Design of a decision making algorithm to support operators in a real-time production environment

S.N.J. Koot

4240383

Master thesis

2020.MME.84

Heineken®

TU Delft

Design of a decision making algorithm to support operators in a real-time production environment

by

S.N.J. Koot

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Monday August 31, 2020 at 13:00.

Student number:	4240383	
Report number:	2020.MME.8444	
Project duration:	January 1, 2020 – August 31, 2020	
Thesis committee:	Dr. ir. D.L. Schott	TU Delft, chair
	Dr. ir. W.W.A. Beelaerts van Blokland	TU Delft, supervisor
	Dr. ir. J.F.J. Pruijn	TU Delft
	Ir. J.J. Stolk	Heineken

An electronic version of this thesis is available at http://repository.tudelft.nl/.

TU Delft

Preface

In front of you, you find my master thesis to obtain the degree of Master of Science at the Delft University of Technology. It is the result of eight months of hard work in collaboration with Heineken, and specifically Alken-Maes brewery in Belgium.

First of all, I would like to thank the graduation committee of the University for their support in this research. I would like to thank my daily supervisor, Wouter Beelaerts van Blokland, for his continuous support and challenging ideas. Finding the right goal has not always been easy during this project, but Wouter managed to challenge me during all our (online) meetings to set the right goal. Also, thank you to my chair of the thesis committee, Dingena Schott, for her supervision over the research and her input during our meetings.

Secondly, I would like to thank Heineken for allowing me to execute this research within their beautiful company. Although we couldn't finish the project together, I would like to thank Ilaria Tropea for her warm welcome at the global department of Heineken and her supervision over the project in the first months. I am glad Joris Stolk allowed me to use the brewery of Alken-Maes as a perfect use-case for this research. He provided me with the right support, and together with Nick Damen, they helped me improve the quality of this master thesis.

Finally, I would like to thank my family and friends for their continuous support during my studies. My parents for providing all the opportunities needed, and together with my sister for all the joy in our lives. My friends for all the fun we had in our time in Delft and beyond. Last but not least, I would like to thank my girlfriend for her support during the past years, specifically during this master thesis.

After eight wonderful years at the Delft University of Technology, my journey in Delft will end. I look back on an extremely educational and fun period of my life, and I would like to thank the University for all the opportunities they have offered me.

S.N.J. Koot
Den Haag, August 2020

Summary

The goal of Heineken, as the second biggest beer label in the world, is to grow in the competitive beer market. The growth can partly be established by improving production processes to increase productivity in its breweries. Production processes suffer under unplanned downtime occasions caused by unexpected errors. The improvement of production processes benefits from the introduction of Industry 4.0 and all its tools. Therefore, the focus of this research is to improve operational productivity by using intelligence-driven intervention proposals in case of unplanned downtime occasions. The research is conducted within Alken-Maes brewery, which is one of the breweries of Heineken. The research is conducted on the Filler, one of the machines of the Alken-Maes packaging line.

The Systems Engineering methodology is applied in this research for the realization of a successful system. The user requirement is to reduce unplanned operational downtime by supporting operators to resolve errors faster to increase machine utilization. Unplanned downtime can be reduced by reducing the number of errors or by reducing the time to resolve errors. The focus of this research is on the latter.

It is necessary to have real-time access to the data of the Filler to reduce unplanned downtime in production processes. Real-time access to industrial equipment is described in theory about a digital twin. A digital twin consists of five elements from which a physical asset, a service model, the digital twin data, and the connections between these components are required in the context of this research. Together with the requirements of the service, these are defined as the systems requirements. The service requires to process input variables from the packaging line into a proposal for a countermeasure to resolve the error. Since the countermeasures are predefined, this is considered to be a multi-class classification problem in the field of operational production processes.

The service required an intelligent decision algorithm that can decide which countermeasures an operator has to execute. A neural network was chosen as this multi-class classifier. A neural network is suitable to solve these problems and showed good results on the specified problem. A training dataset is compiled to train the model, and the model is finally verified on a similar but smaller test dataset. After tuning the hyperparameters of the neural network, the neural network reached a prediction accuracy of 98,98%.

Then, the model is applied to data from the packaging line of Alken-Maes, specifically on the Filler. First, the data is prepared before it is fed into the model. Then, countermeasures for every occurred error are predicted by the model. These predictions are taken into account for the validation of the model. This is done by comparing the current situation with the proposed situation. In the current situation, it takes on average 6 minutes to resolve an error where it only takes 4,5 minutes in the proposed situation. Both situations are simulated on a period of 6 days where 1071 errors occurred and which led to 32 minor stops. The proposed situation achieves a reduction of 16 minutes of unplanned downtime compared to the current situation weekly. This indicates an improved production of 624.000 cans yearly. This corresponds to 22,2% reduction of unplanned downtime for downtime occasions where the operator has to resolve the problem. Implementing the service on each machine of the packaging line and considering every error of the machines further reduces the unplanned downtime.

The results of the service on the Filler are verified on a second machine of the packaging line, the Shrink Packer. A new dataset is created with 57 possible combinations, and the model is trained on this more extensive dataset. The results of the model on the Shrink Packer show comparable results. However, the absolute improvement is smaller due to the distribution of errors over the countermeasures.

The reduction of unplanned downtime can be increased when other machines of the packaging line are considered as well. Machine errors propagate to other machines of the packaging line. Error propagation leads to new errors on other machines caused by one machine. A modular neural network is

proposed to address each machine as a subproblem. An expert neural network solves all these subproblems, and the outputs are combined in a general neural network. This neural network proposes a sequence of countermeasures to the operator to solve the whole problem of the packaging line.

The proposed system, including the decision algorithm, improves operational productivity in case of unplanned downtime by proposing correct countermeasures. However, the research can be extended by applying the service on multiple machines which further improves operational productivity.

List of Figures

List of Tables

Contents

1 Introduction

1.1. Heineken global

Heineken is founded in 1864 by Gerard Adriaan Heineken, and now, almost 160 years later, it is the second biggest beer label in the world (Heineken, 2020c). Heineken brews over 300 international, regional, local, and speciality beers and ciders available in 190 countries (Heineken, 2019b). Approximately 85.000 employees are working for the Heineken brand in 165 different breweries around the world. The company-wide focus of Heineken is on growth to ensure their position in the beer market. Part of this growth is established by entering new beer markets all around the world. Examples of new breweries in entered beer markets are Mozambique in 2019 (Heineken, 2019a) and Mexico in 2018 (Heineken, 2018).

Heineken Global Supply Chain (GSC) enables, equips and empowers Operating Companies (OpCos) to beat the competition from grain to glass. GSC is responsible for supporting the production and distribution of beer by local breweries to fulfil beer demand around the world. Within Heineken GSC, Heineken Global Projects and Engineering (GP&E) designs, builds, extends, and revamps breweries to enable the Heineken N.V. to realize the company ambition. Part of this department is the Operational Set-Up (OSU) team. They are responsible for the non-technical set-up of breweries while the engineering team is responsible for the technical set-up of breweries. This research is conducted within the OSU team in collaboration with Alken-Maes brewery in Belgium.

Greenfield projects

The expansion of breweries involves designing and building new breweries in countries where Heineken did not have any market share or the market share is very small. Building new breweries from scratch is called ”greenfield projects”. The characteristics of greenfield projects are simple, there is no replacement or expansion of an existing brewery, but an entirely new brewery is designed and build at a new location.

In former greenfield projects, the responsibility of project managers stopped after the Operation and Maintenance Handover (OMH) which is the moment of the handover of the brewery from the supplier to the OpCo. Until that moment, experienced operators from the supplier are on-site to support the local operators in starting-up the equipment. During OMH, line performance is measured and if the performance is according to predefined targets the supplier’s operators leave the site and the responsibility of the brewery is transferred to the OpCo.

Heineken experiences a significant drop in performance after the OMH, which is illustrated in Figure 1.1. The responsibility of the project managers is extended to the first year of operation to address this drop in performance. On top of that, the OSU team is established to support OpCo’s in their set-up of the brewery.

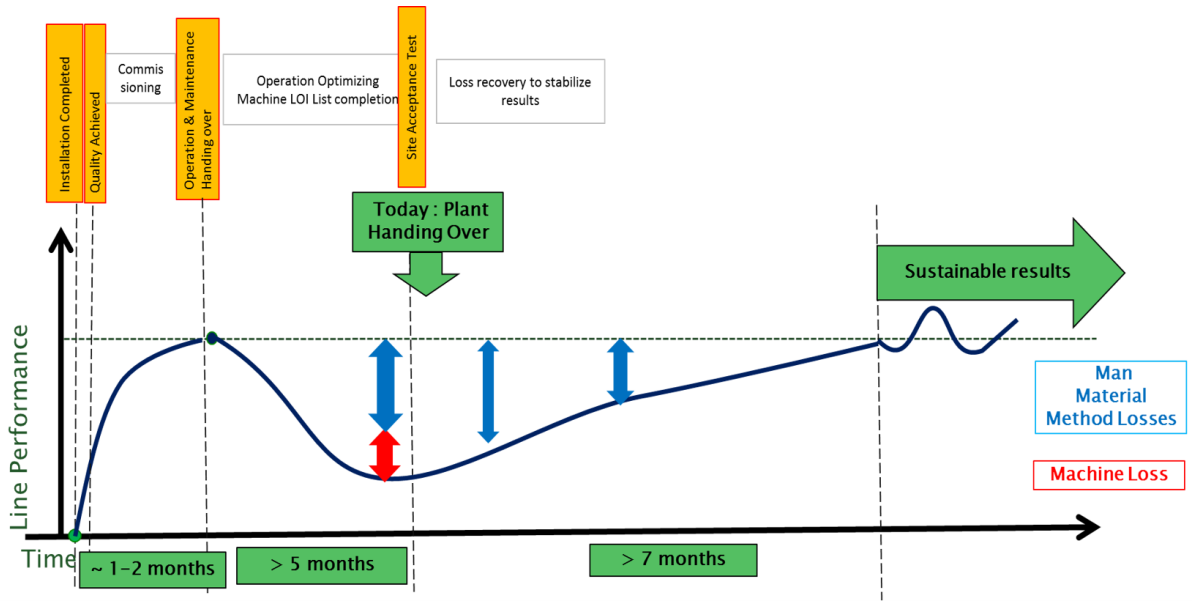


Figure 1.1: Illustrated graph of line performance of breweries (Heineken, 2020a)

The experience of Heineken is a disconnection between theory and reality. After installation, the theory about operating and controlling a line is available in the form of manuals, training, and instructions. However, it takes on average 6-9 months before an operator has gained enough experience to operate the brewery at the sustained performance (Heineken, 2020a).

1.2. Physical environment

The physical environment of a packaging line is different for every brewery, but there are many similarities. There is chosen to use the physical environment of the packaging line of Heineken's brewery in Belgium: Alken-Maes. Alken-Maes offers data and support and is the brewery that is investigated in this research. At this brewery, an Internet of Things (IoT) platform is installed, which makes it a suitable use case for this research. This research is conducted in the context of a stopped machine due to unplanned minor stops and speed losses. In the following section, the layout of the packaging line is globally described.

1.2.1. Canning line Alken-Maes

The packaging line of Alken-Maes is a so-called canning line. A canning line fills cans with different types of beer according to the demand. The canning line consists of multiple machines connected by conveyors, see Figure 1.2. A technical drawing of the layout of the packaging line of Alken-Maes is attached in Attachment B.

The packaging line in Figure 1.2 is an example packaging line but contains the same machines as the packaging line of Alken-Maes. The different machines in Figure 1.2 are described below according to the corresponding number:

1. **Depalletiser:** Unpacks the cans from the pallets and puts them on a conveyor.
2. **Filler:** Washes the cans, fills the cans with beer, and applies the lid.
3. **Tunnel Pasteurizer:** Heats the cans to 62,5 degrees Celsius to pasteurize the beer.
4. **Drying:** Dries the cans to prevent rusting and enables coding
5. **Shrink Packer and Tray Packer:** The Shrink Packer puts a shrink around cans and the Tray Packer places the cans in 6/8/12/24 cartons.
6. **Palletiser:** Places the secondary package on pallets and wraps the pallets.

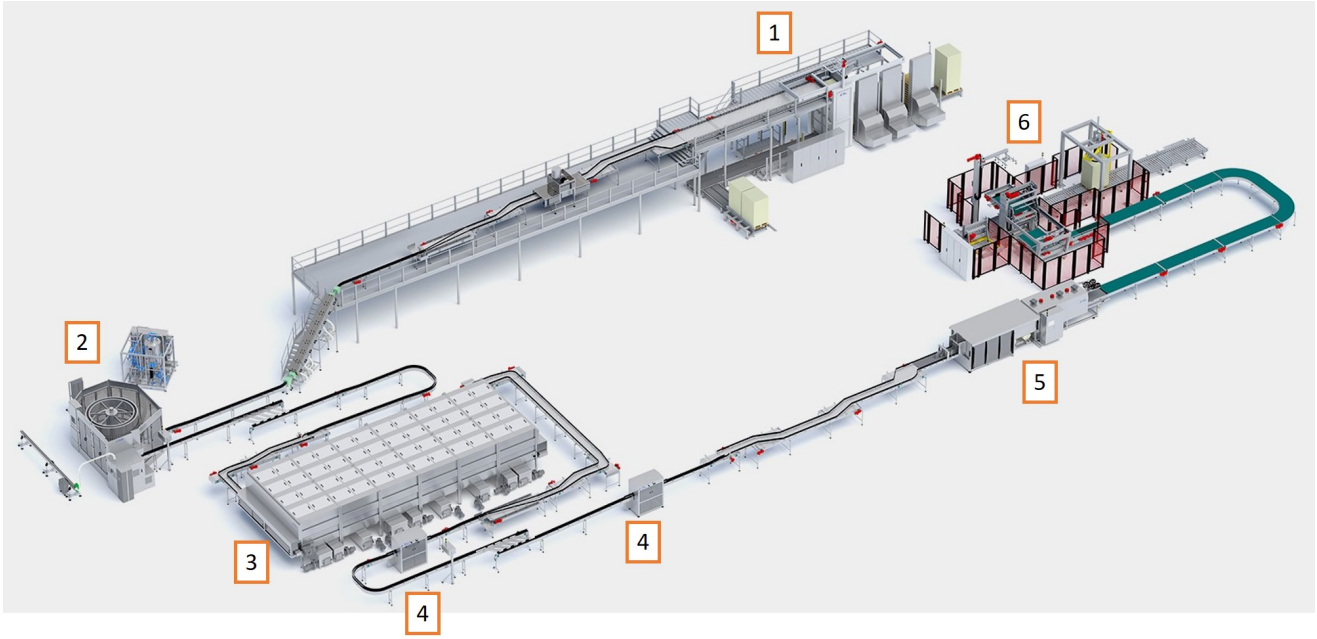


Figure 1.2: Example of canning line Sidel (Sidel, 2020)

The conveyors between the different machines have both a transportation function and a buffer function. They convey cans from one machine to another, and they store cans between the machines so a given machine can keep running for a short period, even if the upstream or downstream machine is temporarily down. Each machine has a nominal production speed but can run faster or slower to compensate for breakdowns.

1.2.2. Filling machine

Figure 1.3 shows a typical filling machine. The Filler first rinses the cans quickly after which they are loaded on a circular star-wheel, which fills the cans volumetrically. Subsequently, the Seamer places a lid on the can and closes the seam of the lid. Finally, the Fill Height Inspector measures the filling height of the can and removes faulty cans from the line. Before the cans go into the Tunnel Pasteurizer, the cans are turned upside down to check for possible leakages.

Minor stops and speed losses

In 24 days in May 2020, the Filler reported 6718 errors corresponding to approximately 280 errors each day. Not every error leads to a minor stop or speed loss, immediately. It depends on the error and the cause of the error.

Numerous causes, including errors, can cause minor stops and speed losses. During operation, an operator is responsible for operating the machine, which includes resolving errors. When an error occurs, the operator determines the cause of the error. The cause determines the countermeasure, which is executed by the operator. In several cases, the operator can choose up to five different countermeasures for one error.

In the first place, problem-solving is relying on standards and protocols. However, the experience and knowledge of the operator do influence the problem-solving. Especially in experienced OpCo's, the operator has many years of experience, which improves solving problems. However, in new OpCo's or greenfield breweries, there is a lack of experience and knowledge. Therefore, the time to resolve errors is much higher and consequently cost more money for the company.

1.2.3. Error propagation

Production lines, as the packaging line of Alken-Maes, are designed to be balanced along the line (Patti et al., 2008). Line balancing is reached when all the machines have a similar cycle time. However,

Figure 1.3: Starcan filling machine of Sidel (Sidel, 2019)

the high variability of the environment requires buffers to deal with any imbalance caused by machine errors or failures. In the time the buffers are used, it is possible to solve an error or failure which reduces any starvation and blockages of other machines (Battini et al., 2009). Starvation is an empty inlet buffer of a machine, and blockage is a full outlet buffer of a machine. Both situations require a machine to stop producing because the supply has stopped, or there is no place to store the produced goods, respectively.

Machine stops or unplanned downtime occasions require a machine to stop producing, which will lead to an increase or a decrease of the buffer upstream or downstream of the affected machine. If any downtime occasion consumes more time than the buffer upstream or downstream of the machine can compensate for, then the machine upstream or downstream will be affected and has to stop producing as well. An error that leads to a machine stop can propagate through the production line and affects the performance of other machines as well.

1.2.4. User requirements

Every time the equipment in a brewery experiences downtime, it is not able to produce any goods. Downtime which occurs unexpectedly or as a result of a failure is called unplanned downtime (Immerman, 2018). Therefore, the reduction of unplanned downtime is one of the main goals in a production environment. The reduction of unplanned downtime can be split into two focus areas, the first one is to reduce the number of errors and the second one is to reduce the time to resolve an error.

The first focus area is described in zero-defect production. Zero-defect producing means zero failures during operation, but not necessarily zero imperfections on the produced goods (Wang, 2013). The focus of this research is on the second focus area to reduce the time to resolve an error.

The user requirement is to reduce the unplanned operational downtime by supporting operators to resolve errors faster to increase machine utilization. Ideally, this requirement leads to zero-defect production. However, the focus in this research is on reducing the unplanned downtime from the moment the machine experiences speed loss or comes to a standstill.

When an error or failure occurs, an operator can choose which countermeasure has to be executed based on the state of the machine. In literature, this type of problem is described as multi-class classification problems (Terry-Jack, 2019). Multi-class classification problems predict something into one of *n* classes (e.g. ”yellow”, ”blue” or ”red”, etc.). More information about multi-class classification problems

is discussed in Section 2.6.1.

1.3. Research motivation

The combination of different machines and interaction with humans suits the field of Multi-Machine Engineering. Multi-Machine Engineering addresses the challenges to meet demands on efficiency, sustainability, and safety in complex environments (Delft University of Technology, 2019). It combines mechanical systems with real-time operation and distributed multi-machine interaction.

Different lean manufacturing methods such as zero-defect production (Wang, 2013) and Total Productive Maintenance (Nakajima, 1984; Pascal et al., 2019) describe the reduction of downtime in their methods. It is necessary to be able to take rapid decisions regarding resolving problems to reduce the unplanned downtime from a practical point of view and aim for the ultimate goal of zero-defect production (Miškuf and Zolotová, 2016). Reduction of unplanned downtime can be established by dealing with real-time and historical data from the packaging line, combined with introducing intelligent analyzing algorithms. Such an algorithm decides which countermeasure an operator has to perform based on several variables from the packaging line.

