be explained by a deeper look into the different filling line characteristics (compare section 6.1) on the one hand and the different ways of manually creating a label on the other hand. Customer-w, with 18% the most decreased one, shows a clearly different line architecture than all other customers since here one-way plastic bottles are filled instead of returnable glass bottles. This result shows that the introduced Neural Network architecture learns generalized feature extraction that is able to solve severely different tasks with solid accuracy. But patterns, learned on one type of filling and packaging lines, are obviously not fully transferable to other types of lines. Customer-a, as a similar filling line compared to customers b and h, shows results decreased by about 14% compared to Multi Task

Learning. This also is a greater gap than expected, but can be explained by the way of creating the expert labels (manual labels). Customer-a is the only one that gave the ground truth for every stoppage from within the filling line without considering the corresponding data, that we used as input for our algorithms. The labels for all other customers were mainly created on basis of the data by analyzing plots of data for each lead-machine downtime. Comparing the results of customer-b and -h, as the customers corresponding to the two most similar classification tasks, to their performance in plain Multi Task Learning, their loss of accuracy is comparably small with 10% and 4%.

As expected, Transfer Learning works the better, the higher the similarity between source and target task are.

Looking deeper into detail, all confusion matrices of Transfer Learning (compare Figure 4.21) show problems of the models to solve the root cause allocation for the start of the filling line.

4.6 spatio-temporal Relational Graph Convolutional Neural Network

In this section, we train the spatio-temporal Graph Convolutional Neural Network which is built up in a fully convolutional architecture (like given in section 3.10) with data that was preprocessed using Temporal Data Downsampling (like described in subsection 3.2.1). The labels of the training data are built up algorithmically (compare section 3.7) while the model is evaluated on expert-labeled validation- and test-set to secure a realistic evaluation.

4.6.1 Training on single lines using stRGCN

First of all, we train the model on each of the filling and packaging lines individually. On the one-hand, we optimize hyperparameters for this model architecture and on the other hand we score the model to compare it to the previous approaches.

During hyperparamter-tuning we sweeped over different values for:

- Numbers of blocks and layers.
- Number of channels (this hyperparameter is used for all convolutions within the stRGCN architecture).

- Droput rate which is applied to the output of the graph convolution layers.

To say one thing in advance: The results are very stable against changing hyperparameters. In general, accuracies do not change significantly when tweaking the hyperparameters in a reasonable range. In-detail results can be found in Appendix (chapter 6).

When selecting the number of blocks and layers mainly extreme configurations should be avoided. But by using at least three layers per block stable results are achived (compare Figure 6.7). We chose to use three blocks of three layers each for our stRGCN architecture.

In terms of channels, again we mainly have to avoid choosing a insufficiently low number. For 16 or more channels the results do not change significantly (compare Figure 6.8). We will use 16 channels in the further work.

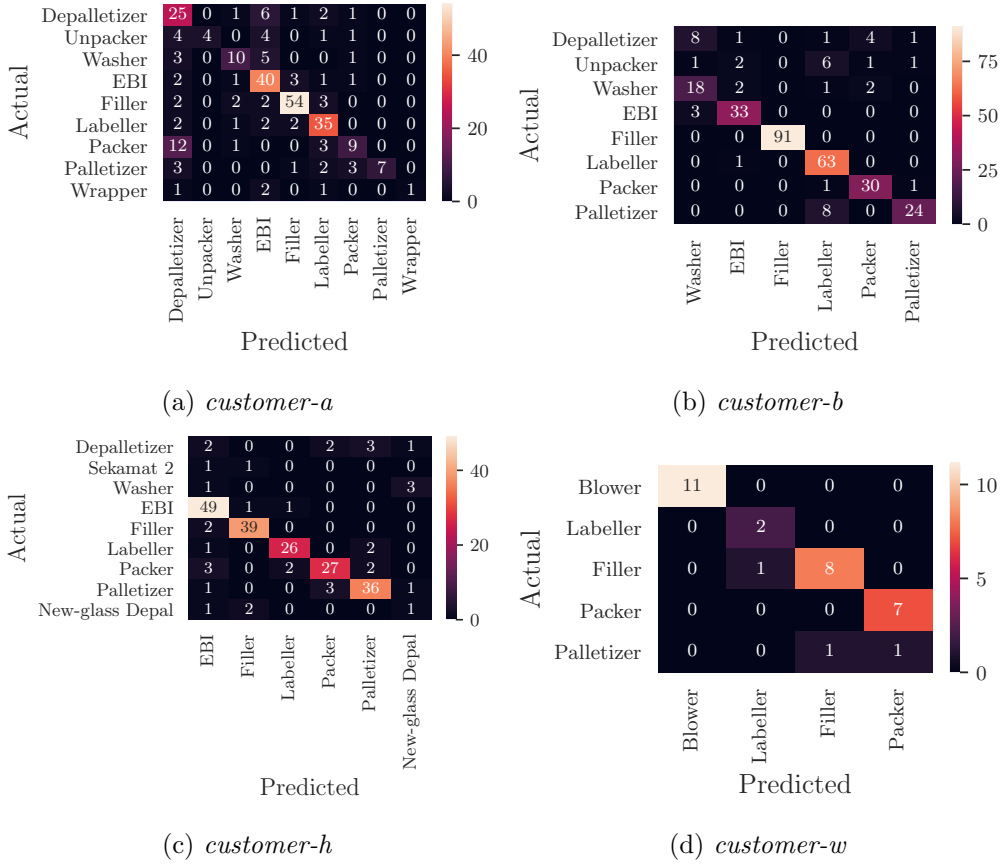
Finally, dropout rate does not influence the resulting accuracies significantly (compare Figure 6.9). It was set to 0.2.

With the chosen hyperparameters, we evaluate the models training on data of each customer individually. The results are displayed in Table 4.16.

	test acc [%]	train acc [%]
customer-a	67.2 ± 2.6	93.0 ± 0.4
customer-b	87.7 ± 1.4	95.9 ± 0.3
customer-h	79.7 ± 2.0	95.9 ± 0.2
customer-w	86.8 ± 6.0	98.2 ± 0.4

Table 4.16 *Train and test accuracies for stRGCN trained on single lines*

As already witnessed before when evaluating GCNs (section 4.5, especially in Transfer Learning), also this model has difficulties in solving the root-cause analysis for the very first machines along the main-stream. The confusion matrices (Figure 4.22) of customers a, b and h show that the model either never predicts the first machines like Depalletizer and Unpacker (customer-b and -h) or too many predictions of the Depalletizer at customer-a. Since especially the Depalletizer at customer-a is way too often predicted by the model, we want to discuss this in more detail. The reason is most probably again the way of labeling. At customer-a the expert often interprets low quality of the returnable glass bottles as the root cause of problems in the line. And for this scenario he usually selects the Depalletizer as corresponding root cause machine since he says there was no correct machine



(a) *customer-a* (b) *customer-b*

(c) *customer-h* (d) *customer-w*

Figure 4.22 *Confusion matrices for training the spatio-temporal Graph Convolutional Neural Network on each single customer.*

to select. Since the algorithm goes the contrary way, so first selects the root-cause machine and in a later step more information (e.g. about glass quality problems) is given via the machines error message. As a result of the different approaches of expert and algorithm our evaluation result when comparing the classified causer machines often shows wrong results.

4.6.2 Multi-Task Learning using stRGCN

With a train-accuracy of $97.9 \pm 0.6\%$ after the training epoch which yielded highest validation accuracy, this model shows an improvement in terms of adption to the training set compared to the RGCNs Multi-Task training accuracy of $91.5 \pm 0.8\%$. As one can see in Figure 4.23, the network even improves its training accuracy steadily to a saturation level of about 100%. That means, the model is capable of learning

the patterns hidden in the training set perfectly.

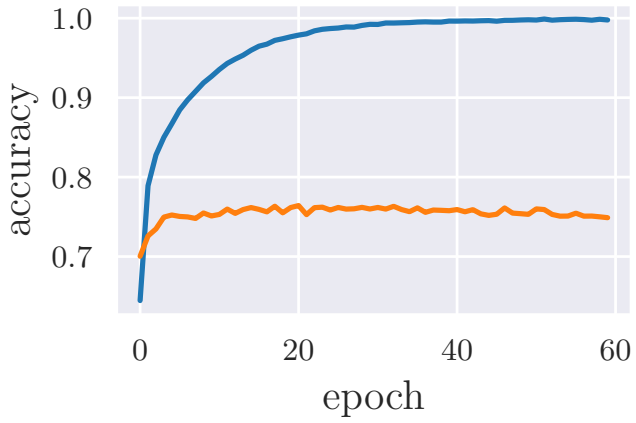


Figure 4.23 *Training- (upper, blue curve) and validation-accuracy (lower, orange curve) of Multi-Task Learning by using the stRGCN architecture. Validation accuracy stagnates early in training process. But this behaviour is independent from hyperparamters like learning-rate.*

But the maximum of validation accuracy is already reached as early as about epoch 15 to 20 and afterwards validation performance decreases. Thus the model gets into overfitting.

We conducted hyperparamter-tuning to decrease overfitting. Therefore, we investigated the influence of Adam optimizer's learning-rate and the dropout-rate that is used within the models blocks, since these are the usual tools to decrease overfitting. Both provide best results with their default values of 0.001 for learning rate and 0.2 for dropout. Thus overfitting can not be decreased by these parameters. Investigating the learning behaviour from a data-perspective opens up new insights: Remembering the fact that training-sets are labeled algorithmically while validation- and test-sets are labeled by human experts, is an important property of the used data. During Multi-Task Learning, the model has to adapt to prediction of four Random Forest models which created the training labels. This is a task that a model with huge adaption abilities like our stRGCN should be able to solve. And the high accuracies of the Random Forest models (between 73 and 94%) set high expectations towards the trained stRGCN model which hopefully combines what the Random Forests learned before. Due to a training accuracy of up to 100% the model showed that it actually can combine what the four Random Forests learned independently,

but the problem now is in converting the learned on unknown examples. And at generalizing its ability, the trained stRGCN model seems to have its deficiencies what results in a heavy difference between training- and validation-/test-accuracy.

	test acc [%]
customer-a	63.9 ± 3.8
customer-b	84.1 ± 1.7
customer-h	82.7 ± 2.4
customer-w	55.8 ± 7.9

Table 4.17 *Test accuracies for each customer. One stRGCN is trained on all filling lines at once using Multi-Task Learning and afterwards evaluated on test-sets of each customer individually.*

In Table 4.17, the results of evaluating the stRGCN model after being trained by Mulit-Task Learning on a combined dataset of all four customers, are shown. Corresponding confusion matrices can be found in the Appendix in subsection 6.2.2. For customers a, b and h, the accuracies are in a good range similar to the results of training the same Neural Network architecture on each line individually (compare Table 4.16). But evaluation on customer-w shows noticeable worse performance compared to 86.8 ± 8.0%, when training on only this lines data. A difference of about 30% was not to expect and is probably caused by the strongly differing line characteristic compared to all other customers. Differences like the fact that customer-ws first three machines are electronically and mechanically blocked and thus do not show temporal delay during error propagation, or the position of the lead-machine at the very start of the line, could be a factor.

This result opens up an important question for the future: Is it better to conduct Multi-Task Learning over all different lines with one model to provide this with as much and as various data as possible or can better results be achieved by clustering filling lines into e.g. returnable glass lines, one-way plastic lines, can lines etc. to provide the opportunity to the model to deeper adapt to the given characteristics of this cluster's lines?

4.6.3 Transfer Learning using stRGCN

The achieved accuracies in Transfer Learning for customers a and b show the potential of this approach! Since the performance loss comparing MTL results (Table 4.17) to

Transfer Learning (Table 4.18) is with approximately 4% and 8% low, respecting the fact that the model only gets the graph-structure of the new filling line as knowledge and cannot adapt to any data.

	mean test acc	mean train acc
customer-a	59.1 ± 3.0	96.1 ± 0.7
customer-b	76.6 ± 2.1	83.9 ± 1.3
customer-h	70.1 ± 4.5	96.2 ± 1.5
customer-w	65.7 ± 3.7	95.1 ± 1.6

Table 4.18 *Test accuracies for each customer after training a stRGCN on all customers at once using Multi-Task Learning.*

This provides the option for productive usage, to transfer a model to a unknown customer and thus put a Machine Learning model in place without needing to wait for a certain amount of label-example pairs for initial training.

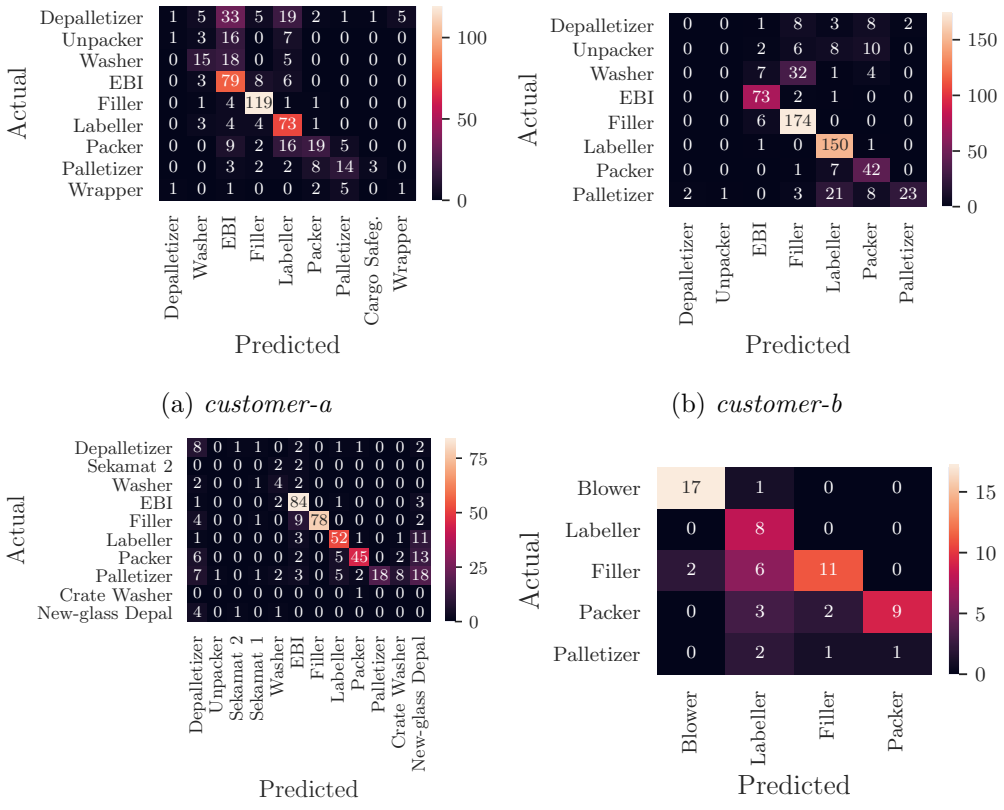
Contrary to that, the gap between Customer-bs accuracy after MTL-training and when using Transfer Learning is with about 15% larger and drops the accuracy significantly. Customer-w however shows a strange behavior when again comparing MTL and Transfer Learning accuracies. It advances by about 10%. This only can be explained by the suspiciously low accuracy during MTL.

When looking deeper into detail of the results (Figure 4.24), the confusion matrices show a known tendency: The central part and the area, downstream of the lead-machine at lines a, b and h can be resolved well. Whereas the first machines along the mainstream again are more difficult to learn. Hint: The New-glass Depalletizer at customer-h is also located at the very beginning of the line, in parallel position to Depalletizer and Unpacker.

4.7 Comparing RGCN to stRGCN

In this work, we built up two Neural Network architectures that are capable of Multi-Task Learning and Transfer Learning. Thus, they are able to firstly adapt to different packaging and filling lines at once and subsequently predicting root-causer machines in new lines without any training-data.

But there is a difference in their data preprocessing pipelines. Since RGCN works on manually extracted features, a manual configuration of the corresponding parameters is needed for each line. This is not the case for stRGCN, which covers feature



(a) *customer-a* (b) *customer-b*

(c) *customer-h* (d) *customer-w*

Figure 4.24 *Confusion matrices for every customer after conducting Multi-Task Learning on the other three customers data and afterward transferring the model to the given customer. Using spatio-temporal Graph Convolutional Neural Network on algorithmically labeled data.*

extraction inside the network and thus needs even less manual work per roll-out at one filling line.

But since test-results of stRGCN are usually worse than those of RGCN, additionally learning the feature-extraction seems to decrease stRGCN's test accuracy compared to RGCN with the current amount of data being available. This is probably caused by a severely more complex network architecture, containing loads of learnable parameters in case of stRGCN. The amount of available data seems to be sufficient for training the smaller RGCN to high accuracies but stRGCN probably needs a larger dataset to reach the accuracies that are possible to hit, like shown by RGCN and Random Forest.

4.8 Milestones for training Graph Convolutional Networks

When examining the potential of GCNs on the given task for the first time, we used manually preprocessed data and a simple three-layer GCN architecture with a fully connected decision layer on top. This approach delivered accuracies of roughly 50% and thus showed that this model type is able to learn patterns in the data. But for competitive results, the performance had to be pushed significantly. This was achieved by introducing edge-types and a fully convolutional architecture for our GCN approach as well as the stGCN approach. For the latter we additionally introduced an improved way of temporal data selection and preprocessing and added information about the lead-machine to the data.

In this section, differences in the results of the models when using the named improvements versus without using them are shown.

4.8.1 Edge types

The most important breakthrough was brought about by introducing edge-types and thus different relations between machines. Since machines show the states "lack" and "tailback" when an error arrives they provide information if the error arrived from upstream or downstream direction. To utilize this information, the model somehow has to know the direction of material flow. We translate this information

	using edge-types	without edge-types
average	87.0 ± 1.0%	58.8 ± 2.9%
customer_h	88.1 ± 2.0%	64.1 ± 3.0%
customer_a	80.3 ± 3.0%	43.1 ± 3.8%
customer_b	89.4 ± 1.0%	65.3 ± 3.6%
customer_w	90.3 ± 1.0%	62.5 ± 6.4%

Table 4.19 *Comparing test-accuracies of fully convolutional GCN architectures on data preprocessed with Manual Feature Extraction using Multi-Task Learning. On the one hand, we see a network using Relational GCN layers and thus different edge-types, on the other hand a network with plain GCN layers and thus no different edge-types. Using edge-types leads to an improvement of* $\approx 30\%$ *in accuracy.*

into "upstream" and "downstream" edges between machines in the graph of each

line. Thus, the undirected connection between two machines is split up into an "upstream" and a "downstream" edge. This enables the network to learn how to utilize "lack" and "tailback" information.

As Table 4.19 shows with accuracy increase of about 30% per customer and severely decreased variance of results, replacing plain Graph Convolution Layers with Relational Graph Convolution Layers was a crucial step for the Neural Network architectures.

4.8.2 Fully convolutional architecture

Our very first approach of using GCNs for root-cause machine classification contained using GCN-layers for feature extraction and a fully-connected layer (Single Layer Perceptron) as so-called *decision layer*, thus converting the features, extracted by the Graph Convolution layers, into class probabilities.