The physical environment of a packaging line is complex and consists of many variables. Decision-making algorithms in complex environments are part of artificial intelligence. Artificial intelligence is the broadest way to think about advanced computer intelligence (Garbade, 2018). Every machine which completes tasks based on specific rules that solve problems is called artificial intelligence. Machine learning (ML) is a subset of artificial intelligence (AI) and can be interpreted as the ability of computer systems to learn. ML enables machines to learn by themselves using available data and solve (prediction) problems. The decision algorithm needed in this research should be able to predict specific outputs based on the inputs. Due to the complex environment, it is necessary that algorithms can learn about the environment instead of using algorithms that are explicitly programmed.

Furthermore, the characteristic of a packaging line where conveyors connect multiple machines is typically a problem addressed in the field of Multi-Machine Engineering. The connection between the machines affects the performance of the entire packaging line and needs to be addressed as a complete system.

1.4. Research questions

This research studies how the operational productivity of an asset in a brewery can be improved by the design of a digital twin using artificial intelligence-driven intervention proposals. Therefore, the research question that will be answered in this research is:

How to improve the operational productivity by using artificial intelligence-driven intervention proposals in case of unplanned downtime occasions?

To answer the main research question, the following sub-questions are defined:

1. What are the requirements to enable real-time monitoring of a production line and what service is required to address the user requirement?
2. What are the design requirements of intelligent decision-making algorithms?
3. How can real-time decision making be applied to an industrial asset?
4. How can real-time decision making be applied in a production environment?

1.5. Research methodology

The problem of this research will be addressed according to the Systems Engineering methodology. Systems Engineering is an interdisciplinary approach for the realization of successful systems (Elm et al., 2008). This methodology is applied in this research to have a structured approach. Elm et al. (2008) define System Engineering as "... a robust approach to the design, creation, and operation of systems that consist of the identification and quantification of system goals and requirements, creation

of alternative system design concepts, performance of design trades, selection and implementation of the best design, verification that the design is properly built and integrated, and post implementation assessment of how well the system achieved its goals".

Systems Engineering is applied sequentially through all stages of system development and is focussed on technology components that can be engineered. Figure 1.4 shows the simplified V-model presented by Forsberg and Mooz (1994), which illustrates the sequence of events necessary for systems development according to the Systems Engineering methodology.

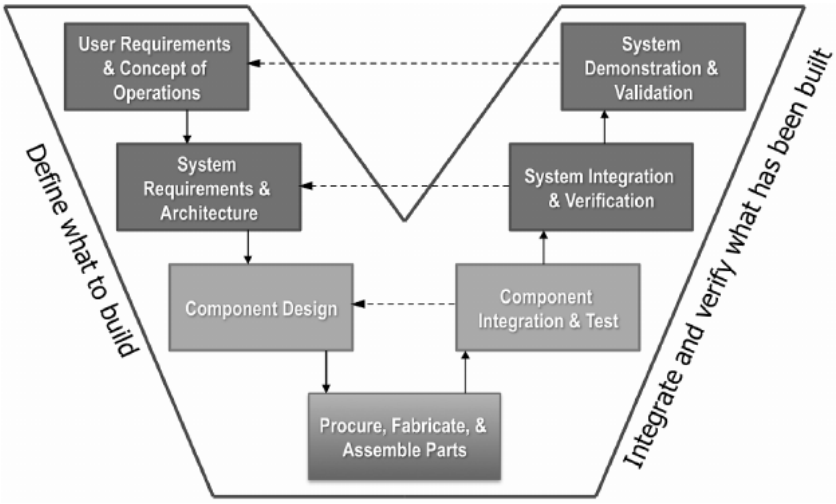


Figure 1.4: Overview of V-model of systems engineering (Sauser et al., 2010)

Time proceeds through the model from left to right, where the left side represents the decomposition of requirements and specifications. The right side represents the integration of parts, the verification, and the validation (Department of Defense, 2001). On the left side, the first step is to gather practical requirements from a user perspective. Then, these practical requirements are converted into system requirements. Thirdly, the different components are designed according to the system requirements. Finally, the components are built according to the component design. The first step on the right is to test and verify the components according to the design of the components. Secondly, the system is integrated into the real environment and verified according to the system requirements. Finally, the system is demonstrated and validated according to the user requirements defined in the first step of the process.

1.6. Research outline

The outline of this research is based on the Systems Engineering methodology described in the previous section. This structure enhances the quality of the research. In this chapter, the physical environment, the user requirements, and the concept of operation is described, which is the input for Chapter 2.

Chapter 2 describes the system requirements from a literature perspective. The combination of lean manufacturing methods and Industry 4.0 is discussed, which results in requirements of the infrastructure. The introduction of Industry 4.0 characterizes the infrastructure. Then, the service is discussed, which is part of this shift to a digital production environment. Finally, this chapter discusses the requirements of the service of decision-making algorithms.

Chapter 3 is about the design of such a decision making algorithm. Based on a model design framework introduced in Section 3.2, seven steps of developing a ML algorithm are discussed. These seven steps are the basis for the design of a decision-making algorithm, which is discussed in Chapter 4.

According to the Systems Engineering methodology, Chapter 4 discusses the development of the system components. The first step is to define and prepare the data. Then, a basic model is introduced,

which is the basis for improving the model. A large part of this chapter is the tuning of the different hyperparameters. When the model is completely optimized, the model is verified according to the evaluation methods described in Chapter 3.

Chapter 5 discusses the integration of the model into the physical environment. Data from the packaging line of Alken-Maes is extracted and used as input of the model. The predicted outputs based on these inputs are used to simulate the service in the environment of Alken-Maes. Furthermore, the model is applied on a second machine, the Shrink Packer, to verify the results of service.

Chapter 6 discusses both the system integration and verification as well as the system demonstration and validation. These two final steps of the Systems Engineering methodology are combined because the physical integration is not done due to the limits of this research.

Finally, Chapter 7 presents the conclusion of this research. Moreover, several recommendations are made for further research and Heineken.

	Systems Engineering Methodology	Sub-question
Chapter 1	User requirements	-
Chapter 2	System requirements	1
Chapter 3	Component design	2
Chapter 4	Development of the model	-
Chapter 5	Component integration	3
Chapter 6	System integration and validation	4
Chapter 7	Conclusion and recommendations	

Table 1.1: Overview research structure

2

Requirements of the service

The second step of the Systems Engineering methodology is to determine the system requirements and architecture. The user requirement is to reduce the unplanned operational downtime by supporting operators in resolving errors faster to increase machine utilization. The user requirement is taken into account to define the system requirements in this chapter. The system requirements are divided into two parts. First, the requirements of the digital infrastructure are discussed. The digital infrastructure is required to enable real-time monitoring of a physical asset. When data is accessible in real-time, the requirements are discussed for the support of operators in resolving errors faster.

In this chapter, the first sub-question is answered: *What are the requirements to enable real-time monitoring of a production line and what service is required to address the user requirement?* The first section describes different production optimization methods. The second section discusses the introduction of Industry 4.0 and the effects on production optimization. Section 2.3 discusses the requirements of a Digital Twin (DT) to offer production optimization. The fourth section discusses specifically the service a DT can offer in the form of ML algorithms. Section 2.5 discusses how ML algorithms are applied in the supply chain. Then, Section 2.6 describes different ML techniques and applications of multi-class classification problems in the literature. Finally, in Section 2.7 a conclusion is made which answers the sub-question addressed in this chapter.

2.1. Lean Manufacturing and Total Productive Maintenance

The focus of lean manufacturing and Total Productive Maintenance (TPM) is to reduce waste and downtime in any (production) process (van Ede, 2008). Heineken applies the technique of TPM in there breweries to reach this goal of perfect production. TPM is introduced by Seiichi Nakajima in 1971 to solve the maintenance problems of systems by giving operators more responsibility (Nakajima, 1984; Pascal et al., 2019). It can be defined as a "... production-driven improvement methodology that is designed to optimize equipment reliability and ensure efficient management of plant assets" (Ginder et al., 1995).

TPM is described by the Japan Institute of Plant Maintenance based on eight pillars, see Figure 2.1 (Ahuja and Khamba, 2008). The first four pillars focus on maintenance, also in favour of reducing downtime. The eighth pillar ”Development management” focusses partially on the start-up of new assets. One of the focus points is to use minimum time for deployment of new equipment (Nakajima, 1984; Sivaram et al., 2013). As the start-up of new assets is an issue for Heineken, the problem in this research is partly addressed in this pillar. Furthermore, the pillar ”Education & Training” is addressed by supporting operators in making the right decision.

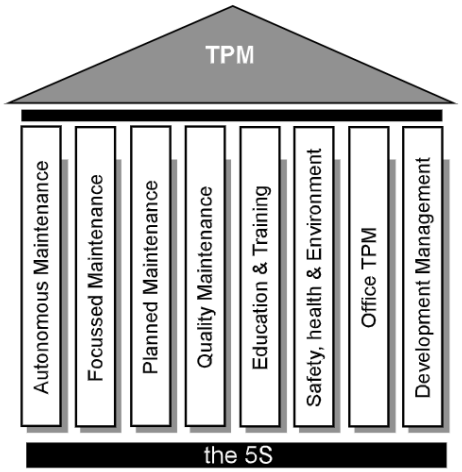


Figure 2.1: Eight pillars of TPM (Ahuja and Khamba, 2008)

Lean manufacturing and Industry 4.0

Besides lean manufacturing, Industry 4.0 is a relatively new research field that enables the handling of complex manufacturing systems. Industry 4.0 has found its origin in Germany in 2011, where the German government created a new vision for its industries (Roblek et al., 2016). It aims to increase the digitalization of production systems to improve the transparency of such systems (Mayr et al., 2018). Both lean manufacturing and Industry 4.0 are promising production paradigms to solve future manufacturing problems. Mayr et al. did research about these two developments and how they can support each other (Mayr et al., 2018).

Figure 2.2 shows a matrix of different lean manufacturing methods and Industry 4.0 tools. From this table, it can be concluded that digitalization contributes to different lean manufacturing methods. The described pillars of TPM are mainly supported by real-time computing in combination with a DT. However, the combination of real-time computing and a DT offers other lean manufacturing methods to become more intelligent as well.

Furthermore, lean manufacturing methods where the operator is centralized are also suitable to be improved by digitalization. In addition to Table 2.2, Goienetxea Uriartea et al. (2018) stated that also Andon can be reinforced by the implementation of DT and real-time computing methods.

Lean methods / I4.0 tools	JIT/ JIS	Hei- junka	Kanban	VSM	TPM 1*	TPM 2**	TPM 3***	SMED	VM 5 S	VM Zoning	VM Andon	Poka- yoke
Additive manufacturing (AM)	x					x		x				
Plug and play							x	x				
Automated guided vehicles (AGV)	x		x									
Human-computer interaction (HCI)			x	x	x				x	x	x	x
Virtual representation (e.g. VR, AR)	x				x			x	x	x		x
Intelligent bins	x		x									
Auto-ID	x		x	x	x			x	x	x		x
Digital object memory	x				x			x				x
Digital twin/simulation	x	x	x	x		x	x	x	x			
Cloud computing	x			x	x	x						x
Real-time computing	x	x	x	x	x	x	x	x	x	x	x	x
Big data & data analytics	x	x	x	x		x						x
Machine learning				x	x			x				x

* autonomous maintenance, ** planned maintenance, *** early product and equipment management

Figure 2.2: Combining Lean Manufacturing methods and Industry 4.0 tools (Mayr et al., 2018)

The user requirement is to improve unplanned operational downtime, which is also the focus of lean manufacturing methods. In combination with Industry 4.0, this offers opportunities for improvement of

manufacturing environments. The next section discusses the characteristics of Industry 4.0 and the required infrastructure.

2.2. Industry 4.0

The basis of digitalization in production environments is the introduction of Industry 4.0. Internet of Things (IoT) is the technological basis of Industry 4.0 (Ashton, 2009), which is discussed for the first time in 1982 with a modified coke vending machine at Carnegie Mellon University (Foote, 2016).

Industry 4.0 (also known as Industrial Internet of Things) is initially founded in 2011 by the German government who created a new vision for their industries (Roblek et al., 2016). Currently, it is attributed as the fourth industrial revolution. It is characterized by IoT, Cyber-Physical Systems and Internet of Services (Roblek et al., 2016), based on the developed communication technologies that allow communication between machines themselves.

The goals of Industry 4.0 are to achieve a higher level of automation, operational efficiency, and productivity (Thames and Schaefer, 2016). Besides, the five major features according to Roblek et al. (2016) are: ”digitization, optimization and customization of production; automation and adaptation; human-machine interaction (HMI); value-added services, and automatic data exchange and communication.”

Internet of Things

The phrase ”Internet of Things” was first used in 1999 by Kevin Ashton working for the MIT Auto-ID Center. However, the first machine was already connected to a network in 1982 (Ashton, 2009). IoT is part of the future internet and aims to collect information from- and offer services to a broad spectrum of physical things. IoT creates a virtual representation on the internet of everyday objects connected to the internet used in daily life. Every object has a unique identity and virtual address and communicates with other ”Things” without human intervention (Bessis and Dobre, 2014).

IoT was first discussed in the context of supply chain management. Nowadays, the definition is covering a wide range of applications like healthcare, utilities, transport, etc. (Sundmaeker et al., 2010). The definition of ”Things” has changed over the years as technology has improved, but the main characteristic of making a computer sense information without any human intervention remains the same. The development of the current internet is now based on interconnected objects which not only collect information from the environment and interacts with the physical world but also uses the already existing internet standards to provide services (Gubbi et al., 2013). Open wireless technology such as RFID, Wi-Fi, Bluetooth, actuator nodes, embedded sensors, and telephonic data services accelerated the development of IoT (Manogna and Dakannagari, 2016).

Recently, the brewery of Alken-Maes is equipped with an IoT system that connects different sensors of the packaging line with the internet. The system retrieves new data every second only when the state or value of the machine/sensor has changed. However, the number of sensors per machine which are available is limited and requires an expansion to create a complete overview of the packaging line.

2.3. Digital Twins

The development of IoT resulted in many more sensors and devices connected to the internet, data acquisition systems, and computer networks. Managing these interconnected systems between physical assets and computational capabilities is called Cyber-Physical Systems (Lee et al., 2015). The controlling software part of Cyber-Physical System is called a DT. The physical devices of Cyber-Physical Systems communicate with each other with the use of a software replica of the physical devices.

2.3.1. Characteristics of a Digital Twin

NASA was the first which presented the definition of a DT as ”an integrated multi-physics, multi-scale, probabilistic simulation of a vehicle or system that uses the best available physical models, sensor updates, fleet history, etc., to mirror the life of its flying twin. It is ultra-realistic and may consider one or more important and interdependent vehicle systems” (Shafto et al., 2010). In general, a DT is the virtual

representation of a physical object created in a digital way to simulate the behaviour of the physical object in a real-world environment.

A DT consists of two necessary aspects: data modelling and data analytics. First, it is essential to define a model with the information and data of the physical system. However, it is impossible to have an excellent virtual representation without real-time data of sensors insight into the physical object. A DT can be integrated into the whole manufacturing process, which creates a closed-loop from product design to smart maintenance, repair, and overhaul (MRO).

Opportunities for Digital Twins

The translation of the DT-concept from the aerospace field to other fields such as the robotics environment or manufacturing created many more functionalities. Many of them are already reported in the literature, amongst them (Macchi et al., 2018):

1. Improved maintenance decision making (damage/cracks prediction, material geometric/plastic deformation, and reliability modelling of physical systems).

2. System life cycle mirroring, supporting decision-making in different ways: i) predicting the system's performances/behaviour in the long term ii) granting digital data continuity along life cycle phases of the system iii) optimizing the control software of the system iv) simulating the organization of IoT devices.

3. Statistically based decision making and optimization, such as optimizing the system's behaviour by simulating it during the design phase or during life cycle phases.

In the context of supporting operators in deciding which countermeasure to perform, the latter two functionalities offer opportunities. In both functionalities, decision making is mentioned in several different ways. For this research, real-time decision making is necessary to offer support to operators when unplanned downtime occurs.

2.3.2. level of integration of a Digital Twin

The definitions DT, Digital Shadow, and Digital Model are often used in similar situations. However, these definitions differ in the level of integration. The differences are mainly in the connection of the model with the physical counterpart (Kritzinger et al., 2018).

Digital Model

The data exchange in a Digital Model is not done automatically but wholly manually, see Figure 2.3. A Digital Model might include simulations or any other models of the physical counterpart, which does not include automatic data exchange. Digital data can still be used to develop these models, but there is no real-time representation of the object.

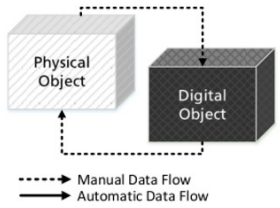


Figure 2.3: Representation of a Digital Model (Kritzinger et al., 2018)

Digital Shadow

When an autonomous one-way data flow exist from the physical object to the digital object, it is called a Digital Shadow, see Figure 2.4. A change of state of the physical object results in a change of the state of the Digital Shadow but not the other way around.

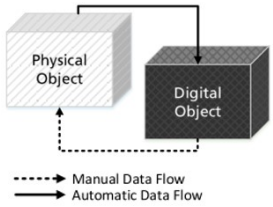


Figure 2.4: Representation of a Digital Shadow (Kritzinger et al., 2018)

Digital Twin
When the data flow between the physical object and the digital object is fully autonomous in both directions, it is called a DT. Figure 2.5 shows this characteristic of a DT. A DT is a controlling instance of the physical object. State changes of both the physical object and the DT lead to a change of state of the counterpart.

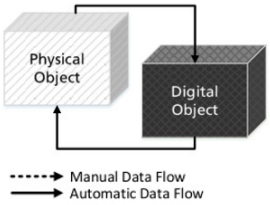


Figure 2.5: Representation of a DT (Kritzinger et al., 2018)

In the context of this research, a Digital Model is the minimum that is required. This research focusses on the development of a proof of concept instead of an entire implementation. Therefore, a Digital Shadow and a DT are not required. However, if the proposed service will be integrated, a DT is required because automatic data flow between the physical asset and the model, and between the model and the operator is necessary.

2.3.3. Five-dimension Digital Twin model

The three-dimensional DT model of Grieves (2016) is currently applied in most researches. However, new trends and demands are developed as the possibilities expand. Therefore, Tao et al. (2019a) presented a five-dimensional DT model based on the original three-dimensional model of Grieves to enable the use of a DT in more fields. Figure 2.6 shows the five-dimensional DT model.