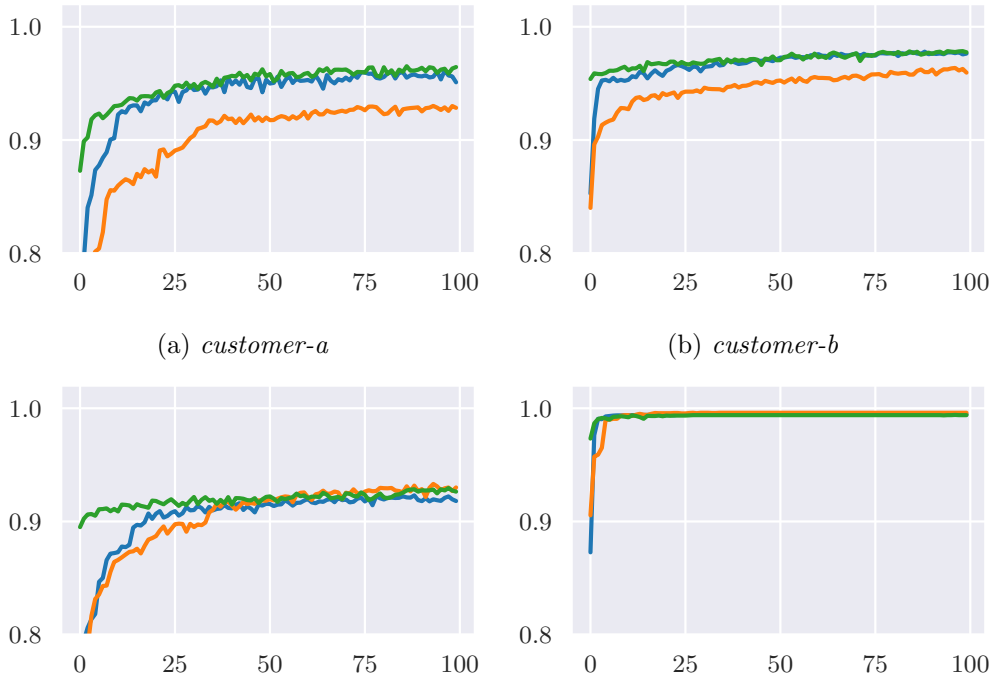
To reduce the number of network parameters that have to be optimized during training process, we exchanged the fully connected decision layer by a convolutional layer in combination with global-max-pooling.

But it became apparent very quickly that this architecture adds unnecessary complexity to the information flow through the network (compare section 3.10). Thus, the fully convolutional approach was adapted to the given problem to result in the final architecture.

	RGCN-with-dense	classic-fcRGCN	fcRGCN
customer_a	78.4 +- 2.7	74.7 +- 2.7	83.0 +- 1.2
customer_b	91.6 +- 0.8	90.0 +- 1.6	92.7 +- 1.8
customer_h	83.8 +- 7.5	85.0 +- 4.1	85.7 +- 0.8
customer_w	90.6 +- 0.0	90.3 +- 0.6	88.3 +- 0.0

Table 4.20 *Test-accuracies for different NN-architectures when training one network per line. Results are averaged over five runs, test-accuracy is computed after 100 epochs. Fully convolutional RGCN outperforms the other networks at three of four customers.*

As we can see in Table 4.20, the newly introduced fully convolutional Relational Graph Neural Network (RGCN) outperforms the other architectures at all customers but customer-w with additionally showing low variances, and thus stable results, for all customers.



(a) *customer-a* (b) *customer-b*

(c) *customer-h* (d) *customer-w*

Figure 4.25 *Comparing the evolution of training accuracy over epochs for a RGCN with a fully connected decision layer (blue), a RGCN with fully convolutional architecture using global pooling (orange) and a RGCN with the newly introduced fully convolutional architecture (green). Results are averaged over five runs. The newly introduced architecture learns significantly fast than the others and is already less than 10% beyond the final training accuracy after the first epoch.*

Additionally, training progress (training-accuracy after each epoch of training) shows significant differences between the architectures (compare Figure 4.25) although the final train-accuracies are similar between all models besides the classic fully-convolutional approach at customers a and b. Looking at the training accuracies after the very first epoch of training, we can see that the introduced fully-convolutional architecture has already reached comparably high accuracies after being trained on each training example only once. In case of customer-a, training accuracy rises from about 87% to 96% after the first epoch. For the other customers only three to four percent of training accuracy are gained after the first epoch. This shows exceptionally fast Neural Network training for the newly introduced net-

work which is a hint for a well chosen architecture which simplifies information flow through the network and thus also facilitates approximating the pattern recognition during model training.

4.8.3 Temporal data selection

We started training DTW+kNN and GCN approaches by extracting data from e.g. 10min before the start of the lead-machine downtime until 10s after it, like also done in related work [2]. Changing this towards selecting the data until the end of the lead-machine downtime (compare Figure 3.6) led to improvements of up to 5%, when using the stGCN, depending on the customer (compare Table 4.21).

	test-accuracy for temp. data selection a)	test-accuracy for temp. data selection b)
customer-a	63.4 ± 2.4	67.0 ± 2.9
customer-b	82.1 ± 1.7	87.1 ± 1.9
customer-h	77.5 ± 2.3	80.9 ± 1.5
customer-w	90.7 ± 4.2	89.7 ± 4.0

Table 4.21 *Comparing test-accuracies of stRGCN between temporal data selection until the stop of the lead-machine (a) or end of the lead-machine downtime (b). Trained on each customer individually.*

By providing information about events in the line also during the lead-machine's stop, we enable the models to also follow patterns during the machines' restarting process towards the lead-machines restart. Thus, correlations between machine stoppages can be better estimated. Accordingly, results of customers a, b and h show an increase of accuracy by 3% to 5%. In contrast, we cannot see improvements at customer-w which is reasonable due to the very simple and few temporary patterns that can be found in its data. Due to the first three machines being blocked and thus stopping and starting simultaneously, there are only two machines left that show temporally delayed events compared to the lead-machine. And they are causers for only 12% of the examples. Thus, it is not surprising that an improvement in processing the temporal patterns inside the data shows little effect at customer-w.

4.8.4 Lead-machine information

When training a ML-model on data of one filling and packaging line, the model is able to learn the characteristics of the line and thus also to properly handle the lead-machine in comparison to the rest of the machines. In contrast, when conducting Multi Task Learning, specialization at one line is impossible. Thus the model can't learn to treat one machine alternatively. Thus, we added the information about the position of the lead-machine to the data like introduced in section 3.11.

Improvements of about 3% in accuracy (compare Table 4.22) verify the assumption of needing this information.

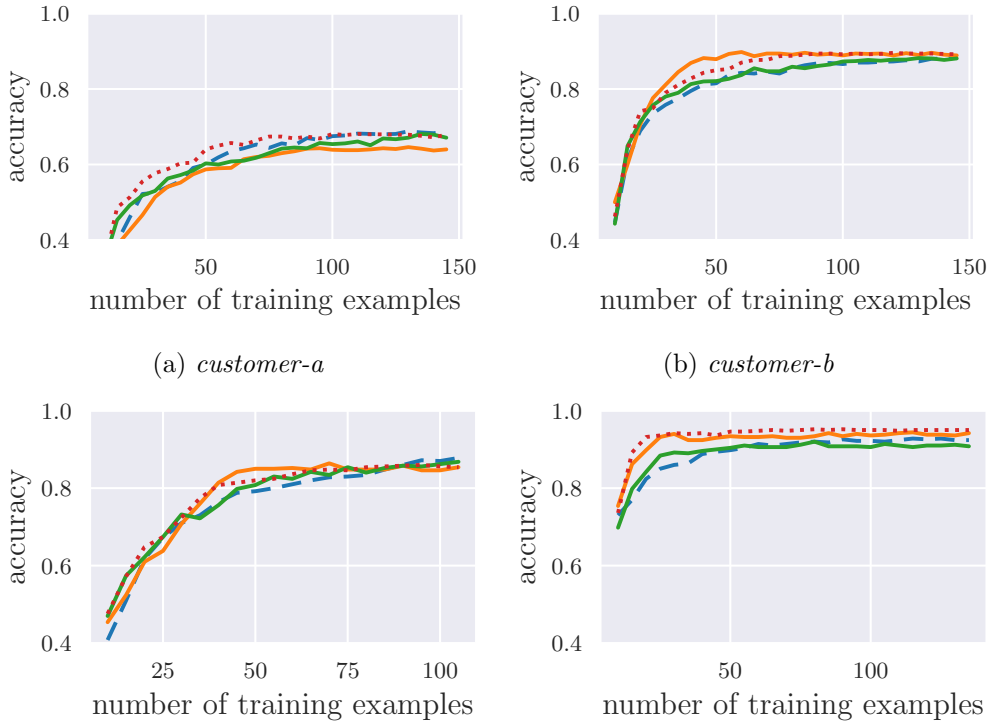
	without lead-machine info	with lead-machine info
customer-a	59.8 ± 3.9	63.9 ± 3.8
customer-b	81.9 ± 2.7	84.1 ± 1.7
customer-h	79.5 ± 1.6	82.7 ± 2.4
customer-w	56.1 ± 8.6	55.8 ± 7.9

Table 4.22 *Comparing test-accuracies of Multi-Task training using stRGCN between data without lead-machine info vs. data with lead-machine information. Averaged over 10 runs. Providing information about the position of the lead-machine generally improves test-accuracy.*

As for the last milestone, introducing to provide the position of the lead-machine to the model improves accuracy for all customers but customer-w. And once again, this most probably is caused by the simplicity of its line structure which makes the added information redundant to the machine-state information in the majority of the examples.

4.9 Active Learning

In this section, we show that selecting lead-machine stoppages, which should be labeled, by a query strategy outperforms randomly selecting the stops significantly. In a first step, this result is shown using pool-based Active Learning, afterwards we show the same behavior for querying examples from mini-pools that only contain examples of one shift in a filling line. Since the goal of Active Learning is to reduce labeling effort, we will compare the reached test accuracy of a Random Forest classifier with a given number of training examples that are chosen by different query strategies.



(a) *customer-a* (b) *customer-b*

(c) *customer-h* (d) *customer-w*

Figure 4.26 *Comparing different query strategies for pool-based Active Learning. AL starts with 10 labeled samples. Model: Random Forest with 500 estimators and maximum depth of 7. Training is conducted for every number of training samples 10 times and results are averaged. Random Sampling strategy is plotted in dashed blue, Uncertainty Sampling in orange, Nearest Neighbor Criterion in green and Dynamic Ensemble Active Learning in dotted red.*

Results shown in Figure 4.26 witness increased accuracy for training on examples

queried using DEAL(combining Uncertainty Sampling and Nearest Neighbor Criterion) in comparison to randomly sampling. Especially for low amounts of training examples, the difference is large. In other words, the model learns faster along the growing number of training examples. Nevertheless Uncertainty Sampling (UC) outperforms DEAL at some occasions. But there are multiple reasons to prefer DEAL to Uncertainty Sampling. The main one is called *sampling bias* and describes the bias that is introduced by the query strategy when selecting the examples. Due to this influence, the labeled training set shows a different example distribution than the full dataset. In the case of Uncertainty Sampling, examples near the models decision boundaries (boundaries between areas in input space that correspond to a class) are over-represented in the labeled dataset. This often leads to poor generalization ability of the model.

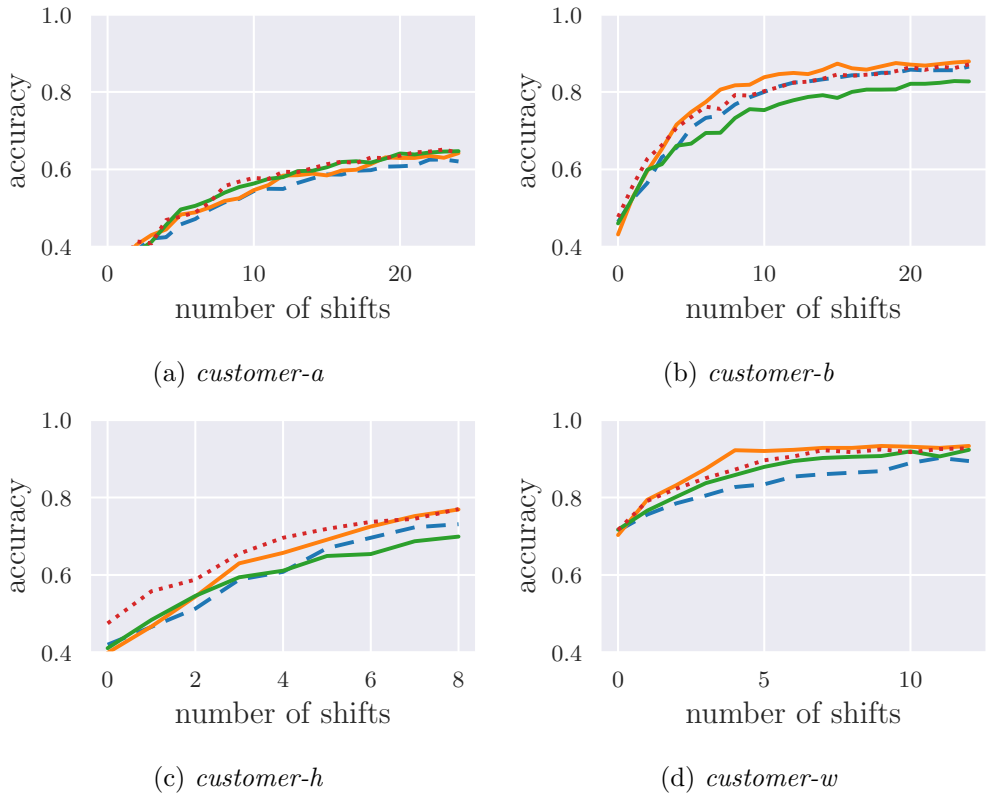
Connected to the sampling bias, the variance between training runs on datasets queried by Uncertainty Sampling is high since sometimes the distribution of training samples is sufficient, sometimes e.g. only depicts some but not all boundaries between classes in the example space.

Therefore, the sampling strategy Nearest Neighbor Criterion (NNC) turned out to be a very well suited combination for Uncertainty Sampling. NNCs goal is to tweak the labeled training-sets distribution to be as similar as possible to the original data distribution. Thus training accuracies are stabilized and generalization ability is increased by combining UC and NNC using DEAL. For deeper insight into these results the reader is also referred to [36].

After looking at the results of plain pool based Active Learning, we now evaluate shift-wise pool-based Active Learning. As starting point, the model gets ten randomly selected labeled examples. After that, a shift is assumed to provide ten lead-machine stoppages (examples) each. The query selects three of the ten sample-points which are labeled and added to the train-set. We again train the Random Forest classifier after each epoch for ten times (for each query strategy) and average over the results (Figure 4.27).

As before, this approach also shows a severe speed-up of the training process. Customers a and h need about 25% less labels when the examples are queried by DEAL. Customer-b shows advantages in the early stages of training, until an accuracy of apporximately 80% is reached. Afterwards the progress is similar for Random Sampling and DEAL. But even in this case accuracy of 90% is reached after approx 25 shifts which equals about 2 weeks of production. That means that

the end-user only needs to provide three selected labels per shift over two weeks to get 90% accuracy instead of labeling all 250 stoppages that occured during the two weeks. At customer-w 90% accuracy is reached after eight shifts when using DEAL. Model training based on Random Sampling has not reached equal accuracy after 14 shifts. Only needing eight instead of at least 14 shifts until an accuracy of 90% is reached equals saving more than 40% work for labeling data.



(a) *customer-a* (b) *customer-b*

(c) *customer-h* (d) *customer-w*

Figure 4.27 *Comparing different query strategies for Active Learning. AL starts with 10 labeled samples. Afterwards, the query always selects 3 out of 10 examples (mini-pool per shift) to simulate the behavior of Active Learning per shift. Model is a Random Forest with 500 estimators and maximum depth of 7. Random Sampling strategy is plotted in dashed blue, Uncertainty Sampling in orange, Nearest Neighbor Criterion in green and Dynamic Ensemble Active Learning (combining Uncertainty Sampling and Nearest Neighbor Criterion) in dotted red.*

4.10 Results overview

This section gives an overview of results and compares all models that were evaluated in this work. Firstly, strengths and weaknesses of all types of models are compared qualitatively. Afterwards, the quantitative accuracies are compared over all algorithms.

	Avalanche	Fair Detective	ML per Line	ML over multiple lines
Level of accuracies	medium	high	very high	high
Duration of roll-out	medium	fast	slow	fast
Explainability	high	high	low	low
Robustness against low data quality	high	medium	very high	high
Adaptability to customer's interpretation standards	no	no	yes	no
Amount of labeling effort	no	no	high	medium
Need of Machine Learning environment	no	no	yes	yes

Table 4.23 *Comparing different approaches qualitatively. As a result, Fair Detective is the reasonable first increment to improve root-cause analysis starting from Avalanche Algorithm. In cases of low data-quality or individual customers' interpretation standards for lead-machine stoppages, ML approaches nonetheless, offer the chance to achieve highest accuracies.*

	Avalanche algorithm	DTW+ KNN
customer-a	47.4	67.7 ± 2.1
customer-b	67.6	70.0 ± 2.8
customer-h	63.3	63.3 ± 3.9
customer-w	63.1	87.0 ± 2.8

Table 4.24 *Baseline accuracies including Avalanche algorithm, which is the currently productive system at Syskron, and combination of Dynamic Time Warping and kNN. Results given in percent.*

	rule-based	Random Forest	RGCN single customer	RGCN Multi-Task	RGCN Transfer Learning	stRGCN single customer	stRGCN Multi-Task	stRGCN Transfer Learning
customer-a	53.5	72.8 ± 4.0	**82.8 ± 1.0**	81.3 ± 1.4	66.9 ± 2.5	67.2 ± 2.6	63.9 ± 3.8	59.1 ± 3.0
customer-b	92.1	90.7 ± 1.6	**93.4 ± 0.6**	91.3 ± 1.0	80.9 ± 1.5	87.7 ± 1.4	84.1 ± 1.7	76.6 ± 2.1
customer-h	79.9	82.5 ± 5.0	86.0 ± 1.3	**86.3 ± 0.7**	82.5 ± 2.3	79.7 ± 2.0	82.7 ± 2.4	70.1 ± 4.5
customer-w	68.5	**93.9 ± 1.3**	83.3 ± 3.3	88.3 ± 1.1	70.3 ± 5.5	86.8 ± 6.0	55.8 ± 7.9	65.7 ± 3.7

Table 4.25 *Accuracies of newly introduced approaches: Rule-based expert system, Random Forest and Neural Networks. Each NN is trained on data of each line individually (single customer), on all lines at once (Multi-Task) and on all lines but one and afterwards transferred to the left-out one (Transfer Learning). Neural Networks are trained on algorithmically labeled data. Results given in percent.*

5 Conclusion and Outlook

5.1 Conclusion

In this thesis multiple approaches for algorithmically finding the root-cause machine corresponding to a stoppage of a filling and packaging line, were developed and evaluated. The goal was to increase classification accuracy and decrease roll-out costs per filling line. Additionally, any model should be immediately usable within the day of roll-out.