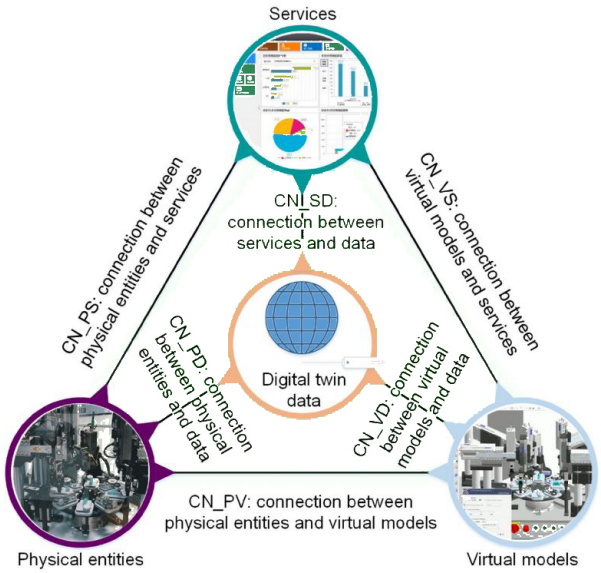


Figure 2.6: Five-dimensional DT model (Qi et al., 2019)

Physical entities in Digital Twin
The physical entities are the foundation of DT and are the starting point to work from. The behaviour of physical entities is simulated by the virtual models created by the DT (Tao et al., 2018). The physical entities may consist of industrial machines, products, devices, or even operators working with the entities. Physical laws are the basis of these entities and uncertain environments are considered as well.

In the physical environment of this research, the physical entity is a combination of the Filler of the packaging line in Alken-Maes and the operators operating the packaging line. The communication between the physical entity is two-sided because real-time data is collected from the Filler, and decisions are communicated back to the operator.

Virtual models in Digital Twin
The virtual models of a DT are faithful replicas of the represented physical entity, with corresponding physical geometries, properties, behaviours, and rules (Tao and Zhang, 2017). First, the 3D geometric

models describe the size, shape, tolerance, and structural relations. The physics models describe the physical phenomena of the entities. Thirdly, the behaviour model describes the behaviour of the entities against changes in the external environment, such as performance degradation and state transition. Finally, the DT is equipped by the rule model with logical reasoning based on historical data or expert knowledge.

Virtual models are an important asset of a DT. However, this research does not focus on the development of virtual models. The packaging line of Alken-Maes is equipped with a dashboard where only real-time values are displayed, but this dashboard does not contain any 3D models. Virtual models will not contribute to the user requirement of this research as the focus is on providing a service to support operators in resolving problems faster.

Digital Twin data

Data completely drives the DT (Qi et al., 2018). Data can be obtained from different sources, such as physical entities or virtual models, reflecting the simulation results. Other data sources can be services or expert knowledge. Data from different sources can also be combined to generate new usable data.

The data layer is the connection between the cyber layer and the physical layer. It receives all data from the physical assets and process and convert the data into a machine-readable form and makes readable data accessible for the cyber layer. According to Zheng and Sivabalan (2020) the following aspects require attention:

- The data needs to be encrypted.
- The collected data must be transformed into a standard, machine-readable format to process the data efficiently.
- Communication protocols follow the OSI model.
- The identified data packets are presented in a machine-readable data format such as *.txt* and *.csv* file, which is accessible from the cloud.

The data from the packaging line of Alken-Maes is collected by connecting the computers to an IoT-kit. This IoT-kit transforms the local data into globally accessible data. Part of the IoT-kit is encrypting the data, and the data is transformed in *.csv* files.

Services in Digital Twin

The importance of service in all aspects of modern society is more and more considered by enterprises (Tao and Qi, 2019a). First, application services concerning, optimization, simulation, diagnoses and prognosis, monitoring, verification, etc., are provided to users by a DT. Furthermore, several third-party services are needed in the process of building a functioning DT, like knowledge services, data services, algorithms services, etc. Thirdly, various platform services are required for the operation of a DT.

Connections in Digital Twin

Connections between the physical entities, virtual models, services, and data are crucial in the operation of a DT. Connections enable required information and data exchange for real-time simulations, operations, and analysis. According to Figure 2.6, there are six connections (Qi et al., 2019):

1. Connection between physical entities and virtual models (CN_PV)
2. Connection between physical entities and data (CN_PD)
3. Connection between physical entities and services (CN_PS)
4. Connection between virtual models and data (CN_VD)
5. Connection between virtual models and services (CN_VS)
6. Connection between services and data (CN_SD)

Infrastructure requirements
From the five-dimensional DT model, four requirements are defined. These four requirements are:

- Access to sensors of physical asset
- An online database with data from the physical asset and models
- Connections between the components
- Service model that can make intelligent decisions

The fifth component of the five-dimensional model is a virtual model that is not required for the implementation of intelligent services. The first three requirements are focussed on the infrastructure of the different components. These are assumed to be available in the coming three chapters and will be discussed once more in Chapter 6. The focus of the remainder of this research is on the last requirement, a service that provides intelligent decisions to operators in case of unplanned downtime.

2.3.4. Digital Twin and services

From the previous sections can be concluded that the development of real-time services is part of implementing a DT. The physical environment of the packaging line in Alken-Maes is already equipped with data extraction hardware which is essential in the five-dimensional DT model. This data enables the development of other models, such as service models.

The objective of this research is to provide a digital service in a production environment based on real-time data. As mentioned in the previous section, this is typically provided in the service domain of a DT. Service plays an increasingly more critical role in manufacturing as manufacturing evolves toward socialization and servitization (Lightfoot et al., 2013). The potential of a DT can be fully released through services in the concept of everything as a service.

Past research has proven that a wide variety of high-quality products and low manufacturing and distribution costs characterizes successful enterprises (Esposito et al., 2016; Ferreira et al., 2017). The implementation of a DT-based framework enables enterprises to reduce the cost of inefficient production (Min et al., 2019).

Service-Oriented Smart Manufacturing
As described in Section 2.3.3, virtual space consists of virtual models and services, both based on the physical entity and the corresponding data. After data mining and analysis is it possible to generate knowledge and rules. This knowledge and rules can be used to make autonomous decisions and control and execute the physical space (Tao and Qi, 2019b).

In the situation of complex environments, ML algorithms are commonly used in mathematical models (Tao et al., 2019b). ML algorithms are not programmed explicitly but are trained by providing a large dataset so they can learn how to make decisions based on this provided dataset. In this way, the model can learn from the changing environment. As mentioned, ML is a subset of AI, and AI is applied in many different fields, also in workflow management and manufacturing environments. The following section discusses different researches where AI is implemented in the field of workflow management and manufacturing environments.

2.4. Business process management

The problem considered in this research is about proposing intervention measures to operators, which is in the field of workflow management and business process management (BPM). BPM is studied for many years and is defined as: ”all efforts in an organization to analyze and continually improve fundamental activities such as manufacturing, marketing communications and other major elements of company’s operations” (Trkman, 2010). The Task-Technology-Fit theory can best describe the technology in BPM. Task-Technology-Fit describes that the implementation of digital services in a process is more likely to work out positively if the proposed service matches the tasks of the user (in this case, the

operator) (Goodhue and Thompson, 1995). The automation of tasks in a business process improves the performance of business activities by faster execution with a better result (Nikolaidou et al., 2008).

Traditional approaches to BPM and workflow management use predefined logical procedures of activities to model and manage the process (Wang and Wang, 2005). A complete list of activities and paths is available, and the particular path to follow is specified. This works well for simple and stables processes. However, it does not fulfil the requirements for complex processes due to a lack of flexibility (Van der Aalst and Kumar, 2003).

2.4.1. Knowledge-based approaches

Knowledge-based approaches and adaptive workflow techniques are considered in multiple types of research in a way to provide flexibility and adaptability (Kammer et al., 2000; Narendra, 2004). These techniques improve the conventional workflow models by proposing alternative paths as a solution for solving problems. These techniques offer limited flexibility and adaptability compared to conventional methods.

Another applied technique in BPM is the Event-Condition-Action approach in which rules are used to enforce additional operational control of processes (Wang and Wang, 2005). Event-Condition-Action uses rule/knowledge-based workflow systems where the rules make databases react to certain events. The traditional workflow process is extended with rules which may support run-time processes when certain events trigger the model. The application of rules in BPM offers more flexibility and adaptability than conventional models. However, when dealing with complex situations where more uncertainty and interactions are involved, this approach is not sufficient.

2.4.2. Cognitive approach

Simon et al. addressed the challenge of a changing and complex environment and stated that it requires adaptive mechanisms to handle unstructured problems (Simon and Mintzberg, 1977). Unstructured systems do not contain routine procedures for dealing with problems. Cognitive science has provided a way to encounter these problems from a human thinking perspective. During the interaction between the environment and the asset, information is combined and used as input for a cognitive algorithm or heuristic method (Newell, 1990).

Wang and Wang (2005) presented a cognitive approach that monitors the environment and makes real-time decisions for an unstructured system. This cognitive approach manages activities by knowledge and rules based on the real-time environment. Underlying process logic enables the model to make real-time decisions about tasks based on the state of the environment. The research of Wang et al. also compared the cognitive approach against conventional workflow approaches, see Table 2.1. Traditional workflow technologies are good at task routing without many operational constraints. However, workflow approaches with Event-Condition-Action rules are also successful in reacting to certain events. Both methods are not able to consider more dynamic and complex processes as these processes can not be encountered with routine procedures. The cognitive approach shows better results in real-time decision-based control of the process. It applies to complex and dynamic domains, such as e-commerce, Manufacturing Resource Planning, Supply Chain Management, etc.

Features	Workflow approach based on predefined process schema	Workflow approach based on predefine process schema with ECA rules	Cognitive approach
Task routing	++	++	+
Operational constraints	+	++	++
Reaction to certain events	-	+	++
Continuous perception of environment	-	-	+
Decision-based control of process	-	-	+
Manipulation of business strategies	-	+	++
Support of interactive tasks	-	-	+

Table 2.1: Comparison of cognitive approach with workflow approach (Wang and Wang, 2005)

2.5. Artificial intelligence in supply chain

AI is studied for a long time, and the potential of AI in many domains is proven. Also, supply chain management is studied in combination with AI. Different AI applications are discussed by Min, such as transportation network design, purchasing and supply management, demand planning, and forecasting, order-picking problems and customer relationship management (Min, 2010).

Also, Patel et al. (2018) studied the implementation of AI techniques together with IoT platforms in supply chain environments. Patel et al. mention the importance of integrating IoT platforms in manufacturing assets to develop intelligent and smart services to reach smart manufacturing. One of the use cases of the research of Patel et al. is about the integration of such a system to interlink sensor measurements, manufacturing execution systems, business processes, workflow, etc. IT systems already capture most of these data, but it is not accessible without significant manual effort. The objective of this use case is to support real-time decision-making by using all available data. An example scenario is a technician who must quickly troubleshoot a physical asset that encounters a problem. This technician will be helped if a system would support him with a summary of the problem, suggested manuals and necessary tools and parts to resolve the problem.

AI is also applied in real-time control of assets in the manufacturing industry. The research of Rossit et al. (2019) proposes a data-driven approach for scheduling in the manufacturing industry based on real-time data from the system. The goal of the research of Rossit et al. is to make scheduling decisions earlier in time to encounter problems earlier. The research is proposing a Cyber-Physical System approach, combined with a data-driven engine which uses big data techniques to make decisions. The research also mentions the importance of real-time access to data from a machine to adapt to the decisions rapidly. Traditionally, scheduling problems are solved by using data parameters as processing times, delivery dates, preparation times, etc. combined with negative impacts like downtime or quality losses. Then, schedules are adapted when situations in the physical environment change (Kis and Pesch, 2005). This approach is event-driven since the schedules changes when an event has occurred or afterwards to restate the scheduling program (Ouelhadj and Petrovic, 2009). The research of Rossit et al. proposes an architecture in which real-time data of a machine is available to all control systems of the machine. Now, this information is available in real-time, smart algorithms can predict future states of the machine and adapt the schedules according to these predictions. Decisions can be made before the event has occurred, which is typically the moment of decisions in event-driven systems.

The discussed researches show the potential of AI in the supply chain and manufacturing industry. The combination of the cognitive approach in BPM and the study of Patel et al. is interesting because this combination addresses the user requirement of this research. However, this research goes beyond the research of Patel et al. because it not only uses real-time information to decide where the problem occurs but will also propose countermeasures to resolve the problem. A decision-making algorithm is required for this service to support operators in resolving problems faster.

2.6. Machine learning as a service

Many theoretical and empirical pieces of research have proven that ML including big data-based approaches as data mining, artificial neural networks (ANN) and pattern recognition are promising in the manufacturing industry (Carbonell et al., 1983; Kateris et al., 2014; Köksal et al., 2011; Wen et al., 2012). The type of ML algorithm needed is mainly based on the type of problem. This section discusses different ML algorithms in combination with different studies on these algorithms.

2.6.1. Machine learning styles

ML models can be divided into three different groups: supervised, unsupervised, and semi-supervised ML algorithms (Brownlee, 2016b). This division is based on the type of dataset available.

Supervised machine learning

Supervised ML algorithms are the most occurring models, see Figure 2.7. Supervised learning is where input variables and output variables (labels) are combined to learn the model how to get the right labels based on the input variables (Brownlee, 2016b). It is called supervised learning because the learning process can be seen as a teacher supervising the learning process. The correct output label is known corresponding to the input variables, and the algorithm iteratively makes predictions on this data while the 'teacher' corrects the model. Supervised models can be further divided into two groups, regression and classification problems (Soni, 2018):

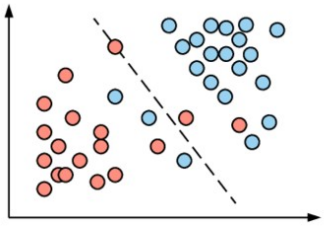


Figure 2.7: Supervised learning example (Qian et al., 2019)

- **Classification:** When the labels are a category, like "red" or "blue", the problem is defined as a classification problem.
- **Regression:** When the output variable is a real value, the problem is defined as a regression problem.

Unsupervised machine learning

When the problem has input variables but no output labels, it is called unsupervised learning, see Figure 2.8. Unsupervised ML algorithms try to find the underlying structure of the data to learn more about the data (Brownlee, 2016b). It is called unsupervised learning because the desired output is not known, and the model searches and presents interesting structures in the data itself. Also, unsupervised learning problems can be further divided into two groups, clustering, and association:

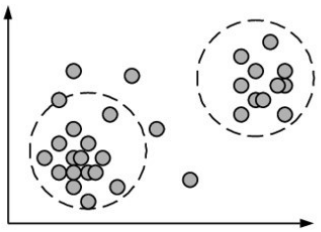


Figure 2.8: Unsupervised learning example (Qian et al., 2019)

- **Clustering:** If groupings of data need to be discovered, it is called clustering.
- **Association:** If the goal is to discover rules that describe large portions of data, it is called association.

Semi-supervised machine learning

The third ML category is the semi-supervised ML category, where a large amount of input variables is available, but only some of the data is labelled (Brownlee, 2016b). Many real-world ML problems are semi-supervised ML problems because the workload is too high to label all data. Unsupervised learning algorithms can be applied to the dataset to discover and learn the structure in the input variables. It is also possible to use supervised learning techniques to predict the label of the unlabelled data and use that in the supervised learning algorithm train the model on this data.

The conclusion from this section is that the problem in this research is a supervised machine learning problem. Furthermore, it is a classification problem as a training dataset is composed of inputs with the required labels of countermeasures. The requirement for a classification problem is a supervised training/learning process to enable the algorithm to learn to predict the right label based on the input. Moreover, the classification problem is a multi-class classification problem. The characteristics and applications of multi-class classification problems are discussed in the following section.

2.6.2. Multi-class classification problems

Classification problems can be divided into binary and multi-class classification problems. Binary classification involves predicting whether something is of one of two classes (e.g. ”on” or ”off”, ”black” or ”white”, etc.) (Terry-Jack, 2019). Multi-class classification problems predict something into one of *n* classes (e.g. ”yellow”, ”blue” or ”red”, etc.). Figure 2.9 shows the difference between the two, where the number of classes *n* in a multi-class classification problem can be of an infinite amount. The problem in this research considers multiple inputs and output classes. Therefore, this problem is defined as a multi-class classification problem.

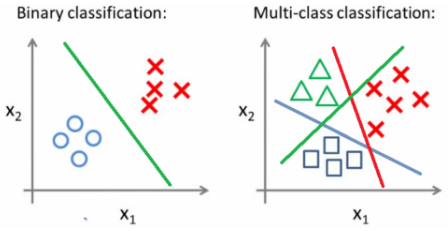


Figure 2.9: Binary classification and Multi-class classification (Terry-Jack, 2019)

2.6.3. Multi-class classification algorithms

Multi-class classification algorithms can be divided into different classes. Algorithms are often grouped by similarity in their function or by their learning style (Brownlee, 2019). The different learning styles are already discussed in Section 2.6.1, in this section, the different multi-class classification algorithms are grouped by their function and are discussed briefly.

Instance-Based Algorithms

Instance-Based Algorithms are decision algorithms with examples of training data that are important or required to the model (Brownlee, 2019). These methods are built on a database of example data, and new data is compared to the database to determine the best match to make a prediction. Frequently used Instance-Based Algorithms are k-Nearest Neighbor (kNN) and SVMs.

kNN is one of the oldest non-parametric classification algorithms (Aly, 2005). This means it makes no assumptions on the underlying data (MissingLink, 2019). The strengths of kNNs are the simple implementation and understanding of the model and the effectiveness of low dimensionality problems. However, kNNs are not suitable for high dimensionality problems and are computationally intensive. An example of kNNs is shown in Figure 2.10.

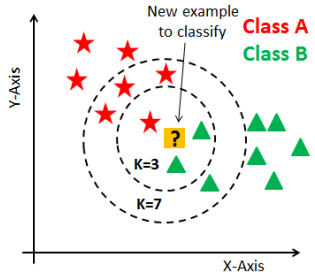


Figure 2.10: k-Nearest Neighbor algorithms (Navlani, 2018)

Basic SVMs supports only binary classification problems but extensions of SVMs supports also multi-class classification problems (Aly, 2005). SVM are among the most robust and successful classification algorithms. However, SVMs in multi-class classification problems can result in large optimization problems, which may be impractical. An example of SVMs is shown in Figure 2.11.

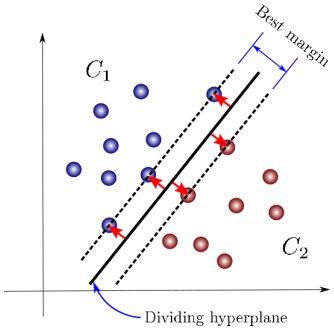


Figure 2.11: Support Vector Machine algorithms (Carrasco, 2017)

Decision Tree Algorithms

Decision Trees are powerful classification algorithms and can handle multi-class classification problems naturally (Aly, 2005). Decision Tree methods construct a model where decisions are made based on the actual values of the input variables (Brownlee, 2019). The model is similar to a tree where a variable is processed trough all nodes until a predictions decision is made. Decision Trees are often fast and accurate and are trained in classification and regression data. The strengths of Decision Trees are the ability to model complex decision processes and the intuitive interpretation of the results (MissingLink, 2019). The weakness of Decision Trees is relatively easy overfitting of data. Overfitting of data appears when a model is not able to generalize well from training data to unseen data.

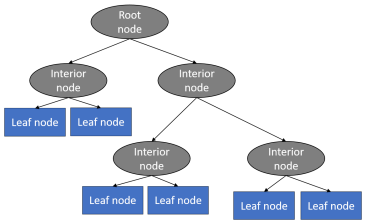


Figure 2.12: Decision Tree Algorithms (Brownlee, 2019)

Artificial Neural Network Algorithms

ANN are naturally suitable for solving multi-class classification problems (Aly, 2005). ANNs are models based on biological NNs like our brain (Brownlee, 2019). ANNs are commonly used for classification and regression problems and contains hundreds of algorithms and variations. Frequently used ANNs are (multilayer) Perceptrons and Feed Forward NNs. To address a multi-class classification problem, the network has multiple neurons in the output layer according to the number of classes, see Figure 2.13. NNs are very effective for high dimensionality problems and are able to deal with complex relations (MissingLink, 2019). The disadvantages of NN are the theoretical complexity and difficult implementation.

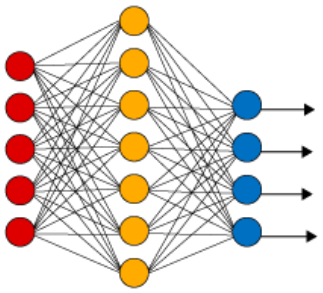


Figure 2.13: Artificial Neural Network Algorithms (Williams, 2018)

Deep Learning Algorithms

Deep Learning algorithms are an update of ANNs, see Figure 2.14. They are able to build much larger and more complex NNs and are able to work with very large datasets of labelled analog data (Brownlee, 2019). Frequently used Deep Learning algorithms are Convolution NNs, Recurrent Neural Networks and Long Short-Term Memory Networks.

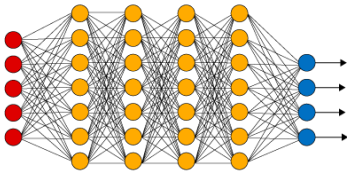


Figure 2.14: Deep Learning Algorithms (Williams, 2018)

In the end, several algorithms suit the goal of solving a multi-class classification problem. Some algorithms naturally suit the requirements of a multi-class classification problem, and others have to be modified to suit these requirements. The following section discusses the application of these different algorithms.

2.6.4. Application of multi-class classification algorithms

Many research is done about multi-class classification problems. However, the amount of researches considering multi-class classification problems in the field of real-time operations is little. Research of multi-class classification problems is more focussed on text and speech recognition. Multi-class classification algorithms are used in many real-world problems like speech recognition, face recognition, medical diagnosis, fraud detection, and fault detection (Bhardwaj et al., 2016). Most of the work on multi-class classification is related to text categorization (Paolanti et al., 2018).

Research is done on different strategies among which direct multi-class classifiers such as NNs and Decision Trees, an ensemble of binary classifiers, and an ensemble of one-class classifiers are represented dominantly. The two ensemble methods use a decomposition of the original multi-class problem into several smaller subproblems (Kang et al., 2015). The two conventional approaches of the second strategy are the one-versus-one and one-versus-rest approach (Lorena et al., 2008; Rokach, 2010).

On the other hand, the third strategy is a combination of classifiers each trained on a single class (Hao et al., 2009; Juszczak and Duin, 2004; Lee and Lee, 2005; Tax and Duin, 2008).

Quatrini et al. (2020) and Scime and Beuth (2018) investigated the use of ML for anomaly detection and classification in manufacturing processes. Quatrini et al. considered a Decision Tree and Random Forest algorithm as classifier algorithms to solve the problem. The Decision Tree is used to identify the process phase and the Random Forest algorithm implements the anomaly detection. Farid et al. (2014) presented a multi-class classification problem solved by a Hybrid Decision Tree and Naïve Bayes classifier. The Naïve Bayes classifier removed the noise from the training set before the Decision Tree was inducted. Support Vector Machines (SVMs) are also used in classification problems many times. SVMs are primarily designed for binary classification problems. However, a combination of multiple SVMs can be used to solve multi-class classification problems (Mayoraz and Alpaydm, 1999). Mayoraz et al. investigated the problem of scaling in such combinations of SVMs. Various normalization methods are proposed to cope with the scaling problem.

ANNs are also a very popular ML technique (Bhardwaj et al., 2016). It offers excellent opportunities for solving multi-class classification problems and is applied in many different fields, from image recognition to inventory management. Most related research is done in the field of image recognition. For instance, Singh et al. (2012) presented a Haar wavelet transform and backpropagation NNs approach for texture image recognition.

Ding and Dubchak (2001) researched a multi-class protein fold recognition using SVMs and NNs. The research shows the difference between the two algorithms and shows higher accuracy achieved by the SVMs. However, the accuracy of the NN is improved significantly by implementing noise reduction.

ANNs are also used to perform fault diagnosis because of several advantages compared to other ML algorithms (Grezmak et al., 2020). One of these advantages is the overall classification accuracy. In the research of Grezmak et al., fault diagnosis is executed on a motor, operating in multiple operating conditions. Bhardwaj et al. (2016) researched Genetically Optimized NN in multi-class classification problems and presented better results compared to other ML algorithms. Price et al. (1995) presented research where a multi-class classification problem is addressed with a divide and conquer strategy. This strategy divides the problem into multiple two-class problems, and for each pair of classes, a (small) NN with a single output unit is trained.

Ou and Murphey (2007) researched the differences between different approaches of NNs. From this research can be concluded that individual NNs per class pair are more straightforward than a single NN for all classes. The different individual NNs can be modelled individually, which enables fast learning. However, a single NN is trained on all information available which can result in an optimal classification. The research showed a good result when the training dataset is not too large, and there are not too many classes.

Another research is done by Miškuf and Zolotová (2016), comparing different multi-class classifiers with a focus on Industry 4.0. In this research, the Letter Recognition dataset from UCI is used. The research mentioned the comparison of this dataset with an industrial environment. Values from various sensors can replace all numerical columns, and the classes can contain measures or actions. The research compared six different multi-class classifiers:

- Multi-class NN
- Multi-class Decision Jungle
- Multi-class Logistic Regression
- Multi-class Decision Forest
- Ensemble of two-class SVM
- Deep Learning

The different models were trained on a dataset containing 16.000 samples. The results of the models on the test dataset (4.000 samples) are shown in Figure 2.15. The results show a better accuracy for the Deep Learning algorithm (96,48%) compared to the Decision Forest (92,50%), NN (92,43%), Decision Jungle (80,10%), Logistic Regression (73,93%) and SVM (66,20%).

All these different studies addressed a variety of algorithms. However, only a few algorithms show good potential for this research. The next section discusses the characteristics of these algorithms and in Section 3.2.3, the most suitable algorithm is chosen for the development of a ML model.

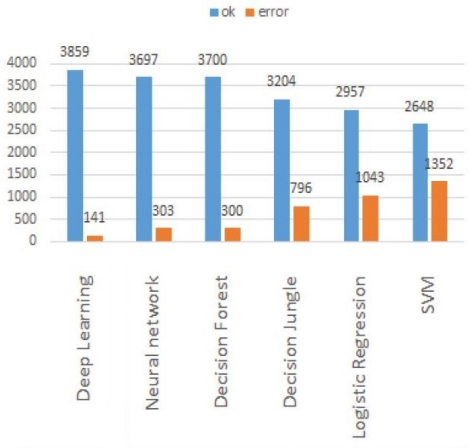


Figure 2.15: Multi-class classifiers and Deep Learning (Miškuf and Zolotová, 2016)

2.6.5. Challenges multi-class classification problems

A problem of multi-class classification problems is to gather the labels of the recorded data samples. Labelling is time-consuming, which requires a high workload for experts because they have to look into al all the data samples and label these according to their observations (Lughofer, 2012). Labelling means a high investment in time and money for the company where the expert/operators are employed. For example, the classification of image recognition in surface inspection requires an expert to look into all images and determine to which class they belong (e.g. showing no faulty occasions or showing faulty occasions).

2.7. Conclusion

This chapter discusses the requirements of a system to fulfil the user requirement. The user requirement is to reduce the unplanned operational downtime by supporting operators to resolve errors faster to increase machine utilization. The system requirements are split into requirements for the infrastructure and requirements for the service to support operators in resolving errors faster. The sub-question which is answered in this chapter is: *What are the requirements to enable real-time monitoring of a production line and what service is required to address the user requirement?*

First, the infrastructure requires the extraction of data from the physical environment via an IoT system. The historical data is stored in a database, and the real-time data is accessible in real-time. When real-time data is accessible, it is possible to develop different services and models based on this data. The five-dimensional model (see Figure 2.6) of Tao et al. (2019a) describes this as the five requirements of a DT. The data of the packaging line in Alken-Maes is already accessible in real-time, which enables the development of a service model to address the user requirement. However, the amount of sensors available is low and has to be extended to offer a complete overview of the packaging line.

The user requirement is addressed in this five-dimensional model as a service. This service requires a decision algorithm that can predict output classes based on several input parameters. Such a decision algorithm is a multi-class classification algorithm. These types of algorithms are studied extensively in the past in multiple different areas. However, there is a gap in the literature about multi-class classification problems in real-time production environments. The research of Miškuf and Zolotová (2016) addresses this topic in his research by relating the used dataset to industrial sensors and desired outputs. However, his dataset is still a Letter Recognition dataset instead of a dataset consisting of industrial instances.

The research gap will be addressed in this research in the following chapters. First, the requirements for designing a multi-class decision algorithm are discussed. Then, a multi-class decision algorithm is built based on data from the physical environment. Finally, the integration of the decision algorithm in the physical environment is tested and discussed.

3

Design of service model

In Chapter 2, the system requirements are determined. The next step of the Systems Engineering methodology is to design the different components; in this case, the service of the DT. The service of the DT is an algorithm to solve a multi-class classification problem. The algorithm will predict certain output measures performed by an operator when an error occurs. The sub-question which will be answered in this chapter is: *What are the design requirements of intelligent decision-making algorithms?* First, in Section 3.1, an overview of the entire process is given. In Section 3.2, a design framework is discussed, which is the main part of the remainder of this research. Finally, the output of the brewery is discussed.

3.1. Overview

An overview of the entire process is shown in Figure 3.1. First, the physical equipment in the brewery needs to be equipped with sensors. Secondly, this data is captured by a central system that connects the physical object to a server. Next, the data is uploaded to the cloud from where it is possible to access the data. From there, the data needs to be processed in a model to make the data useful to make decisions based on the data. Finally, the output of the model (decisions) is communicated back towards the brewery, in this case to the operator.

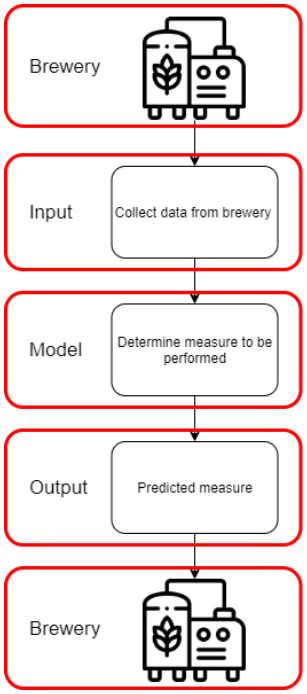


Figure 3.1: Overview model

The previous section already described the requirements for the data extraction and communication between the production environment and the digital environment. When the data is accessible in a digital environment, and a model is developed to analyze the data, it is possible to make decisions based on real-time data.

The decisions consist of countermeasures dependent on the input variables from the packaging line. This service enables operators to execute the right countermeasure to resolve errors faster and to reduce the unplanned downtime. Therefore, a model is developed, which can predict countermeasures based on the variables of the packaging line.

3.2. Model design framework

The development of ML algorithms has been studied extensively over the last decades. These studies all use a seven-step framework for designing ML algorithms (Brownlee, 2013; Guo, 2017; Mayo, 2018; van Rijmenam, 2019). The seven steps are:

1. Data collection
2. Data preparation
3. Choose a model
4. Train the model
5. Evaluate the model
6. Hyperparameter Tuning
7. Prediction

The remainder of this chapter describes the content of this framework and discusses the relevant design requirements and parameters.

3.2.1. Data collection

Data collection consists of defining and obtaining data. This first step is immediately crucial because the quality and quantity of the data determine how good the prediction of the model is (Guo, 2017). This step is dependent on the type of problem and the desired output of the model. The outcome of this step is generally a representation of data (in the form of a table) which will be used for training the model (step four). From the packaging line of Alken-Maes, different variables are available to use as input. These variables are described below, as well as the output labels.

Active program

The active program is dependent on the brewed beer and type of packaging. Every combination has a unique program that includes different speeds, dimensions, and secondary packaging. The program does not often change, on average once every two days, and is set by an operator. Despite the inactive character of this variable, it is included in the dataset to enable variations in future tests. As the program does not change over the day, this variable is set to one for every data sample.

Errors

The three most occurring errors are included in the dataset. These three errors are:

- Error 1: Can lack at inlet 1
- Error 2: Low speed inlet conveyor
- Error 3: Outlet can too full

Speed of machine

The nominal speed of the machine is dependent on the program. However, the speed variates regularly because of varying variables and states of components in the entire packaging line. The nominal speed is set between 740-760 [cans/min].

Furthermore, the speed of the Filler's inlet conveyor is considered as it does influence the machine, and it indicates any problems in the stage before the Filler. However, the inlet speed is not accessible and is generated based on the assumption that the nominal speed of the inlet conveyor is similar to the speed of the Filler and can never be higher than the speed of the Filler.

State

The state of the Filler is an integer value as there are multiple operating states. The Filler's state is not considered in the training dataset because the state is already reflected in the speed of the Filler.

However, the state of the machine before the Filler is considered as it might influence the performance of the Filler. The machine before the Filler is the Depalletiser, which takes the empty cans of the pallet and feeds them into the packaging line. For simplicity, the state of the Depalletiser can either be zero or one, which represents a non-operating state and operating state, respectively.

Output classes
The model output is a countermeasure that has to be performed by the operator to solve an error. These measures are labelled classes and are used to train the model. The countermeasures considered in the model are:

- Countermeasure 1: Unknown cause: investigate cause
- Countermeasure 2: Stop Filler
- Countermeasure 3: Solve a problem with Depalletiser
- Countermeasure 4: Solve can block at Filler
- Countermeasure 5: Slow down Filler
- Countermeasure 6: Solve can block at outlet Filler

During operation, it sometimes appears that the cause of the error is unknown. The first countermeasure addresses this problem and, therefore, requires additional investigation to the cause of the error by the operator to solve the problem. The second countermeasure is a countermeasure where the Filler automatically stops due to the cause of the problem. The third countermeasure solves a problem at the Depalletiser indicated by one of the input variables. The fourth countermeasure solves a can block at the entrance of the Filler. The fifth countermeasure is similar to the second countermeasure, but now the Filler slows down automatically. Finally, the sixth countermeasure is again a can block, now at the outlet of the Filler.

An overview of all the input variables and the output classes is provided in Table 3.1. The values of the different variables can be found in the second column of the table.

Variabele	Value
Active program	1
Error	1, 2, 3
Speed of Filler	0-760
Inlet speed of Filler	0-760
State Depalletiser	0, 1
Output classes	1, 2, 3, 4, 5, 6

Table 3.1: Overview variables dataset

3.2.2. Data preparation
The second step is data preparation. The data is loaded into a suitable format and prepared for use in the ML algorithm (Guo, 2017). Data preparation is important because it will affect the result positively or negatively depending on the taken steps.

The important steps in data preparation are (Brownlee, 2013; Guo, 2017; Mayo, 2018):

- **Formatting:** The selected data might not be available in the right format and has to be transformed into a machine-readable format.
- **Cleaning:** Especially data from production environments is not always complete. Therefore, it is possible that data instances are incomplete or do not carry the correct data. These instances might be removed from the dataset as well as variables that are not needed in the dataset.
- **Sampling:** The available data might exceed the necessary data in terms of the amount for training and testing the algorithm. Too much data can result in large computational and memory requirements and will lead to long-running times of the algorithm. Therefore, it might be better to sample the data to use a smaller dataset and achieve faster running times. Before taking a sample, randomization of the dataset can be necessary to make sure the output is not dependent on the order of the dataset.
- **Scaling:** The available data contains different variables with various scales or quantities. Many ML algorithms require variables with the same scale, such as between 0 and 1. Therefore, the data can be normalized, standardized, or scaled by any other data scaling technique.

- **Decomposition:** It might be possible that input variables represent a complex concept and are more useful when split into constituent parts. An example is the date, which can be split into the day, month, year, and even in time components.
- **Aggregation:** Some variables might be more useful when aggregated into a single feature.

Furthermore, it might be useful to visualize the data to see if there are relevant relationships in the data and if there are any outliers or imbalances in the model (Guo, 2017).

When the data is prepared, the dataset has to be split into a training and test dataset (Guo, 2017). The majority of the dataset will be used for training purposes, while the test dataset will be used for evaluating the performance of the model.

3.2.3. Choose a model

The third step consists of selecting the right model (van Rijmenam, 2019). Many different ML models can be used for many different purposes. As discussed in Section 3.2.1, the available data, and desired output define the type of ML model. The problem in this research is a multi-class classification problem with a supervised learning process. Based on these characteristics, different algorithms are suitable to solve this problem.

The algorithms suitable for solving this problem are already discussed in Section 2.6.3 and can be divided into two groups (Aly, 2005). The first category algorithms are naturally suitable for multi-class classification problems and include Decision Trees (Gordon et al., 1984; Salzberg, 1993), NNs including Deep Learning algorithms (Bishop, 1995) and kNN (Bay, 1998). The second category includes approaches for breaking down the problem in multiple binary problems which are solvable by algorithms suitable for solving binary classifiers such as Support Vector Machines (Burges, 1998; Cortes and Vapnik, 1995).

The goal of this research is not to compare the performance of different algorithms. Therefore, a NN is chosen as multi-class classifier because of the natural suitability for this kind of problems. Moreover, a NN is useful for high dimensional data, and as the problem in real-life can expand to a high dimensionality problem, this algorithm is very suitable for this research.

A NN consists of an input layer, one or more hidden layers of neurons, and an output layer. The input variables are already defined in Section 3.2.1 and consist of five different variables. To each input variable, a neuron is assigned in the input layer. Similarly, there are six output classes, and each output class is assigned to a neuron in the output layer. The number of hidden layers and neurons in each hidden layer will be tuned in Chapter 4. As a starting point, a NN with one hidden layer consisting of four neurons is chosen, see Figure 3.2.