I built up a dataset containing about 1800 manually labeled examples of four different lines. By algorithmically labeling additional data, I was able to create a dataset that includes about 10 500 examples, which is, to the best of my knowledge, by far the largest dataset for this kind of problem.

All developed algorithms were compared to the general baseline "Avalanche algorithm", which is the algorithm that is currently used in Syskron's productive system.

I introduced multiple approaches, allocated in the field of Time-Series Classifcation, including a rule-based algorithm as well as four different Machine Learning algorithms. Beyond the ML-approaches, Dynamic Time Warping combined with k-Nearest Neighbors (DTW+k-NN) acts as canonical baseline method. Additionally, Random Forest, a Relational Graph Convolution Neural Network (RGCN) and a spatio-temporal Relational Graph Convolution Neural Network (stRGCN) were applied. For the rule-based system and the Neural Network approaches, the filling and packaging line was represented as a graph with machines as nodes and conveyors as edges.

The introduced rule-based approach outperforms the Avalanche algorithm. Furthermore, all Machine Learning approaches outperform the DTW+k-NN baseline and the Avalanche algorithm. Thus, all developed approaches fulfill the requirement of increasing the classification accuracy.

During this work, a tool for automatically extracting characteristic values of tem-

poral error propagation was developed and implemented. It is used for configuring the Avalanche algorithm and the introduced rule-based system and heavily decreases configuration effort per roll-out. Therefore, the tool contributes to rendering the rule-based models scalable.

To apply Neural Networks on categorical data with dynamic temporal length (since stoppages in the line have variable durations), two approaches of data preprocessing were elaborated. The first approach extracts engineered features from the temporal data-dimension as input to a Random Forest and a RGCN. The second one down-samples the multivariate time-series of variable length, either with fixed sampling rate to a shorter time-series that still has flexible lengths for DTW+k-NN, or by using an adaptive sampling rate, to a fixed length for stRGCN.

The Neural Network(NN) approaches are elaborated in order to solve a Graph Classification task with the special property of classes being equal to the graphs nodes, to select the root-cause machine in the line. I exploited this property to build up the above mentioned Neural Network architectures in a fully convolutional approach. This way, the models are adaptable to any filling line by exchanging the line's graph but without the need for model training. Using these models, it is now possible to use Multi-Task Learning(MTL) over multiple production lines, and Transfer Learning(TL) to apply a trained model on an unknown line. This contributes to the goals of cheaper roll-out costs per line as well as immediate usability, since after training a model on a variety of lines, it can be transferred to another line without training it on the new line's data. To the best of my knowledge, the fully convolutional Graph Neural Network architecture in general as well as the special adaption for MTL and TL are novel contributions to scientific discourse.

During evaluation, it has turned out that choosing the appropriate model for the given task is severely dependent on the available data-quantity as well as on the degree of standardization in machine-state data and in selecting the root-cause machine by the users.

If the described standardization is of low degree, only a Machine Learning(ML) model, that is trained on only one filling line, has the ability to adapt to all peculiarities of the given line and thus will be able to solve the task with high accuracy. On the other hand, a high degree of standardization simplifies the task in such way that the introduced rule-based expert system is able to solve the task with accuracies of over 90%. In this case, classifications are highly explainable and all efforts regarding training, deploying and maintaining ML-models are avoided.

In between the two described scenarios, when a sufficient but non-optimal data-quality is present, ML-approaches promise an increased ability of overcoming data inconsistencies due to, in comparison to the rule-based approach, significantly larger receptive fields and the ability to learn a huge variety of patterns.

In case of implementing one of the introduced Machine Learning models, expert labels are necessary for building up the corresponding dataset. In order to decrease the labeling effort, Active Learning was introduced and optimized to choose the most promising examples and request the corresponding expert's assessment. We were able to show labor savings of the experts of about 40% by combining the query strategies Uncertainty Sampling and Nearest Neighbor Criterion by using Dynamic Ensemble Active Learning. This heavily contributes to the requirement of saving labor per roll-out.

With this work, multiple algorithms for detecting the root-cause machine were elaborated which were able to solve the given problem with accuracies of at least 83% and up to 94% and were able to be activated for a new filling- and packaging-line quickly and with substantially decreased effort.

5.2 Outlook

In general, improvements in data-standardization and a so-called feedback loop, for gaining customers' classifications corresponding to stoppages as labels, have to be realized. Additionally, the standards for interpreting a lead-machine stoppage have to be set and published by Syskron and Krones to gain a shared awareness beyond users, and thus increase acceptance of algorithm results by increasing transparency.

By implementing the described feedback loop, the foundation for a growing dataset is laid. In the first step, this can be used to further improve the rule based system and implement it to the productive service. Choosing the rule based system as first increment is reasonable since the corresponding effort is relatively low but introduces a significant accuracy increase. Depending on the status of data-quality and the standardization the of customers' interpretation, introducing a Machine Learning based model has to be re-evaluated on a regular basis.

Over time, the behavior of introduced NN-models with growing datasets has to be evaluated and hyperparameters have to be re-evaluated to match the new demands. Additionally, the effect of introducing filter pruning strategies has to be analyzed.

I assume that Multi-Task Learning on expert labels will start outperforming training on the algorithmically labeled dataset when a sufficient amount of manually labeled examples is available. This assumption has to be verified and the corresponding amount of manually provided labels has to be estimated. When working with expert labels in any way, label cleaning will play a very important role and has to be developed. Additionally, the question, whether one model should be trained to solve the classification task for all filling lines or if alternatively overall performance is increased by splitting the lines into clusters of similar characteristics (like returnable glass lines vs. one-way plastic lines) and train one model per cluster, has to be answered.

6 Appendix

6.1 Line graphs

For better understanding the four filling and packaging lines studied, their graph structures with machines as nodes and connecting conveyors as edges are presented and explained in this section.

The architectures of customer a's and b's lines are very similar and made for filling beer into multi-way glass bottles like usual for German beer (compare Figure 6.2 and Figure 6.1). The process starts by depalettizing crates, full of empty and used bottles, from the incoming palette. The empty palette is transported to the Palettizer to recieve the completed full crates. The depalettized crates are transported to the Unpacker where empty bottles are removed from the crates and put on bottle transport towards the Washer. The empty crates are carried to the packer. After washing the bottles, they have to pass the Empty Bottle Inspector which controls the bottle for all types of quality issues before filling the beer into it.

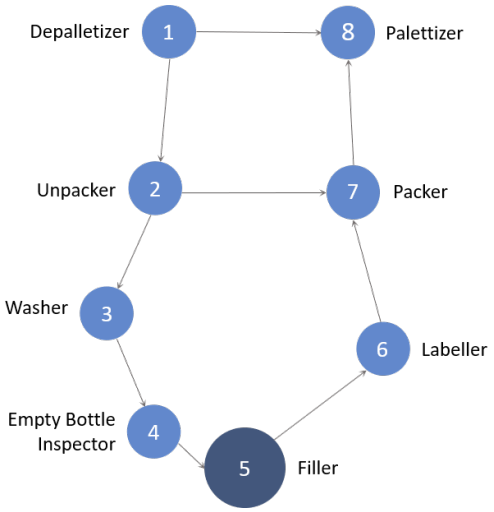


Figure 6.1 *Line graph of customer-b filling and packaging line. This is used to fill beer into multi-way glass bottles*

The Filler includes a capping module s.t. the bottle is filled and also closed when exiting from the Filler. Subsequently, the bottle has to get labelled in the Labeller. Now the bottle itself is finished and is put into a crate at the Packer and the crate is placed onto a palette inside the palettizer.

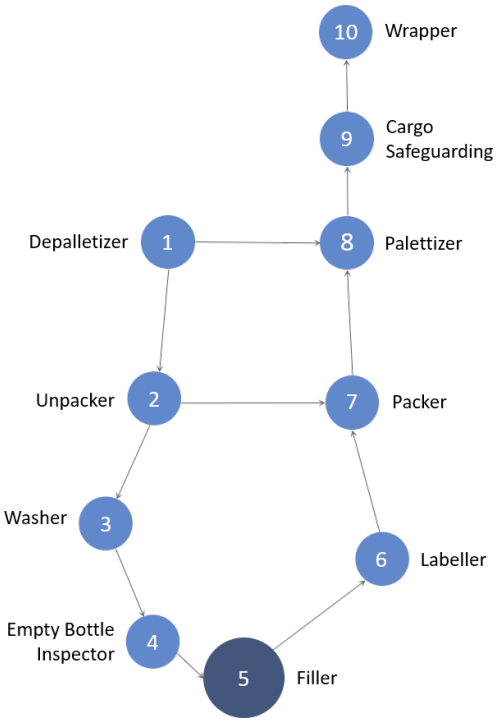


Figure 6.2 *Line graph of customer-a filling and packaging line. This is used to fill beer into multi-way glass bottles*

In between all of the described steps, there are usually again multiple smaller machines or modules e.g. for checking the filling level or placement of the labels. These are commonly either directly connected to their parent-machine like the filling level inspector is connected to the Filler, or show very low probability of causing a stoppage. Thus, they are not or only indirectly taken into account in the root cause analysis.