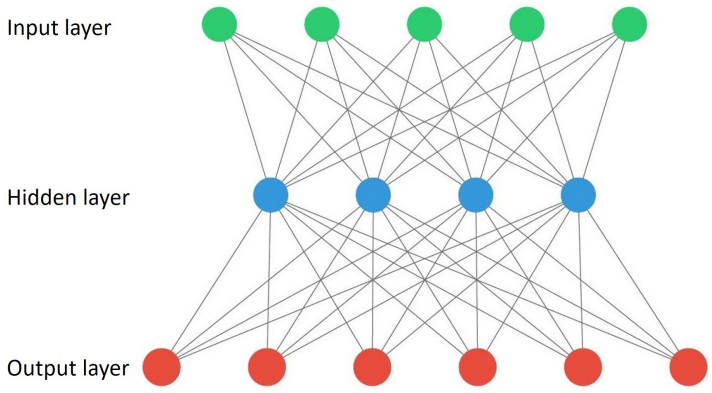


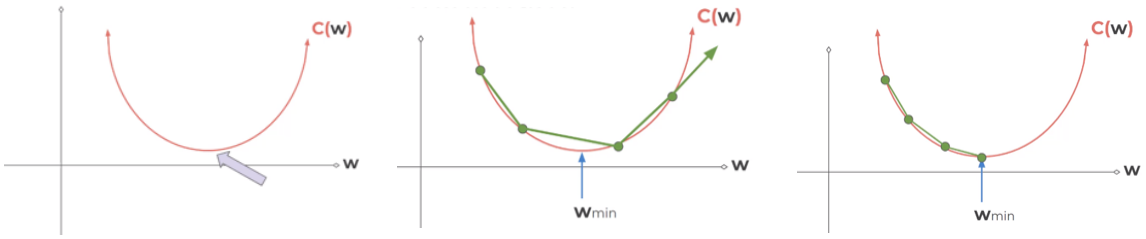
Figure 3.2: Initial NN

Neural Network Optimization
The optimization of NNs is a non-convex optimization (Stewart, 2019). Non-convex optimization contains multiple different local optima and one global optimum. A NN is optimized on its loss function. It can be challenging to find the global optimum because it is dependent on the loss surface. In earlier research, finding the global optimum was considered to be a significant problem in NN training. However, recent studies have proven that most local optima already reaches low minima close to the global minimum, which is sufficient in terms of optimization (Choromanska et al., 2015). Hyperparameter tuning is vital to reach the global minimum or an acceptable good local minimum.

Keras
Due to the increased interest in ML and Deep Learning, several different software packages can be used to build a user-friendly NN. Keras is such a user-friendly package (Nain, 2017). Keras is entirely modular, so users can easily combine different modules to build extensive or straightforward NNs. Therefore, Keras is used to build the NN of this research.

3.2.4. Training the model
Training the developed model is the bulk of ML (Guo, 2017; van Rijmenam, 2019). The goal of training the model is to incrementally improve the prediction of the model based on the training data. At the very start of training, the prediction based on the input variables is wrong according to the corresponding output. By iteratively comparing the predicted output with the corresponding output, the model can adjust its prediction. After training the model extensively, it can predict the right output corresponding to the input variables.

The training of a NN considers the training dataset and the loss function. The goal is to increase the loss function iteratively by comparing the estimated outputs to the real values of the label. The loss function is a combination of all weights of the model. The weights are denoted by *W*, and *B* denotes the biases. The minimum of the loss function $C(W)$ (see Figure 3.3a) is found by taking the derivative of the loss function. The learning rate determines how fast the model reaches the minimum, see Figure 3.3c. However, if the learning rate is too high, the loss function may overshoot the minimum, see Figure 3.3b.



(a) Loss function example (b) Overshoot of loss function (c) Minimum of loss function

Figure 3.3: Example of training a NN (Heineken, 2020b)

3.2.5. Evaluate the model
After training, the next step is to evaluate the model. The test dataset, which we set aside at the data preparation step, is now considered and fed into the model (Guo, 2017). The test dataset is new data which has not been seen by the model and is, therefore, used to determine the skill of the model on new data (Brownlee, 2017c).

Classification accuracy
Classification accuracy is a common used evaluation method. It can be described by the following equation (Mishra, 2018):

$$Accuracy = \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ predictions} \tag{3.1}$$

Accuracy works well as an evaluation method when the number of samples is equally divided over the classes. If the samples are not equally divided over the classes, for example, 98% belongs to class A, and 2% belongs to class B, our model can easily predict 98% accurate by predicting all samples to class A.

The training dataset in this problem is well divided over the different classes. Therefore, classification accuracy can be used to evaluate the performance of the model.

Logarithmic loss
The logarithmic loss penalizes the false classifications of a model and is very suitable for evaluating multi-class classification problems (Mishra, 2018). The classifier assigns probabilities to each class for all samples and penalizes the false classifications. A perfect classifier would have a logarithmic loss of zero. Logarithmic loss heavily penalizes classifiers that assign high probabilities to incorrect classifications.

Confusion matrix
Performance of classification models can be evaluated by precision and recall. It can be best described with the so-called ”confusion matrix” (Shmueli, 2019). Depending on the number of classes *n*, it has *n* rows and columns. The matrix shows how many samples of one class were predicted correct or wrong. Figure 3.4 shows an example of such a confusion matrix with three classes.

		True/Actual		
		Cat	Fish	Hen
Predicted	Cat	4	6	3
	Fish	1	2	0
	Hen	1	2	6

Figure 3.4: Example of confusion matrix (Shmueli, 2019)

It shows the number of times a cat picture is predicted as a cat and how many times it is predicted as a fish or hen. If a picture contains a cat and a cat is predicted, it is called a True Positive (TP), and if a fish or hen is predicted it is called a False Negative (FN). The opposite yields if a picture does not contain a cat, but a cat is predicted. That is called a False Positive (FP), but if a fish or a hen is predicted, it is called a True Negative (TN).

Precision and Recall
From the confusion matrix, the precision and recall can be calculated. Precision answers the question: *”What proportion of predicted positives is truly positive?”* (Shmueli, 2019). Precision can thus be calculated according to the following equation:

$$Precision = \frac{\sum_{i=1}^{l} TP_i}{\sum_{i=1}^{l} (TP_i + FP_i)} \tag{3.2}$$

In case of the example, the precision of predicting the cat is:

$$Precision = \frac{4}{4+6+3} = 30,8\% \tag{3.3}$$

Recall answers the question: *”What proportion of actual positives is correctly classified?”* (Shmueli, 2019). Recall can thus be calculated according to the following equation:

$$Recall = \frac{\sum_{i=1}^{l} TP_i}{\sum_{i=1}^{l} (TP_i + FN_i)} \tag{3.4}$$

In case of the example, the recall of predicting the cat is:

$$Recall = \frac{4}{4+1+1} = 66,67\% \tag{3.5}$$

Similarly, the precision and recall for the other classes can be calculated.

The service of the model is to support operators in resolving errors faster by proposing the correct countermeasure when an error occurs. It is essential to predict the correct countermeasure because this will reduce downtime. In the current situation, it happens that operators execute the wrong countermeasure, or it takes a long time to decide which countermeasure is the right one. Therefore, the goal is to reduce the overall loss to an absolute minimum because this indicates a high accuracy.

In this application of the model, it is more important to achieve a high recall than a high precision. A high recall indicates that the actual class is recognized correctly, which is vital in predicting the right countermeasure.

3.2.6. Hyperparameter Tuning

The sixth step is more an ”art form” rather than science (Mayo, 2018). This step refers to hyperparameter tuning intending to improve the training of the model further (Guo, 2017). Hyperparameters are different compared to variables (Brownlee, 2017d). Variables are configuration variables that are inserted in a model. The model learns from these variables to predict certain outputs. Hyperparameters are configurations that are external to the model and whose value cannot be estimated from data. The tuning of the hyperparameters is done experientially and heavily depends on the specifics of the dataset, model and training process. The hyperparameters of a NN that can be tuned are:

- Number of training epochs
- Batch size
- Optimization algorithm
- Learning rate of the optimization algorithm
- Activation function
- Dropout regularization
- Number of hidden layers
- Number of neurons in each layer

The architecture of a NN is determined by two primary hyperparameters: the number of hidden layers and the number of nodes in each hidden layer. The most reliable way to configure these hyperparameters is via systematic experimentation. Much research is done to determine the number of hidden layers and the number of nodes scientifically. Lippmann (1987) shows that a two-layer NN is sufficient for solving any non-linear problem. In contradiction, another theoretical finding stated that a NN with one hidden layer can approximate any function required (Goodfellow et al., 2016). Furthermore, Reed and Marks (1999) shows that a large one-hidden-layer NN can be less efficient in solving specific problems that NNs with two (or more) hidden layers.

These different studies contradict each other, but it shows that the configuration of the number of layers in a NN is highly dependent on the specific application. Therefore, experimentation is the key to find optimal hyperparameters.

3.2.7. Prediction

Finally, when the model satisfies the needs, it is possible to predict based on the dataset and the model. It is possible to use further data which have never been used in the training of the model. It is also possible to predict on data that has no output values or classes.

3.3. Output layer

The output layer consists of the number of output classes addressed in the problem. The predicted output exists of the probabilities of classes, which is achieved by using the Softmax activation function in the output layer (Radecic, 2018). The Softmax activation function reports the confidence score for each class, and the class with the highest confidence score is the predicted class. The mathematical representation of the Softmax function is:

$$Softmax(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{3.6}$$

The equation states that to each output an exponential function is applied and these values are normalized by dividing by the sum of all the exponentials. This ensures the sum of all exponential values adds up to one.

When the output class is predicted, this is communicated back to the brewery. The operator is notified about this countermeasure. Finally, the operator has to execute the countermeasure.

3.4. Conclusion

This chapter discusses the design requirements of the decision algorithm, which delivers the service required to support operators in resolving errors faster. The sub-question answered in this chapter is: *What are the design requirements of intelligent decision-making algorithms?*. The requirements are discussed by the seven-step framework consisting of data collection, data preparation, choose a model, train the model, evaluate the model, tune the hyperparameters, and predict.

First, the collected data consists of input variables with corresponding (output) labels, which is required to train the model. A NN is chosen as the decision algorithm in this research. A NN is effective for high dimensional data, so the problem in this research can be easily extended. The NN requires training according to the training dataset to achieve excellent performance. The performance of the NN is evaluated with several evaluation methods. The most important evaluation method for this research is the logarithmic loss. A small logarithmic loss indicates a high prediction accuracy. Furthermore, high recall is required because this indicates that the actual output class is recognized correctly, which is essential to predict the right countermeasure. It is required to improve the performance of the NN by tuning the hyperparameters experimentally. Once the model is trained, it is possible to predict the right countermeasures according to the input variables.

The output layer of the NN consists of output classes which can be predicted by the NN. The output classes are different countermeasures. These countermeasures are proposed to operators working in the physical environment. Every time an error or fault occurs, a countermeasure is predicted by the NN, and the operator performs this countermeasure to resolve the error. The next chapter describes the development of the model, including training the model and hyperparameter tuning according to the required design.

4

Developed service Model

The previous chapter describes the requirements of designing a decision algorithm in a DT environment. These requirements are used in this chapter to build a NN which can predict a certain output class based on the specified inputs. This is the fourth step of the Systems Engineering methodology. In Section 4.1, the setup of the model is discussed. In Section 4.2, the data is defined and prepared before a basic NN is built in the following section. Then, the hyperparameters of the NN are tuned to achieve good prediction accuracy. The final model is described in Section 4.5. Then, the model is verified in Section 4.6 to verify the predictions compared to the desired outputs.

4.1. Setup of the model

To be able to design an appropriate model, the capability of the model needs to be determined. As discussed in Chapter 3, the goal is to design a model that can predict a certain countermeasure based on a combination of different variables. Because it is important to predict the right outputs, the accuracy of the model is important. Furthermore, the computation time for predicting an output is important because the model is designed for real-time operations. Therefore, the time to predict an output must be short.

As already discussed in Section 3.2.1, the problem considered in this research is a multi-class classification problem. The output is in a set of classes. These classes represent the countermeasures to be taken by the operator in the brewery. The model always predicts one output class, together with the certainty of the decision.

4.2. Define and prepare data

The considered variables of the packaging line are discussed in Section 3.2.1. There are five input variables considered: the program, the state of the Depalletiser, the speed of the Filler, the inlet speed of the Filler, and the three most occurring errors. The output consists of six possible countermeasures. In this Section, the preparation and visualization of data are discussed.

4.2.1. Overview training data

The dataset for training and testing is composed based on the specified input variables and output classes. Each sample of the dataset consists of multiple variables that occur during a shift. Each data sample is complete and does not have any missing data. In real-life, this is not the case, and the data has to be cleaned before it can be used in the model.

In Table 4.1, an overview of the generated data is given. This data is a realistic representation of data from the packaging line of Alken-Maes. However, this data is simulated and used to train the NN. The value or outer bounds of the five considered input variables, the corresponding output class, and the number of samples are shown in the table.

Variables	1	2	3	4	5	6	7	8	9	10	11	12
Program	1	1	1	1	1	1	1	1	1	1	1	1
State Depalletiser	1	1	2	2	1	2	1	2	1	2	1	1
Speed Filler [cans/min]	740-760	740-760	740-760	740-760	740-760	0-750	0-750	740-760	0-750	0-750	0-750	740-760
Inlet speed [cans/min]	740-760	600-750	0-750	740-760	740-760	0-750	0-600	0-600	0-600	0-600	0-750	740-760
Error	1	1	1	1	1	1	2	2	2	2	3	3
Measure/ Class	1	2	3	3	4	3	2	2	4	3	5	6
Number of samples	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000

Table 4.1: Overview of trainings dataset

4.2.2. Data preparation

According to step two of the design framework described in Section 3.2, the data is prepared before the data is used for training the model. The dataset is generated in *.csv* format and every sample contain all the information, so formatting and cleaning are not necessary. The order of samples of the generated dataset is shuffled randomly. From Table 4.1 can be seen that the different input variables have different scales. These scales need to be the same to penalize all different variables with the same magnitude. Therefore, the data is normalized to realize the same scale between zero and one for every variable.

Algorithm 1: Preparation of dataset

```
Result: Normalize dataset
for i in length(dataset) do
    normalize dataset ;
end
Result: Split dataset
Input training dataset = 90% input samples;
Output training dataset = 90% output samples;
Input test dataset = 10% input samples;
Output test dataset = 10% output samples;
```

4.2.3. Visualization of data

In Section 3.2.2 it is suggested to visualize the dataset to check for relationships in the dataset and find any outliers. The data presented in Section 4.2.1 has five dimensions, excluding the output class. A dimensionality reduction technique is used to reduce the dimension from five to two dimensions, which enables visualizing the dataset in a two-dimensional graph. The dimensionality reduction algorithm used is the t-Distributed Stochastic Neighbor Embedding algorithm (t-SNE). This technique produces better visualizations by reducing the inclination to crowded points together in the center of the graph. T-SNE is a variation on the Stochastic Neighbor Embedding technique (Van Der Maaten and Hinton, 2008). The result is presented in Figure 4.1. The x- and y-axis both have no units as it is a combination of five variables. The colours indicate the different output classes defined in Section 3.2.1.

The most significant advantage of visualizing the dataset is to find any overlap between samples in the dataset. If an overlap between samples occurs, the model will not be able to predict the correct output class for that sample based on the input. The reason is that the input variables are the same for different samples despite different output classes. In Figure 4.1, all the data samples which are labelled as output class one can be located in red in the lower right corner. All data samples, which are labelled as output class two, can be located in different yellow clusters divided over the graph. The other output classes can be located similarly, according to the colours in the diagram legend.

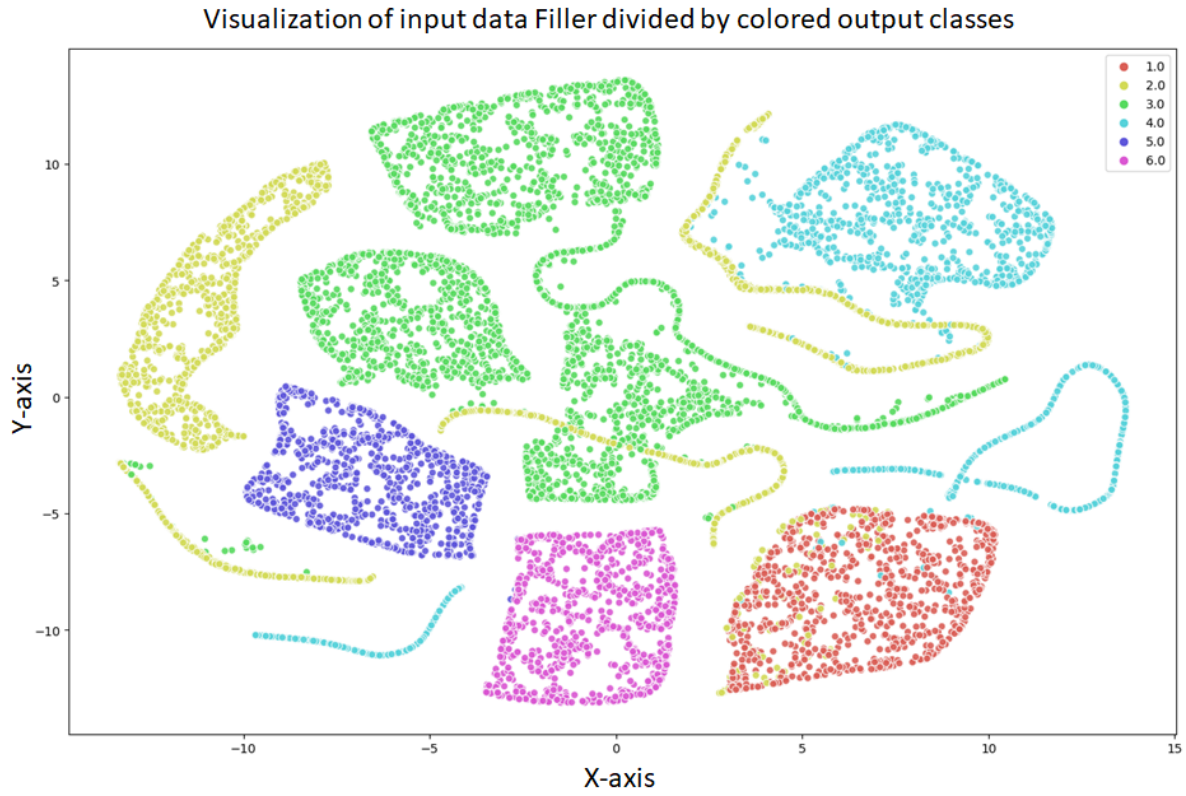


Figure 4.1: Visualization of input data after dimensionality reduction

From this figure, the relationship between the inputs and outputs is visible. This relationship is non-linear because non-linear lines can separate the data points. Furthermore, it can be seen that some output classes show overlap with other output classes. Especially output classes one and two, two and three, and two and four show some overlap in the data points.

4.3. Model built-up

A model is built as a basis from where the hyperparameters are tuned. The number of nodes of the input and output layer are already defined and are five and six, respectively. For the basic model is chosen for regularly used hyperparameters to be able to experiment on these hyperparameters.

Activation function

The activation function (or transfer function) determines the output of any node. It maps the output into a value between 0 and 1 or -1 and 1. The function used in this basis NN is the ReLU activation function. The ReLU function is half rectified which will cause the output to be 0 when *z* is less than 0 and *f(z)* is equal to *z* when *z* is above or equal to 0, see Figure 4.2. The ReLU function is a frequently used activation function in almost all NNs or Deep Learning (Sharma, 2017).

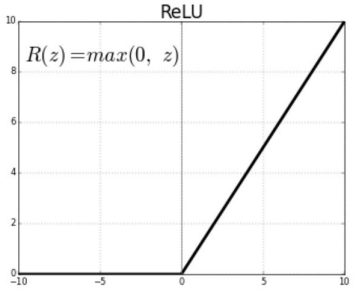


Figure 4.2: ReLU activation function (Sharma, 2017)

Other activation functions are also considered during experimentation such as the Sigmoid activation function, the Tanh activation function, the Leaky ReLU activation function, etc., see Section 4.4.

Furthermore, the output layer contains a Softmax activation function. The Softmax activation function turns the numeric output from the last hidden layer into probabilities by taking the exponents of each output and normalize each number by the sum of those exponents. The output vector contains all

probabilities, and the sum of the output vector is one.

Optimizer
The right optimization function achieves good results in minutes while a wrong optimization leads to good results in hours. The Adam optimization algorithm is used in the basic model, which is a combination of two stochastic gradient descent procedures: Adaptive Gradient Algorithm (AdaGrad) and Root Mean Square Propagation (RMSProp) (Brownlee, 2017a). The hyperparameter of the optimization function, which can be tuned, is the learning rate. The default learning rate for each optimization function is 0,001. Also, during experimentation, other optimization algorithms are used, such as Stochastic Gradient Descent (SGD), Nadam, AdaGrad, RMSProp, etc.

Loss function
The most commonly used loss function for multi-class classification problems is the cross-entropy function. Cross-entropy calculates a score that is the difference between the predicted and desired output for all classes. In Keras, the cross-entropy loss function can be specified as *'categorical_crossentropy'*. It requires *n* nodes (one for each class) and a Softmax activation function in the output layer.

***K*-fold cross validation**
A NN is stochastic by nature, which means that randomness is used to determine the initial weights, and when the dataset is shuffled during each training epoch. The model is fit on the training dataset and evaluated on the test dataset. However, due to the randomness, the skill of the model can vary every time the model is fit on different data. Applying *k*-fold cross validation gives more accurate results because it splits the data into *k*-folds (Brownlee, 2017b). Then, it fits the model on *k-1* folds, evaluates this on the last fold, and repeats this for each fold. This results in *k* different models and in turn, *k* different skill scores. In the end, the average of all folds is taken, which provides a more realistic performance of the model. The number of folds for every experiment is set to ten in this research.

Initial model
The initial model is shown in Figure 4.3. The model consists of one hidden layer with four nodes. This model is used as a base reference during experimentation. The goal is to improve the performance of the initial model by tuning the hyperparameters in the next section.

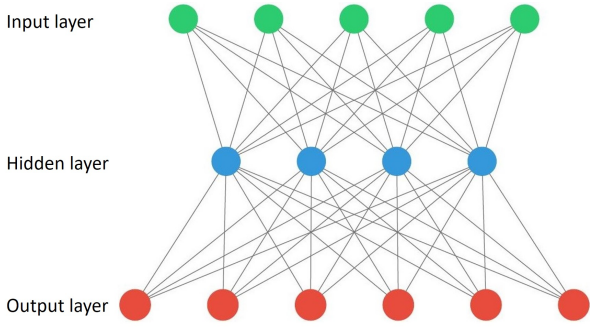


Figure 4.3: Initial NN

Algorithm 2: Initialization of model

Result: Setup of model
Number of hidden layers: 1;
Number of neurons: 4;
Activation function: ReLU;
Optimizer: Adam;
Learning rate: 0,001;
Loss function: Cross entropy;
Number of k-folds: 10;

All the default hyperparameters discussed in this section are used, which results in a logarithmic loss of 0,38 and an average accuracy of 85,25%. Figure 4.4 shows the logarithmic loss and accuracy curves of the basic model. These curves are checked after each experiment to check whether the model is not over-fitted or under-fitted. If the model has too much computational capacity, it learns the training data very well but is not able to generalize the knowledge to the test data. This is called overfitting and can be spotted as a continuous decrease in the training loss, while the test loss decreases and starts increasing again. Another symptom of overfitting is when the validation curves are noisy. Underfitting refers to a model that cannot learn from the training dataset. Under fit models may show a flat line or noisy values of relatively high loss or a continuous decrease of the training loss until the end of the training.

Figure 4.4 shows an underfit model because the training loss is still decreasing at the end of the training. The training and test loss also shows a small gap between the curves, which indicates a relatively easy test dataset compared to the training dataset. This causes the model to predict better on the test dataset.

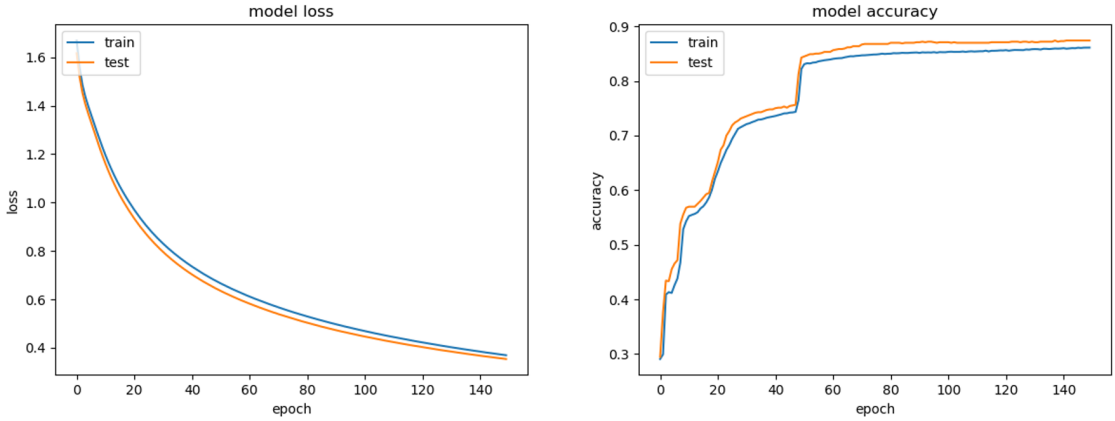


Figure 4.4: Logarithmic loss and classification accuracy for one fold of base model

4.4. Hyperparameter tuning

The ultimate goal of hyperparameter tuning is to reduce the logarithmic loss to zero, which indicates a perfect trained model and consequently, perfect prediction. The tuning of the hyperparameters is done by structured experimentation. For every hyperparameter, multiple values or options are considered to check whether the performance improves.

As mentioned before, the strength of a NN is the stochastic behaviour of the model. Despite this strength, is it necessary for hyperparameter tuning to fix the randomness to compare the performance of different hyperparameters.

Number of training epochs and batch size

The number of training epochs, and the batch size are dependent on each other and together influence the performance of the model. The number of training epochs is the number of times the entire training dataset is shown to the NN during training. The batch size is the number of patterns shown to the network before weights are updated. From Figure 4.4, it can be concluded that the model is underfitting because the loss function is still decreasing at the final training epoch. The number of training epochs in this initial model is set to 150, and the batch size is 512.

Algorithm 3: Find the optimal training epoch and batch size

```
Result: Optimal training epoch and batch size
Training epochs: [150, 250, 350];
Batch size: [512, 256, 128, 64];
for epochs in Training epochs do
    for batch in Batch size do
        Train model(epochs,batch);
    end
end
```

Table 4.2 shows the results of iterating over the number of training epochs and the batch size. The table shows the logarithmic loss, the average accuracy over 10 folds, the standard deviation (std) of the accuracy over the 10 fold, and the time it took to train the model 10 times (10 folds). Multiple combinations of epochs and batch sizes show similar results. However, the standard deviation and time of these combinations vary a lot. The best result is achieved with 350 training epochs and a batch size of 64. Figure 4.5 shows the logarithmic loss and accuracy curves of one fold of this best result. This figure shows a noisy validation accuracy which indicates an overfitted curve. Overfitting often occurs if the model has more capacity than is required for the problem. This also yields for the achieved result with 350 epochs and a batch size of 128. Therefore, the best usable result is achieved with 350 epochs and a batch size of 256. Figure 4.6 shows the logarithmic loss and accuracy curves of one fold for 350 epochs and a batch size of 256.

Nr. of epochs	Batch size	Loss	Accuracy [%]	Std accuracy	Time [s]
150	512	0,3170	88,46	5,86	60
250	512	0,1930	94,11	3,43	79
350	512	0,1644	95,82	2,12	105
150	256	0,2348	91,43	5,87	64
250	256	0,1518	95,32	2,81	89
350	256	0,1251	96,80	1,63	225
150	128	0,1835	93,97	4,50	148
250	128	0,1252	96,13	2,59	245
350	128	0,1014	97,31	1,42	342
150	64	0,1523	95,34	4,34	306
250	64	0,1058	96,83	2,25	515
350	64	0,0951	97,76	0,63	749

Table 4.2: Result of iterating number of training epochs and batch size

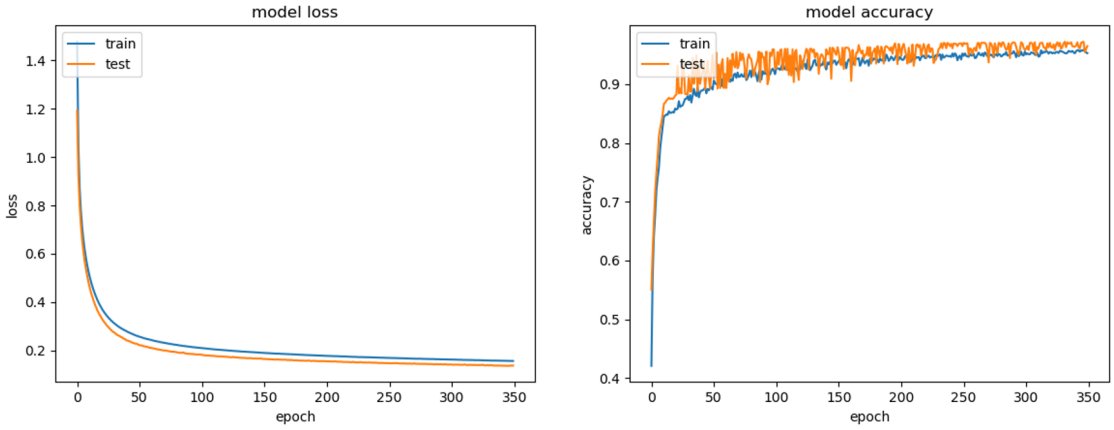


Figure 4.5: Logarithmic loss and classification accuracy for one fold with 350 training epochs and batch size of 64

During the remaining hyperparameter tuning, the number of training epochs and batch size is not tuned anymore, but the result of this section is used. However, during the tuning of other hyperparameters, the logarithmic loss and accuracy curves are always checked to make sure the model is converging to a stable loss and is not over fitted or under fitted.

Optimization algorithm

There are several optimization functions available to optimize the model. An overview of these algorithms is shown in Table 4.3. Besides tuning the hyperparameters of an optimization algorithm, also the type of algorithm can vary. Therefore, different optimization algorithms are compared. The results can be found in Table 4.3. From the table, it can be concluded that Adadelta and Nadam achieved the

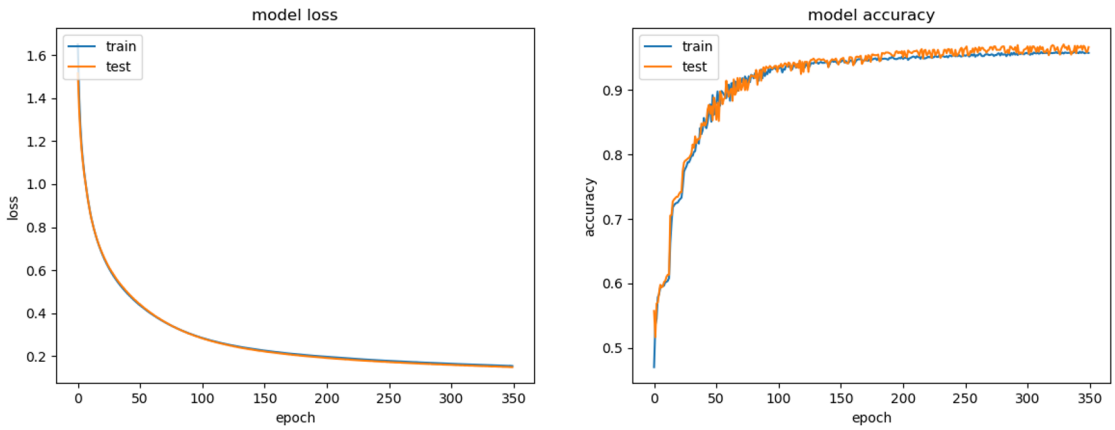


Figure 4.6: Logarithmic loss and classification accuracy for one fold with 350 training epochs and batch size of 256

best results. However, both show over fitted logarithmic loss and accuracy curves. Therefore, Adam is used as the optimization algorithm during the remaining hyperparameter tuning. Finally, when the configuration of the model is completed, Nadam and Adadelta are both considered once more.

Algorithm 4: Find the best performing optimizer

```
Result: Best performing optimizer
Optimizers: [Adam, SGD, RMSprop, Adagrad, Adadelta, Adamax, Nadam];
for optimizer in Optimizers do
    Train model(optimizer);
end
```

Optimizer	Loss	Accuracy [%]	Std accuracy	Time [s]
Adam	0,1251	96,80	1,63	225
SGD	0,3589	88,65	9,02	255
RMSprop	0,1449	96,47	1,76	293
Adagrad	0,4230	87,97	5,29	289
Adadelta	0,1215	97,08	0,81	307
Adamax	0,1526	95,74	2,20	332
Nadam	0,0910	97,52	1,18	368

Table 4.3: Results of applying different optimization functions

Learning rate of optimization algorithm

The learning rate controls how much the weights are updated at the end of each batch. Different small variations are taken around the default learning rate to experiment on the learning rate. Furthermore, it is necessary to take the batch size and number of training epochs into account because there is a dependency between the learning rate, batch size, and the number of training epochs. Table 4.4 shows the result of the used learning rate. The two experimentations with a learning rate of 0,005 and 0,01 show the highest accuracy. However, the loss and accuracy curves show both an overfitted model for a learning rate of 0,005, see Figure 4.7. Therefore, the optimal learning rate in this model is determined to be 0,001, but a minor optimization will be done when the number of layers and neurons is specified.

Algorithm 5: Find optimal learning rate

Result: Optimal learning rate
Learning rate: [0.0001, 0.0005, 0.001, 0.005, 0.1, 0.15];
for *rate in Learning rate* **do**
  Train model(rate);
end

Learning rate	Loss	Accuracy [%]	Std accuracy	Time [s]
0,0001	0,4591	84,30	4,93	418
0,0005	0,1781	94,54	3,59	351
0,001	0,1251	96,80	1,63	225
0,005	0,0900	97,35	1,68	285
0,01	0,0814	97,33	2,10	395
0,015	0,1288	94,97	8,59	405

Table 4.4: Results of different learning rates

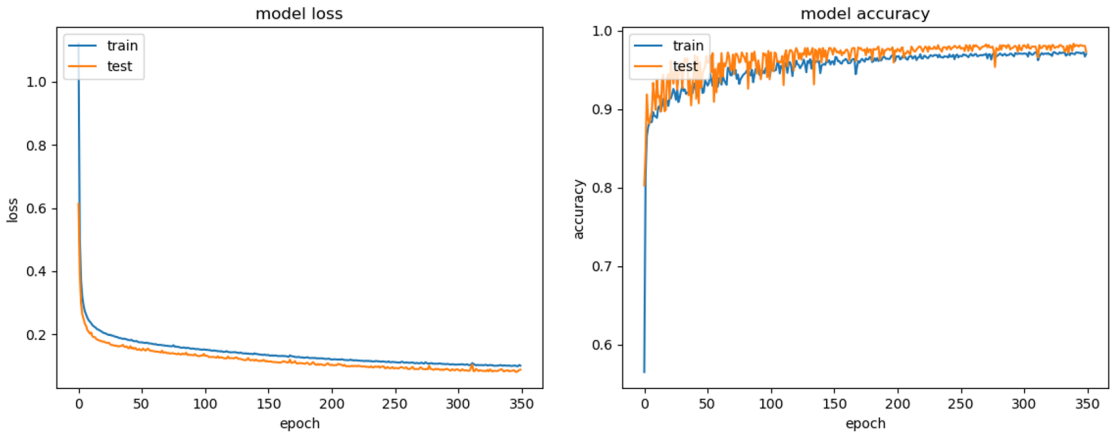


Figure 4.7: Logarithmic loss and classification accuracy for one fold with learning rate 0,005

Activation function

In the same way, the activation function in the hidden layer is evaluated. The activation function in the output layer has to be the Softmax function for a multi-class classification model and is not changed in this evaluation. Table 4.5 shows the result of the different activation functions. From this table can be concluded that the best result is achieved with the Tanh activation function. The Tanh activation function is shown in Figure 4.8. The Tanh activation functions map the output between -1 and 1 depending on the input. Both logarithmic loss and accuracy curves are shown in Figure 4.9. Both curves show a good fit and converging result.

Algorithm 6: Find the best performing activation function

Result: Best performing activation function
Activation function: [ReLU, Tanh, Sigmoid, Linear, Softmax];
for *function in Activation function* **do**
  Train model(function);
end

Activation function	Loss	Accuracy [%]	Std accuracy	Time [s]
ReLU	0,1251	96,8	1,63	225
Tanh	0,0929	97,6	0,18	123
Sigmoid	0,1758	95,3	3,07	129
Linear	0,1054	97,46	0,14	140
Softmax	0,2163	92,14	1,3	165

Table 4.5: Results for different activation functions

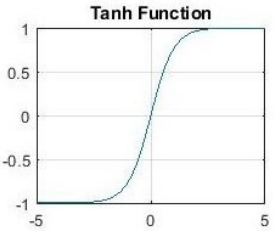


Figure 4.8: Tanh activation function (Nwankpa et al., 2018)

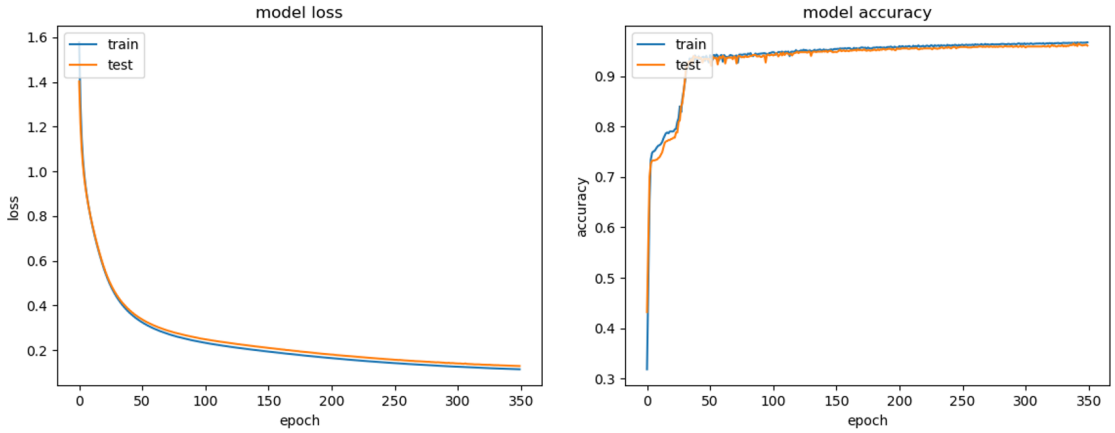


Figure 4.9: Logarithmic loss and classification accuracy for one fold with Tanh activation function

Dropout regularization

Dropout regularization tuning is a way to limit overfitting and improve the model's ability to generalize. In Table 4.6, the considered dropout rates are displayed with the corresponding results. It can be concluded that adding dropout regularization to the model is not improving the performance.