An example for machines like this are Cargo Safeguarding and Wrapper of customer-a. They also exist at customer b and secure the crates from falling off their palette. But at customer-b the decision was made that their probability of being the root causer is too small to take them into the analysis while customer-a wanted to observe them. As we can see in the table of available labels e.g. Table 4.3, Cargo Safeguarding has never been the root cause for a labeled example and Wrapper was selected only ten from 539 times by the expert.

Customer-h also runs a multi-way glass line which thus is very similar to the already mentioned lines (compare Figure 6.4). This time, water and soft drinks are filled and additionally, the line is a little more complex since it needs to run on a very high efficiency level due to smaller win margins on selling water compared to selling beer.

In addition to the already described machines, in this line two Sekamats, arranged in parallel, are present. These filter the unpacked bottles for shapes that are not the currently processed one. Due to their parallel arrangement, the functional area of sorting becomes significantly more stable since most probably one of the two machines is usually able to remain working when the other has a failure. Thus the

subsequent conveyor areas fill with only about half the speed. This can also be witnessed when looing again into Table 4.3 where the Sekamats are only chosen 4 of 428 times by the expert.

The last of our analyzed lines is very different to the previous ones (compare Figure 6.3). On the one hand, it does not fill glass but plastic bottles, on the other hand, these are one-way bottles and not returnable ones. Thus, the line starts with a Blower that recieves so-called preforms that look a little similar to test tubes which are known from laboratories but already possess the thread which will later carry the cap. This preform is heated and expanded into its final shape. Afterwards it is handed to the Labeller which is directly connected without a conveyor in between. Subsequently, the labeled bottle again is directly handed over to the Filler. Those three machines are combined to one block. That also means: If one of them has a problem, all three are stopped. But due to severe place saving this drawback is more than erased. The bottle now is in its final state and has to be packed and palettized. As we are used to, one-way plastic bottles most of the time are not put into a crate but combined to e.g. a sixpack by a plastic sleeve. The sixpacks are afterwards put onto their palette in several layers.

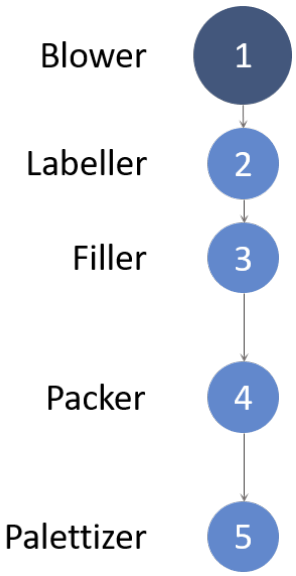


Figure 6.3 *Line graph of customer-w filling and packaging line. This is used to fill water into one-way plastic bottles*

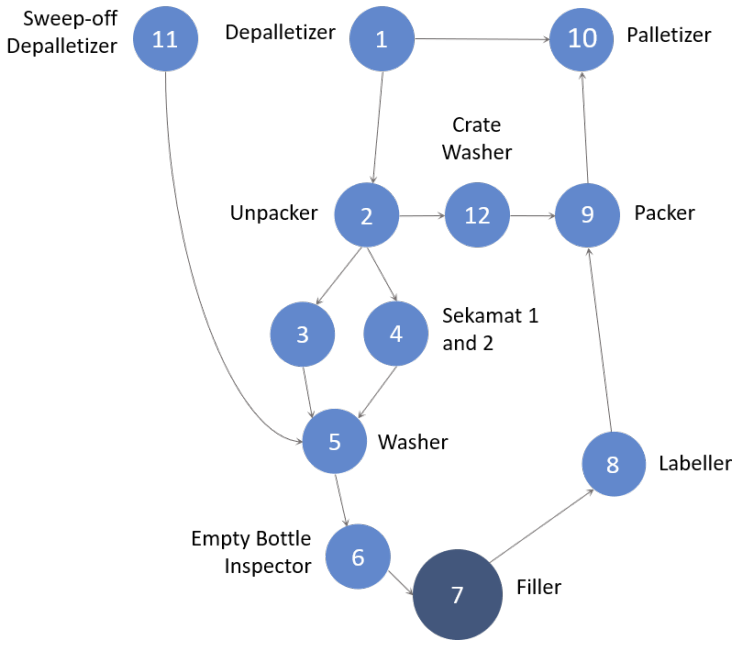
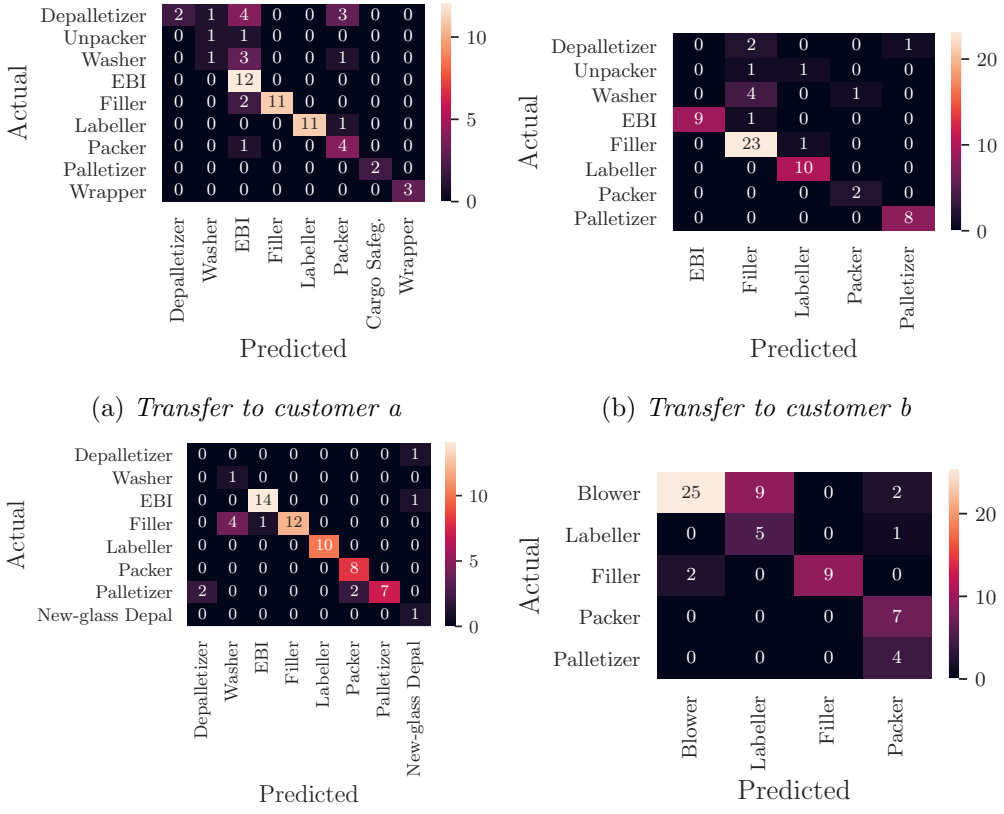


Figure 6.4 *Line graph of customer-h filling and packaging line. This is used to fill water or soft-drinks into multi-way glass bottles*

6.2 Confusion matrices

6.2.1 Transfer Learning: RGCN



(a) *Transfer to customer a*

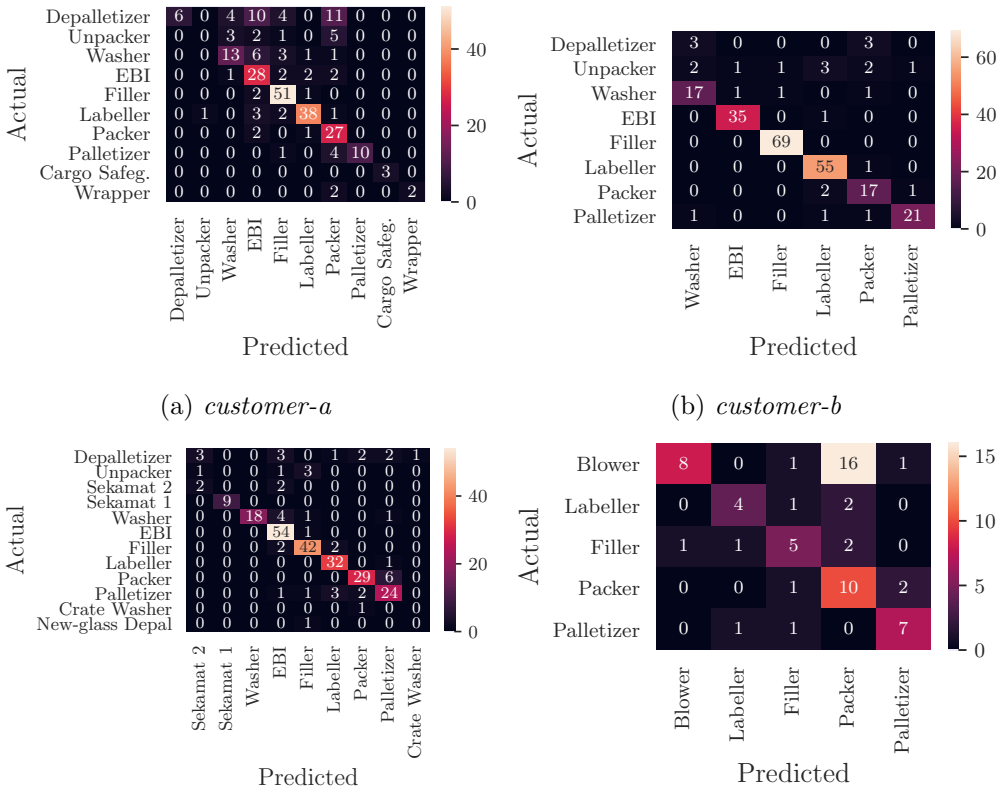
(b) *Transfer to customer b*

(c) *Transfer to customer h*

(d) *Transfer to customer w*

Figure 6.5 *Confusion matrix for performance of RGCN when transferring to given customer after Multi-Task Learning on the other three customers.*

6.2.2 Multi-Task Learning: spatio-temporal RGCN



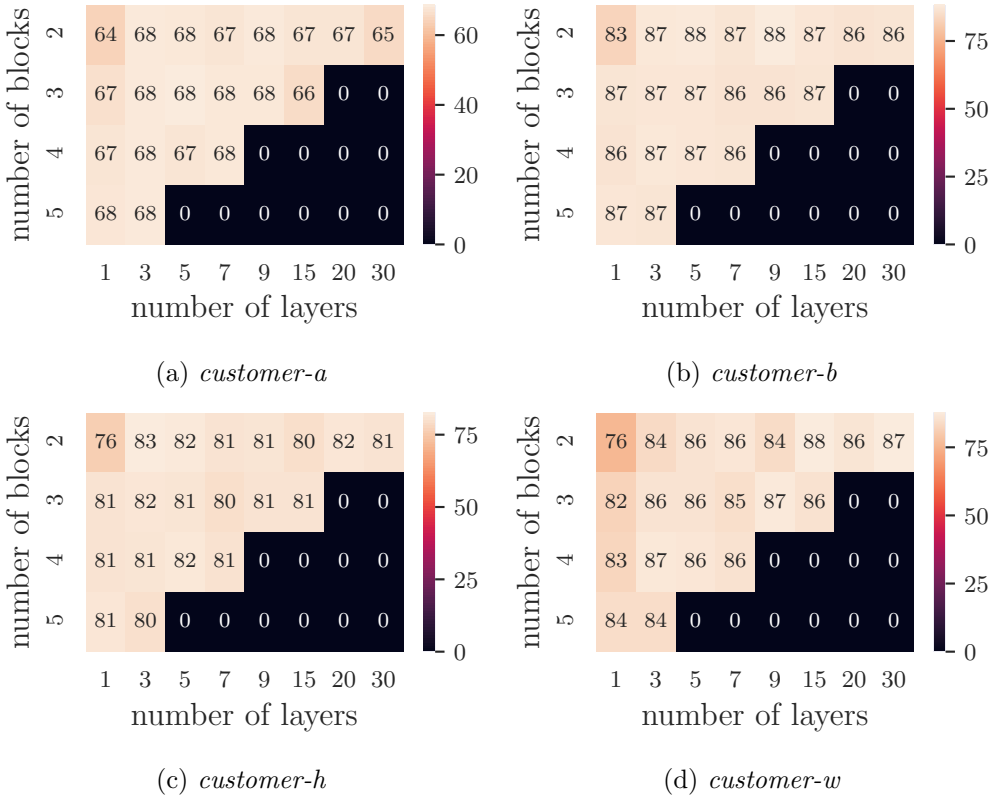
(a) *customer-a* (b) *customer-b*

(c) *customer-h* (d) *customer-w*

Figure 6.6 *Confusion matrices for every customer after conducting Multi-Task Learning using the spatio-temporal Graph Convolutional Neural Network on algorithmically labeled data of all four customers.*

6.3 Hyperparameter tuning stRGCN

6.3.1 Number of blocks and layers

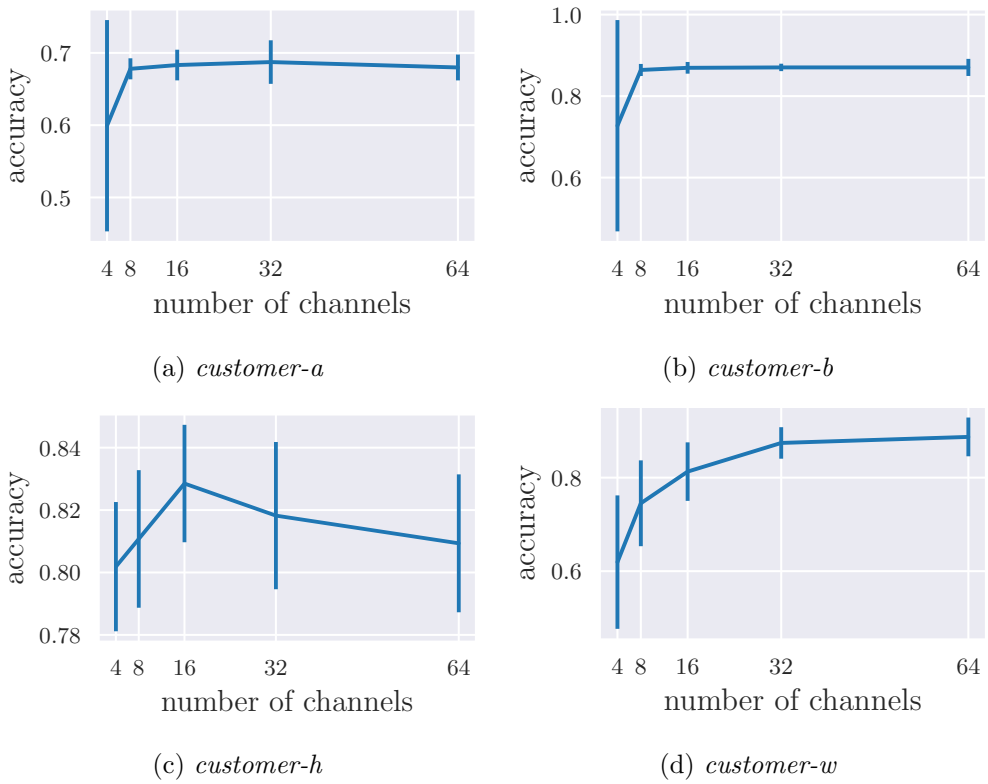


(a) *customer-a* (b) *customer-b*

(c) *customer-h* (d) *customer-w*

Figure 6.7 *Accuracies in percent for comparing accuracies depending on selected number of blocks and number of layers. Averaged over 10 runs. Combinations showing an accuracy of zero were not evaluated. Three blocks with three layers each were selected as hyperparameters for later usages.*

6.3.2 Number of channels

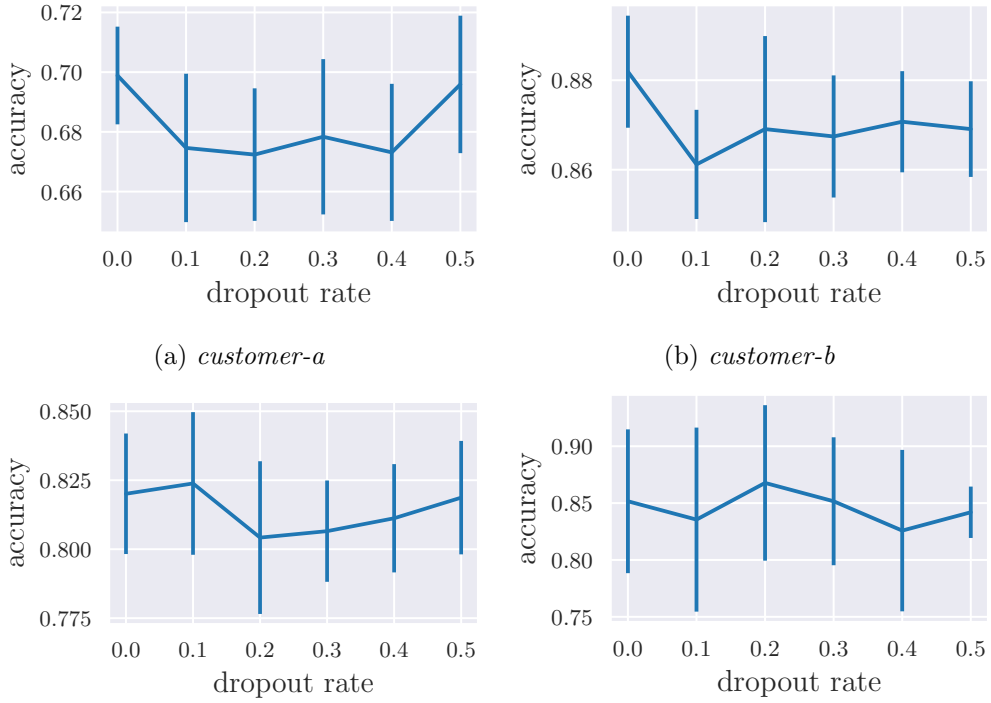


(a) *customer-a* (b) *customer-b*

(c) *customer-h* (d) *customer-w*

Figure 6.8 *Accuracies in percent for comparing accuracies depending on selected number of channels. This hyperparameter is set in all convolutional layers. Averaged over 10 runs. 16 channels were selected as hyperparameter for later usages.*

6.3.3 Dropout rate



(a) *customer-a* (b) *customer-b*

(c) *customer-h* (d) *customer-w*

Figure 6.9 *Accuracies in percent for comparing accuracies depending on dropout rate. Dropout is used on the output of graph convolution layers. Averaged over 10 runs. Dropout rate of 0.2 selected as hyperparameter for later usages.*

Bibliography

[1] Frank L Härte. *Efficiency analysis of packaging lines*. Delft University Press, 1997.

[2] Tobias Voigt. *Neue Methoden für den Einsatz der Informationstechnologie bei Getränkeabfüllanlagen*. Dissertation, Technische Universität München, München, 2004.

[3] Yves Dallery and Stanley B Gershwin. Manufacturing flow line systems: a review of models and analytical results. *Queueing systems*, 12(1-2):3–94, 1992.