Algorithm 7: Find the best performing dropout regularization

Result: Best performing dropout regularization
Dropout regularization: [0.0, 0.1, 0.2, 0.3, 0.4];
for *dropout in Dropout regularization* **do**
  Train model(dropout);
end

Dropout rates	Loss	Accuracy [%]	Std accuracy	Time [s]
0,0	0,1251	96,80	1,63	225
0,1	0,1388	96,56	1,53	144
0,2	0,1592	93,90	2,50	242
0,3	0,1881	92,39	2,55	259
0,4	0,2386	90,87	1,80	369

Table 4.6: Results of tuning dropout regularization

Number of hidden layers and neurons

The number of hidden layers and neurons in each layer controls the representational capacity of the network. A larger network also requires more training and the batch size and number of training epochs needs to be optimized. Table 4.7 and Table 4.8 show the considered number of layers and neurons with a ReLU and Tanh activation function, respectively. Both activation functions are considered because the Tanh function showed the best results and the ReLU function is part of our base model.

Algorithm 8: Find the optimal number of hidden layers and neurons

```
Result: Optimal number of hidden layers and neurons
Activation function: [ReLU, Tanh];
Hidden layers: [1,2,3];
Neurons: [4,8,16,32,64];
for function in Activation function do
    for layer in Hidden layers do
        for neuron in Neurons do
            Train model(function, layer, neuron);
        end
    end
end
```

Neurons	Layers	Loss	Accuracy [%]	Std accuracy	Time [s]
4	1	0,1251	96,80	1,63	225
8	1	0,1000	97,37	1,14	141
16	1	0,0691	98,02	0,15	155
32	1	0,0598	98,35	0,17	189
64	1	0,0533	98,49	0,08	198
4	2	0,1188	95,69	4,98	195
8	2	0,0546	98,35	0,24	142
16	2	0,0452	98,61	0,11	169
32	2	0,0418	98,73	0,15	262
64	2	0,0406	98,77	0,17	225
64	3	0,0404	98,80	0,18	340

Table 4.7: Experimentation with number of neurons and hidden layers with ReLU activation function

Neurons	Layers	Loss	Accuracy [%]	Std accuracy	Time [s]
4	1	0,0929	97,60	0,18	123
8	1	0,0703	98,03	0,13	122
16	1	0,0602	98,35	0,13	147
32	1	0,0538	98,48	0,11	163
64	1	0,0498	98,59	0,09	178
4	2	0,1601	98,33	0,16	209
8	2	0,0438	98,60	0,24	135
16	2	0,0380	98,86	0,12	153
32	2	0,0385	98,87	0,20	245
64	2	0,0390	98,88	0,21	226
64	3	0,0392	98,90	0,12	321

Table 4.8: Experimentation with number of neurons and hidden layers with Tanh activation function

The result improves with the number of neurons and hidden layers for both activation functions. However, the results for the Tanh activation number are slightly better. The result with two hidden layers and 16 neurons for each layer is 0,26% better than the previous result, and from there on, the improvement is minimal. Therefore, the final model is equipped with two hidden layers of 16 neurons each.

4.5. Final model

Now the final lay-out of the model is found, the results of tuning the different hyperparameters can be used to determine the optimal NN configuration. First, the Tanh activation function shows the best result of the different activation functions. Secondly, there is no dropout regularization applied in the model. There are three different optimization algorithms which showed good results, Adam, Nadam,

and Adadelta. All three are evaluated ones more in the new model lay-out. Finally, Nadam shows the best results with a learning rate of 0,0015, see Table 4.9. The logarithmic loss and accuracy curves are shown in Figure 4.10. The final logarithmic loss, accuracy and standard deviation are 0,0368, 98,98% and 0,12, respectively. Both show a stable converging curve with only small noise which indicates a good fit model.

Learning rate	Loss	Accuracy [%]	Std accuracy
0,0009	0,0375	98,88	0,09
0,00095	0,0373	98,90	0,09
0,001	0,0372	98,95	0,24
0,0015	0,0368	98,98	0,12
0,002	0,0368	98,97	0,12
0,0025	0,0370	98,93	0,22

Table 4.9: Results of tuning learning rate of Nadam activation function

Hyperparameters	Value or option
Nr. of training epochs	350
Batch size	256
Optimization algorithm	Nadam
Learning rate of optimization algorithm	0.0015
Activation function	Tanh
Dropout regularization	0
Nr. of hidden layers	2
Nr. of neurons in each layer	16

Table 4.10: Overview of optimal hyperparameters

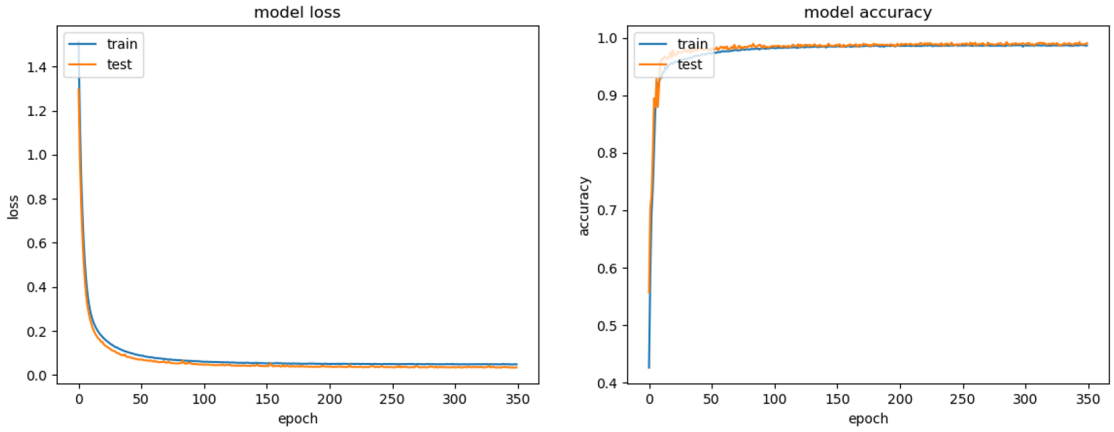


Figure 4.10: Logarithmic loss and classification accuracy for one fold of the final model with two hidden layers and 16 neurons

The test loss and accuracy both show a little gap with the training loss and accuracy. The fold of these curves has created a relatively easier test dataset compared to the training dataset. Because of this test dataset, it is easier for the model to predict the correct output, and this creates this small gap between test and training curves.

An overview of the NN's lay-out is showed in Figure 4.11, and an overview of the optimal hyperparameters is showed in Table 4.10. As discussed in Section 3.2.6, much research is done to determine the number of hidden layers and neurons scientifically. So far, it is still not possible to determine the number of hidden layers and neurons scientifically, and the experiments in this research showed comparable results for NNs with one and two hidden layers. However, the training took less time for the NN with two hidden layers and 16 neurons than the NN with one hidden layer and 64 neurons.

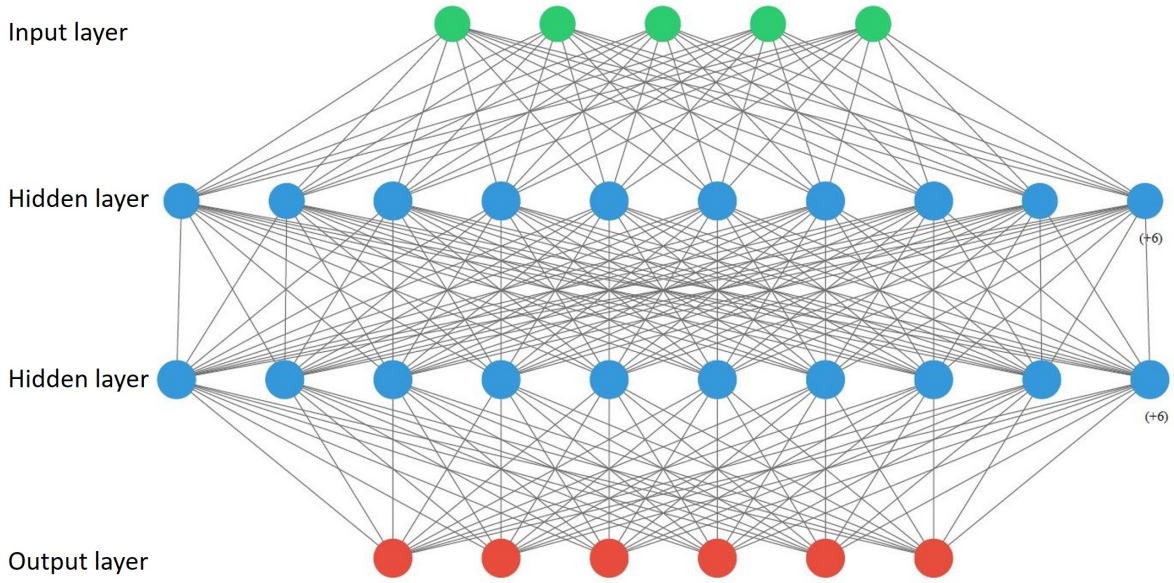


Figure 4.11: Final NN

4.6. Verification

During the entire process of training and hyperparameter tuning, the model is checked and verified according to the test dataset. However, the correctness of a classification model can be evaluated in several other ways. Several performance measures can be done to gain more insight into the model. These tests will be discussed in this section.

Hand verification

First, manual verification is done to check if the predicted values are indeed similar to the labelled classes. This is done on the first 100 entries (approximately 10%) of the test dataset. The first 100 entries can be found in Appendix C. The accuracy of the prediction of these first 100 entries is 98%. Two predictions are wrong where the predicted class is 2 instead of 4. This is explained by the resemblance of the dataset, where there is some overlap in data. This can also be seen in Figure 4.1. 98% is corresponding to the accuracy of the model predictions of the previous section.

Performance measures classification

As discussed in Section 3.2.5, precision, and recall are two important measures of a classification model. The Skikit-learn library has two convenient functions which automatically calculates the confusion matrix and precision and recall of the model. Table 4.11 and Table 4.12 shows the confusion matrix and precision and recall for the six different classes, respectively. The last column of Table 4.12 (”support”) shows the total number of samples which are classified in that class.

		Predicted					
	Class	1	2	3	4	5	6
Actual	1	95	0	0	0	0	0
	2	5	304	0	0	0	0
	3	0	2	409	0	0	0
	4	1	3	0	180	0	0
	5	0	0	0	0	112	0
	6	0	0	0	0	0	89

Table 4.11: Confusion matrix

Class	Precision	Recall	Support
1	0,94	1,00	95
2	0,98	0,98	309
3	1,00	1,00	411
4	1,00	0,98	184
5	1,00	1,00	112
6	1,00	1,00	89

Table 4.12: Precision and recall of the NN

Figure 4.1 in Section 4.2 already shows a resemblance between classes 1 and 2, classes 1 and 4, and between class 2 and 4. This already indicates that the prediction of these classes might experience problems because the classes are not unique based on their inputs. Table 4.11 and Table 4.12 shows this problem as well. Here it can be seen that the prediction of classes 1 and 2 is not perfect.

Since the accuracy is almost 100%, the precision and recall show both high percentages as well. However, from Table 4.12 can be concluded that recall is slightly better than precision. The recall percentages can be verified with Table 4.11, where it can be seen that the model made some mistakes in predicting the right output classes for class 2 and 4.

Random verification

As mentioned at the beginning of Section 4.4, during hyperparameter tuning the randomness is fixed to compare different hyperparameters. However, the strength of NNs is stochasticity. Therefore, different random seeds are evaluated to verify the stochasticity of the model. Table 4.13 shows the result of ten different seeds. From this table, it can be concluded that the model's performance is slightly better or worse, depending on the seed. However, the overall performance is comparable for all different seeds.

Random seed	Loss	Accuracy [%]	Std accuracy	Time [s]
1	0,0447	98,86	0,16	156
2	0,0372	98,96	0,25	155
3	0,0411	98,74	0,14	165
4	0,0601	98,44	0,04	148
5	0,0600	98,25	0,16	158
6	0,0514	98,49	0,07	165
7	0,0536	98,52	0,15	144
8	0,0392	99,10	0,14	155
9	0,0638	98,37	0,08	171
10	0,0517	98,61	0,07	169

Table 4.13: Final NN verification with 10 random seeds

Comparison of performance with literature

The performance of the NN is high and raises the question if this performance is reasonable. Therefore, the performance is compared with different studies to NNs. The study of Er et al. (2012) about predicting the diagnosis of 324 patients with a NN used a dataset with 34 variables and one output label. One of these 34 variables contains the same values as the output label. Because of this, the model was able to predict the correct output labels with 98% accuracy. In this research, there is no input variable with similar values as the output classes. However, as mentioned in Section 4.2, the classes are almost entirely separable by their input variables, and only several data samples show overlap to other output classes. This is not comparable to the result of the study of Er et al. (2012) because now the combination of input variables defines the output class instead of 1 input variable which contains the same information as the output class.

Furthermore, different studies among which the study of Brownlee (2016a) shows that data profoundly influences the performance of a ML model. The quality of the training data defines the quality of the model. Besides the quality of the data, also the amount of data is essential as ML model performances improve significantly with more data. In this research, the dataset used to train the model consists of 12.000 different samples and show excellent performance of the model. However, it is interesting to check the relation between the size of the dataset and the performance. It is easier and faster to train a model on a smaller dataset compared to a large dataset. The relationship between the size of the dataset and the performance is shown in Table 4.14 and Figure 4.12.

Size	Loss	Accuracy [%]	Std accuracy
12.000	0,0368	98,98	0,12
4.000	0,0539	98,90	0,12
3.000	0,0685	97,83	0,37
2.000	0,0866	98,10	0,37
1.500	0,0676	97,13	0,43
1.000	0,1111	96,20	0,60
500	0,1553	93,60	1,20
250	0,2749	90,80	5,08
100	0,3387	84,00	10,20

Table 4.14: Relationship between performance and dataset size

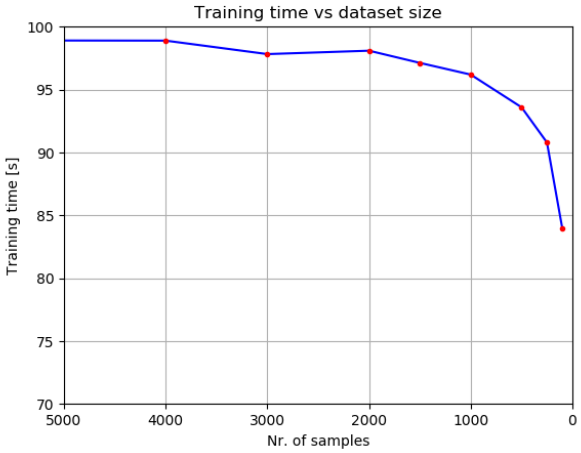


Figure 4.12: Relationship between performance and dataset size

From this table and figure can be concluded that the dataset size does indeed influence the performance. The datasets with a minimum size of 2.000 samples still show similar performance and datasets with less than 2.000 samples show a decreasing performance. This indicates that the model's ability to be trained on a smaller dataset is less compared to larger datasets. Figure 4.12 also show a stabilizing performance for the model trained with at least 2.000 samples, which indicates that generating more data will not improve the performance of this model significantly.

Finally, many research is done to the MNIST dataset. This dataset contains handwritten digits and is an extensively used example for NNs. Standard implementations of NNs are able to achieve an accuracy of 98%. Achieving higher accuracies is difficult and requires optimal hyperparameter tuning. Both Deotte (2018) and Gupta (2020) described their efforts to achieve accuracies above 98%, which comes down to hyperparameter tuning and improving the quality of the data. However, these studies show the possible performances of NNs, which also confirms that the results achieved in this research are possible with a NN.

K-fold cross validation

As mentioned in Section 4.3, the training of the model in each configuration is verified by a *k*-fold cross validation. *K*-fold cross validation divides the dataset into *k* different training and test splits. Then, the model is trained according to these *k* different training datasets and verified according to the test datasets. This results in *k* different models and performances. The dataset in this research is divided into ten different folds. The results for the ten different folds are shown in Table 4.15. The results show a stable logarithmic loss and accuracy over all ten different folds.

K-fold	Loss	Accuracy [%]
1	0,0352	98,92
2	0,0396	98,83
3	0,0354	98,91
4	0,0358	98,92
5	0,0349	99,08
6	0,0378	99,25
7	0,0361	98,92
8	0,0400	98,92
9	0,0371	99,08
10	0,0358	99,00
Average	0,0368	98,98

Table 4.15: K-fold cross validation

Size of dataset
The dataset is split into six different datasets containing two thousand samples each to validate the relationship between the size of the dataset and performance discussed in the previous section. The model is trained on these six different datasets, and the result is shown in Table 4.16. The performance of the model is comparable for five of the six datasets. However, the performance of the model on one of the datasets is lower compared to the others, which is caused due to the low precision of output class one. In the test dataset are relatively many samples with output class one, which is hard to distinguish from other output classes based on their input variables. The distribution of the samples over the six different datasets is done randomly so this would indicate a higher performance of the model on another dataset. Table 4.16 also shows one dataset where the performance is much higher compared to the other datasets. In this dataset are fewer samples that are hard to distinguish from other samples, and this causes a higher performance for this dataset.

Part of dataset	Loss	Accuracy [%]	Std accuracy
0-1999	0,0872	98,10	0,37
2000-3999	0,1111	95,65	0,55
4000-5999	0,0872	97,75	0,25
6000-7999	0,0663	98,45	0,42
8000-9999	0,0604	98,25	0,34
10.000-11.999	0,1500	99,95	0,15

Table 4.16: Result of model trained on six different datasets containing 2.000 samples

4.7. Overview

In this chapter, the development of the model is discussed according to the requirements specified in Chapter 3. The NN is tuned according to the hyperparameters specified in Section 3.2.6. The model is trained on a training dataset that is generated according to real-life variables from the physical environment. The samples of the training dataset are labelled with the desired output. An overview of the hyperparameters with the best performance is shown in Table 4.10. The optimal NN achieves a logarithmic loss of only 0,0368 and an accuracy of 98,98%.

Furthermore, the model is verified by executing manual verification, performance measures, random verification, comparison of performance with literature, and *k*-fold cross validation. From the verification can be concluded that the model is implemented according to the specifications. The achieved accuracy is correct according to the predictions by hand and is reasonable compared to literature. Furthermore, it can be concluded that the size of the dataset profoundly influences the performance of the model. Below 2.000 samples, the performance of the model decreases significantly.
One of the design requirements is to achieve a high recall because this indicates that the actual class is recognized correctly, which is essential to predict the right countermeasure. The recall of the model is better than the precision. However, both show almost perfect performance. Recall and precision are further validated with data from the packaging line of Alken-Maes in Chapter 5.

Now the model is trained according to the training dataset, and the model can be used in a physical environment. The model can deal with unlabelled data and can predict countermeasures based on this new data. In the next chapter, the model is applied to historical data from the packaging line of Alken-Maes to validate the working of the model in the physical environment.

5

Case study

The fifth step of the System Engineering methodology is component integration and testing. As discussed in Section 1.2, the packaging line of Alken-Maes is used as a use case in this study. The NN is developed and trained in the previous chapter and will be used to predict output classes based on real values retrieved from the packaging line of Alken-Maes. This chapter answers the third sub-question: *How can real-time decision making be applied to an industrial asset?* First, the data of the packaging line is collected and prepared. Section 5.2 and Section 5.3 discusses the prediction and validation of the model in the production environment, respectively. Continuing on the validation, Section 5.4 describes a simulation of the model to the data of the packaging line. Finally, the model is applied to a second machine of the packaging line to verify the results of the Filler and discuss the importance of line balancing in Section 5.5.