[4] Prof. Dr. Willibald A. Günthner and Dipl.-Ing. Makrem Kadachi. Simulationsgestützte planung und nutzung von getränke-abfüllanlagen, 2001. `https://mediatum.ub.tum.de/doc/1188204/file.pdf`, last accessed at 9th January 2022.

[5] Frank L Härte. *Efficiency analysis of packaging lines*. Delft University Press, 1997.

[6] Dipl.-Ing. Axel Kather, Dipl.-Ing. Christian Haufe, and Dr.-Ing. Tobias Voigt. Ws pack specification of the interface content (part 2) version 08. *Weihenstephan Standards for Production Data Acquisition*, 2016.

[7] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`, last visited at 31 august 2020.

[8] John T Hancock and Taghi M Khoshgoftaar. Survey on categorical data for neural networks. *Journal of Big Data*, 7:1–41, 2020.

[9] Amirsina Torfi, Rouzbeh A. Shirvani, Yaser Keneshloo, Nader Tavaf, and Edward A. Fox. Natural language processing advancements by deep learning: A survey. *CoRR*, abs/2003.01200, 2020.

[10] Cheng Guo and Felix Berkhahn. Entity embeddings of categorical variables. *arXiv preprint arXiv:1604.06737*, 2016.

[11] Antoine Bordes, Jason Weston, Ronan Collobert, and Yoshua Bengio. Learning structured embeddings of knowledge bases. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*, 2011.

[12] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. Deep learning for time series classification: a review. *Data Mining and Knowledge Discovery*, 33(4):917–963, 2019.

[13] Anthony Bagnall, Jason Lines, Aaron Bostrom, James Large, and Eamonn Keogh. The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery*, 31(3):606–660, 2017.

[14] Mohammad Shokoohi-Yekta, Jun Wang, and Eamonn Keogh. On the non-trivial generalization of dynamic time warping to the multi-dimensional case. In *Proceedings of the 2015 SIAM international conference on data mining*, pages 289–297. SIAM, 2015.

[15] Alejandro Pasos Ruiz, Michael Flynn, and Anthony Bagnall. Benchmarking multivariate time series classification algorithms. *arXiv e-prints*, pages arXiv–2007, 2020.

[16] Donald J Berndt and James Clifford. Using dynamic time warping to find patterns in time series. In *KDD workshop*, volume 10, pages 359–370. Seattle, WA, USA:, 1994.

[17] Esmaeil Alizadeh. An illustrative introduction to dynamic time warping. `https://towardsdatascience.com/an-illustrative-introduction-to-dynamic-time-warping-36aa98513b98`, last visited at 16 october 2020.

[18] Xiaopeng Xi, Eamonn Keogh, Christian Shelton, Li Wei, and Chotirat Ann Ratanamahatana. Fast time series classification using numerosity reduction. In *Proceedings of the 23rd international conference on Machine learning*, pages 1033–1040, 2006.

[19] Prodromos E Tsinaslanidis, Achilleas D Zapranis, et al. *Technical analysis for algorithmic pattern recognition*. Springer, 2016.

[20] Tin Kam Ho. The random subspace method for constructing decision forests. *IEEE transactions on pattern analysis and machine intelligence*, 20(8):832–844, 1998.

[21] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[22] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning, 2019.

[23] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

[24] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020.

[25] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[26] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *European semantic web conference*, pages 593–607. Springer, 2018.

[27] Yuxiao Liu, Ning Zhang, Dan Wu, Audun Botterud, Rui Yao, and Chongqing Kang. Guiding cascading failure search with interpretable graph convolutional network. *arXiv preprint arXiv:2001.11553*, 2020.

[28] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. Deep learning for time series classification: a review. *Data mining and knowledge discovery*, 33(4):917–963, 2019.

[29] David Kriesel. *A Brief Introduction to Neural Networks*. 2007. `https://www.dkriesel.com/en/science/neural_networks?s[]=brief&s[]=introduction&s[]=to&s[]=neural&s[]=networks`, last accessed at 9th of January 2022.

[30] Zonghan Wu, Shirui Pan, Guodong Long, Jing Jiang, and Chengqi Zhang. Graph wavenet for deep spatial-temporal graph modeling. *arXiv preprint arXiv:1906.00121*, 2019.

[31] Yann N Dauphin, Angela Fan, Michael Auli, and David Grangier. Language modeling with gated convolutional networks. In *International conference on machine learning*, pages 933–941. PMLR, 2017.

[32] Acml2020 wsrl workshop. `https://wsl-workshop.github.io/`, last accessed at 20th November 2020.

[33] Zhi-Hua Zhou. A brief introduction to weakly supervised learning. *National Science Review*, 5(1):44–53, 08 2017.

[34] Charu C Aggarwal, Xiangnan Kong, Quanquan Gu, Jiawei Han, and S Yu Philip. Active learning: A survey. In *Data Classification: Algorithms and Applications*, pages 571–605. CRC Press, 2014.

[35] Elmar Haussmann, Michele Fenzi, Kashyap Chitta, Jan Ivanecky, Hanson Xu, Donna Roy, Akshita Mittel, Nicolas Koumchatzky, Clement Farabet, and Jose M Alvarez. Scalable active learning for object detection. In *2020 IEEE Intelligent Vehicles Symposium (IV)*, pages 1430–1435. IEEE, 2020.

[36] Simon Bachhuber. Increasing label efficiency in supervised classification for industrial application. Master's thesis, University Regensburg, 2020.

[37] Kunkun Pang, Mingzhi Dong, Yang Wu, and Timothy M Hospedales. Dynamic ensemble active learning: A non-stationary bandit with expert advice. In *2018 24th International Conference on Pattern Recognition (ICPR)*, pages 2269–2276. IEEE, 2018.

[38] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010.

[39] Karl Weiss, Taghi M Khoshgoftaar, and DingDing Wang. A survey of transfer learning. *Journal of Big data*, 3(1):9, 2016.

[40] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? *arXiv preprint arXiv:1411.1792*, 2014.

[41] W I Schollhorn, Patrick Hegen, and Keith Davids. The nonlinear nature of learning-a differential learning approach. *The Open Sports Sciences Journal*, 5(1), 2012.

[42] Yu Zhang and Qiang Yang. An overview of multi-task learning. *National Science Review*, 5(1):30–43, 2018.

[43] Kim-Han Thung and Chong-Yaw Wee. A brief review on multi-task learning. *Multimedia Tools and Applications*, 77(22):29705–29725, 2018.

[44] Alexander Ratner, Braden Hancock, Jared Dunnmon, Frederic Sala, Shreyash Pandey, and Christopher Ré. Training complex models with multi-task weak supervision, 2018.

[45] Stefan Walter Flad. *Neue Methoden zur Effizienzanalyse in verketteten Produktionslinien am Beispiel von Getränkeabfüllanlagen.* PhD thesis, Technische Universität München, 2018.

[46] Axel Kather. *Fehlerlokalisierung in verketteten Produktionslinien am Beispiel von Lebensmittelverpackungsanlagen.* PhD thesis, Technische Universität München, 2009.

[47] Tobias Voigt, Stefan Flad, and Peter Struss. Model-based fault localization in bottling plants. *Advanced Engineering Informatics*, 29(1):101–114, 2015.

[48] Rich Caruana. *Multitask Learning*, pages 95–133. Springer US, Boston, MA, 1998.

[49] Sebastian Ruder. An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098*, 2017.

[50] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[51] Matthias Dorfer, Bernhard Lehner, Hamid Eghbal-zadeh, Heindl Christop, Paischer Fabian, and Widmer Gerhard. Acoustic scene classification with fully convolutional neural networks and i-vectors. *DCASE2018 Challenge*, 2018.

[52] Matthias Fey. pytorch-geometric, 2021. `https://pytorch-geometric.readthedocs.io/en/latest/modules/nn.html#torch_geometric.nn.conv.RGCNConv`, last accessed at 17th December 2021.

Danksagung

An dieser Stelle möchte ich allen herzlich danken, die zum Zustandekommen dieser Arbeit beigetragen haben. Insbesondere:

- Meinen Eltern und Großeltern, die mir die Möglichkeit gegeben haben meine Ausbildung bis zur Promotion verfolgen zu können und mich immer unterstützt und motiviert haben.

- Prof. Dr. Elmar Lang für die ausgezeichnete Betreuung, die interessanten Diskussionen und die uneingeschränkte Hilfsbereitschaft bezüglich aller Themen. Das macht die Arbeitsgruppe zu einem familiären Umfeld in dem man sich gerne aufhält.

- Der Syskron X bzw. Syskron GmbH für die Bereitstellung des Themas und der fachlichen Betreuung. Allen Syskron- und Krones Kollegen ein herzliches Dankschön für alle mit großer Selbstverständlichkeit gegebenen Hilfestellungen und die angenehme Zusammenarbeit. Besonders unter direktem Druck von Kunden und Zeitplänen ist das sehr bemerkenswert.

- Meiner Freundin Thea für ihre Unterstützung, Aufmuterung und das Verständnis für den "zerstreuten Physiker" in der letzten Zeit des Verfassens dieser Arbeit. Ich hoffe, das bessert sich wieder ;).

- Meinen Bürokollegen und Masteranden and der Uni für die gute Zusammenarbeit und die Freundschaft, die auch über die Uni hinaus geht. Ich bin mir sicher, es gibt sonst nirgends eine so effiziente Umwandlung von Keksen in Ideen!

- Der DataScience-Community in Regensburg. Was gibt es schöneres als gute Ideen für die Promotion beim Beachvolleyball sammeln zu können?

- Allen Freunden und Bekannten. Ihr macht es mir sehr leicht das Leben zu genießen und in der Freizeit die Akkus wieder aufzuladen.