5.1. Data collection and preparation

The packaging line of Alken-Maes is connected to a new-installed IoT platform which enables the collection of data. Data from the packaging line is tagged in the IoT platform to be saved in a database or to retrieve the data in real-time.

Data points between the 4th up until the 10th of May 2020 will be further used to validate the model. The data is collected from the cloud and saved locally as a *.csv* file. The first step is to combine this event data with data from a reference database to connect every tag to the corresponding name/code of this tag. The next step is to filter the data, so only the relevant data remains. This data format corresponds to the training data format and is, therefore, suitable for validation. Similar to the training dataset, only the top three occurring errors are considered. The filtered data consists of:

- Timestamp
- Error
  - Error 1: External 3527 - Can lack at inlet 1
  - Error 2: Operator Guard 0043 - Low speed inlet conveyor
  - Error 3: External 3549 - Outlet can too full
- Program of the Filler
- Speed of the Filler

The information of the packaging line at one moment is scattered over multiple lines and must be combined into one line of data. Furthermore, data is only stored in the database when the value or status has changed at that moment. For example, if the speed of the Filler is constant for half an hour, only one data sample can be found in the database corresponding to the first next deviation from the previous speed measurement. To fill the gap between consecutive speed measurements, equally spaced time measurements are given the last saved speed measurement from the database.

The same is done with the status of the Depalletiser. The entire dataset of the Depalletiser is retrieved from the database and filtered, so only the status of the Depalletiser is left. The same yields for this data as for the speed of the Filler. The previous status of the Depalletiser is added to all the empty timestamps and then merged into the dataset. Now, the dataset contains the following information: the timestamp, the program, the speed of the Filler, the error, and the state of the Depalletiser.

The last step is to add the inlet speed of the Filler into the dataset. The inlet speed of the Filler is not measured, and therefore, the inlet speed is generated based on the Filler speed. It is assumed that the inlet speed can never be higher than the Filler speed or lower than 80% of the Filler speed:

$$0,8 * Speed_{Filler} \leq Speed_{Inlet} \leq Speed_{Filler} \tag{5.1}$$

The last step is to normalize the data into the same scale as is done with the training dataset. The normalized data has a value between 0 and 1, corresponding to the format of the training dataset. The total amount of samples for this time window is 1071. Every sample of the dataset refers to an occurred error.

5.2. Prediction

Now, the defined and prepared dataset is fed into the model. The result of the model is a predicted output class for each sample of the dataset. Keras is equipped with a function *'predict'*, which predicts the output class for each sample, corresponding to the input variables.

It takes 0,034 seconds to predict the entire dataset of 1071 samples and it took only 0,012 seconds to predict one sample. The result of the prediction can be found in Table 5.1, which shows the number of predicted outputs per class.

Output class	Number of samples
1	1
2	23
3	18
4	850
5	173
6	6

Table 5.1: The number of predicted outputs per class

Interestingly, class 4 and 5 represent 79,4% and 16,2% of the predicted classes, respectively. The training dataset contains two configurations where the required output class is 4 and one where the required output class is 5. To check whether the predictions are right, 20 samples are checked by hand. The complete overview of the results of this verification can be found in Appendix D.1. During verification, it immediately appears that the dataset used for training the model does not satisfy the real dataset of the packaging line. Half of the samples do not correspond to any of the different generated groups of the training dataset. These samples contain the following values:

- Program: 1
- State Depalletiser: 1
- Speed Filler: 0-740 [cans/min]
- Inlet Speed: 0-740 [cans/min]
- Error: 1

The training dataset does not contain an output class for these samples. The model predicts an output class that corresponds best to these samples but does not contain the same values. In this case, class four is predicted because this class corresponds the best to the training dataset. Therefore, the training dataset is expanded with samples that cover the requirements of the dataset of the packaging line. An overview of the new training dataset is shown in Table 5.2, where the seventh column is added to the existing dataset.

Variables	1	2	3	4	5	6	**7**	8	9	10	11	12	13
Program	1	1	1	1	1	1	**1**	1	1	1	1	1	1
State Depalletiser	1	1	2	2	1	2	**1**	1	2	1	2	1	1
Speed Filler [cans/min]	740-760	740-760	740-760	740-760	740-760	0-750	**0-740**	0-750	740-760	0-750	0-750	0-750	740-760
Inlet speed [cans/min]	740-760	600-750	0-750	740-760	740-760	0-750	**0-740**	0-600	0-600	0-600	0-600	0-750	740-760
Error	1	1	1	1	1	1	**1**	2	2	2	2	3	3
Measure/ Class	1	2	3	3	4	3	**2**	2	2	4	3	5	6
Number of samples	1000	1000	1000	1000	1000	1000	**1000**	1000	1000	1000	1000	1000	1000

Table 5.2: Overview of trainings dataset

The model is again trained on this extended training dataset with the same hyperparameters as in the previous model. The new logarithmic loss of the model is 0,0389, and the accuracy is 98,85%. A new prediction on the dataset of the packaging line is made. The predicted output classes for all the samples are summed up in Table 5.3. As expected, the predicted output of class 4 in the previous check is distributed over classes 2 and 4 because the samples which were predicted faulty to be class 4 are now predicted to be class 2.

Output class	Number of samples
1	3
2	502
3	18
4	369
5	173
6	6

Table 5.3: The number of predicted outputs per class

Again, twenty predicted output samples are manually verified according to the expected output classes of these samples. A complete overview of the results of this manual verification can be found in Appendix D.2. Compared to the previously trained model, all samples which did not correspond to an output class are now similar to the added samples. All these samples are predicted corresponding to the expected output classes of the training dataset.

The verification of the model to the real data from the packaging line of Alken-Maes shows the importance of training the model correctly. If the training dataset does not contain all possible combinations from the physical environment, the model can not predict the correct output class. However, the model will predict an output class which contains comparable input variables. In this case, the prediction is wrong because the desired output class is different.

5.3. Validation

Validation of the model is done by comparing the current situation of handling machine errors compared to the proposed automated decision making. Unfortunately, there is a lack of information concerning the current situation. Therefore, assumptions are made in consultation with experts from Alken-Maes to validate the model.

5.3.1. Current problem solving

Alken-Maes do not know the time to resolve an error in the current situation. However, an analysis of the data from multiple periods results in a distributed time to resolve an error according to a normal distribution, see Figure 5.1. From this analysis can be concluded that the average time to resolve an

error is approximately six minutes.

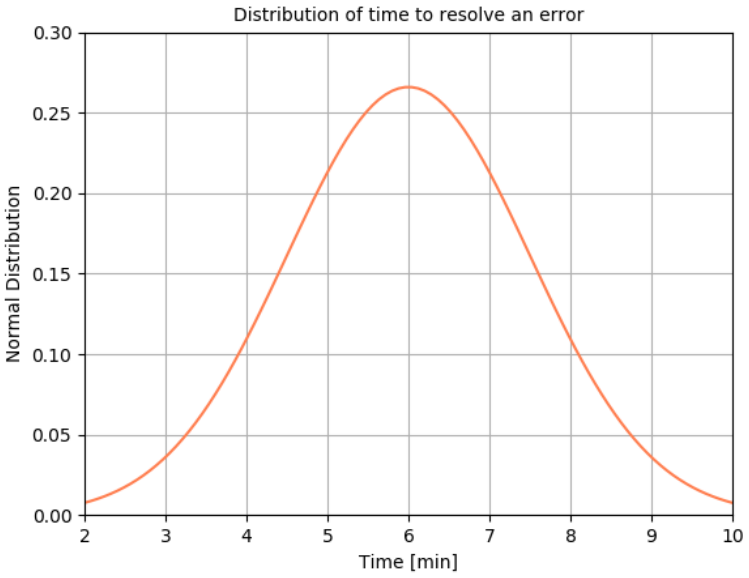


Figure 5.1: Normal distribution of the time to resolve an error

These six minutes are divided over different steps in the process of resolving an error. First, the operator normally responds in approximately 30 seconds from the moment he is notified about an error. Then, the operator walks to the PLC of the machine to determine which error causes the problem. Walking to the PLC takes, on average, 100 seconds. Then, the operator decides which countermeasure is executed to resolve the problem and walks to the location of the problem. Walking to the problem takes on average 60, and 30 seconds, respectively. Finally, the operator resolves the problems in 120 seconds. An overview of these steps can be found in Table 5.4.

Steps to resolve an error	Current situation [s]
Notification	30
Walk to PLC	100
Decide on countermeasure	60
Walk to problem	30
Resolve problem	140
Total time	360

Table 5.4: Time for every step between the moment an error occurs until the error is resolved

The times in the above table are assumed based on expert knowledge in combination with the information available about average times to resolve an error. The times in Table 5.4 are simplifications of the times in real-life. In real-life, the time to resolve errors varies according to the normal distribution of Figure 5.1. However, the uncertainty of this data is high because it is namely based on expert knowledge. Therefore, a total time to resolve an error of five and seven minutes is also taken into account to address this uncertainty. In Table 5.5, this uncertainty is divided over the different steps of the process to resolve an error. It is assumed that the time to walk to the PLC and the problem is the same in the different situations because the distance does not change. The uncertainty is divided over the three other steps of the process.

Steps to resolve an error	Current situation [s]		
Notification	20	30	40
Walk to PLC	100	100	100
Decide on countermeasure	30	60	90
Walk to problem	30	30	30
Resolve problem	120	140	160
Total time	300	360	420

Table 5.5: Uncertainty for time to resolve an error per step

5.3.2. Intelligent problem solving

The time for the proposed model to decide which countermeasure the operator has to perform consists of multiple components. First, once every second, the database is updated with the newest data inputs from the packaging line. Then, the data is prepared, which takes on average 0,032 seconds. The prediction of output classes takes 0,012 seconds. Furthermore, it is assumed that the communication back to the brewery consumes the same amount of time (one second) as the communication between the brewery and the database. This is summarized in Table 5.6. From this table can be concluded that the preparation of data and prediction of output classes takes a minimal amount of time compared to the communication of data between the brewery and the model.

Steps to predict	Time [s]
Data extraction from brewery	1
Preparation of data	0,032
Prediction of output class	0,012
Communication back to brewery	1
Total	2,044

Table 5.6: Overview of the time per step from data extraction to counter measure proposal

The total time to predict the correct countermeasure is approximately two seconds from the moment the error occurs. The operator receives a notification with the proposed countermeasure, walks towards the location of the problem, and resolves the error. However, it is assumed that an operator does not respond immediately to a notification (similar to the current situation). The time it takes for every step is added to Table 5.4, and summarized in Table 5.7.

Steps to resolve an error	Current situation [s]	Proposed situation [s]
Notification	30	30
Walk to PLC	100	-
Decide on counter measure	60	-
Walk to problem	30	100
Resolve problem	140	140
Total time	360	270

Table 5.7: Time for every step between the moment an error occurs until the error is resolved

From Table 5.7 can be concluded that the difference between the two situations is 90 seconds (1,5 minutes) in favour of the situation with a decision-making algorithm. This difference is based on the situation where the operator always decides to execute the right countermeasure. In practice, the operator does not always choose the right countermeasure, so (s)he has to execute a second or even third countermeasure, which takes another 120 seconds each time the executed countermeasure does not resolve the problem. However, this is not considered for the validation of the model because it is not known how many times an operator has to execute different countermeasures to resolve an error.

5.4. Simulation

Now the difference between the current situation and the new situation with the decision-making algorithm is known, the difference in total unplanned downtime can be simulated. Assumptions about the countermeasures must be made to simulate both situations. The considered countermeasures are already defined in Section 3.2.1 and are:

- Countermeasure 1: Unknown cause: investigate cause
- Countermeasure 2: Stop Filler
- Countermeasure 3: Solve a problem with Depalletiser
- Countermeasure 4: Solve can block at Filler
- Countermeasure 5: Slow down Filler
- Countermeasure 6: Solve can block at outlet Filler

Countermeasure 1 is defined as an *Unknown cause* and needs further investigation. Each time this countermeasure is proposed, an operator has to find the cause of the problem by himself. Therefore, the time to solve the problem when this countermeasure is proposed is similar to the current situation (six minutes). Countermeasures 2, and 5 propose to stop and slow down the Filler, respectively, which is done automatically and does not take time for the operator in both situations. Countermeasures 3, 4, and 6 propose a solution to the operator, and it is assumed that the problem is solved in 4,5 minutes.

As mentioned in Section 5.1, the total amount of errors in the considered time window is 1071. From these 1071 errors, 32 led to a minor stop of the Filler, which is considered as unplanned downtime. Therefore, the numbers of errors from Table 5.3 are multiplied by this ratio of minor stops per error. The result is shown in Table 5.8, where the predicted total number of errors and the number of minor stops per output class are shown.

Output class	Number of errors	Number of minor stops
1	3	0
2	502	15
3	18	1
4	369	11
5	173	5
6	6	0
Total	1071	32

Table 5.8: The number of predicted outputs and minor stops per class

The recall of the model is specified in Section 3.2.5 as the percentage of output classes that are recognized correctly. The recall of each class is considered in this simulation. For each class that is not predicted correctly, another two minutes are taken into account to solve the problem. The complete overview of the simulation is summarized in Algorithm 9. An overview of the results of the simulation is given in Table 5.9.

The simulation resulted in an improvement of the proposed situation of 16 minutes compared to the current situation, which indicates a reduction of downtime of 22,2% under the assumptions made. This improvement of 22,2% is according to the expected difference between both situations. The difference between 6 and 4,5 minutes is 1,5 minutes which is an improvement of 25%. Considering the accuracy of the model, the expected difference is a little less than 25%. Compared to the real situation of the packaging line in Alken-Maes, this is plausible.

As mentioned in Section 5.3.1, the uncertainty of the assumed times to resolve an error is addressed by applying the same calculation on 5 and 7 minutes to resolve an error. This results in a difference of 11 and 23 minutes between the current and proposed situation, respectively. This is a change of 31% and 44%, respectively, which indicates that the uncertainty in the assumptions affects the results of

Algorithm 9: Simulation of problem solving in both situations

```
Result: Times of problem-solving in both situations
Classes: [1, 2, 3, 4, 5, 6];
for Classes [1] do
    Total time current situation = (Nr. of errors) * (6 minutes);
    Time first attempt proposed situation = (Nr. of errors) * (6 minutes);
end
for Classes [2, 5] do
    Total time current situation = 0;
    Time first attempt proposed situation = 0;
    Time second attempt proposed situation = (Nr. of errors) * (1-recall) * (2 minutes);
    Total time proposed situation = (Time first attempt) + (Time second attempt);
end
for Classes [3, 4, 6] do
    Total time current situation = (Nr. of errors) * (6 minutes);
    Time first attempt proposed situation = (Nr. of errors) * (4,5 minutes);
    Time first attempt proposed situation = (Nr. of errors) * (1-recall) * (2 minutes);
    Total time proposed situation = (Time first attempt) + (Time second attempt);
end
```

		Current situation	Proposed situation			
Class	Nr. of stops	Total time [min]	Recall	First attempt [min]	Second attempt [min]	Total time [min]
1	0	0	1,00	0	0	0
2	15	0	0,98	0	0,6	0,6
3	1	6	1,00	4,5	0	4,5
4	11	66	0,98	50	1	51
5	5	0	1,00	0	0	0
6	0	0	1,00	0	0	0
Total time [min]		72				56

Table 5.9: Overview of results of simulation of problem solving in both situations

this simulation significantly. However, the actual downtime due to minor stops in that period is 84 minutes, which differs 12 minutes from the total simulated downtime (72 minutes), which is close enough to consider the initial results as realistic.

In the entire problem, only the three most occurring errors are considered. If more errors are considered, this will lead to a higher absolute difference. The same applies to the expansion of this decision model to multiple other machines of the packaging line. Now, only the Filler is considered. However, other machines experience errors as well, and the application of this model on these machines can achieve comparable results.

In the simulation, it is assumed that the operator makes the correct decision regarding the countermeasure because there is no information available about the ratio between taking the correct or wrong decision. This ratio will further increase the difference between the current and proposed situation in favour of the proposed situation.

It is assumed that the packaging line of Alken-Maes is on average, 6 days a week operational. The reduction of unplanned downtime of 16 minutes is multiplied by 52 to determine the reduction of unplanned downtime for an entire year. The reduction of unplanned downtime for an entire year is 832 minutes. The nominal speed of the Filler is 750 cans per minute, which results in a daily, weekly, and yearly improved production of 2.000, 12.000 and 624.000 cans, respectively. The improved production can also be translated into profit in terms of money. The average profit of one can is € 0.05, which

results in a daily, weekly, and yearly profit of € 100.-, € 600.-, and € 31.200.-, respectively.

The implementation of this system into the production environment of Alken-Maes is close to the suggested implementation of IoT systems together with AI techniques in the research of Patel et al. (2018) discussed in Section 2.5. However, the implementation of the model in this research goes beyond the research of Patel et al. (2018) because the operator does not have to troubleshoot an occurring problem.

5.5. Verification on second machine

Dataset of Shrink Packer

After training the model on the Filler, the model is implemented on another machine. This second machine is the Shrink Packer. The Shrink Packer shrinks a plastic wrap around a specified number of bottles. The final model from Chapter 4 is used, and a completely new dataset is generated. This dataset contains the four following input variables:

- Program of the Shrink Packer
- State of the Shrink Packer
- Speed of the Shrink Packer
- Error
  - Error 1: ALL306 - Minimal accumulation layer
  - Error 2: ALL321 - Slow down outlet
  - Error 3: ALL233 - Deformed pack at uphill conveyor
  - Error 4: ALL273 - Minimal level reel film

In contradiction to the dataset of the Filler, for the program of the Shrink packer are three options included in the dataset. Furthermore, three different states and the top five errors except the third error: *machine out of production* are included. The six countermeasures to resolve the errors are:

- Countermeasure 1: Slow down Shrink Packer
- Countermeasure 2: Stop Shrink Packer
- Countermeasure 3: Remove deformed pack at uphill conveyor
- Countermeasure 4: Fill reel film
- Countermeasure 5: Reset sensor reel film
- Countermeasure 6: Fill accumulation layer

The dataset is created and contains many more different combinations compared to the dataset of the Filler. Where the dataset of the Filler contains only 13 combinations, the dataset of the Shrink Packer contains 57 different combinations. The complete overview of the dataset can be found in Appendix E This extensive dataset is created to test the model on a (slightly) more complex dataset. The dataset is visualized according to the method described in Section 4.2.3, see Figure 5.2. From this Figure can be seen that many more small regions of data points show small overlap between each other, which indicates that the dataset is more complex than the dataset of the Filler. However, the coloured lines in Figure 5.2 are clearly separable from the other lines, which indicates that the model is able to achieve good performance again.

Results

The model is trained similar to training the model on the dataset of the Filler, among which, training the model on 10 folds. The time to train the model took more time due to the more extensive and complex dataset. Training the model took 70 seconds compared to 15 seconds of training the model for the Filler. However, after training the model, the logarithmic loss, accuracy, and standard deviation of the model are 0,0248, 99,26%, and 0.07, respectively. The logarithmic loss and accuracy curves for one

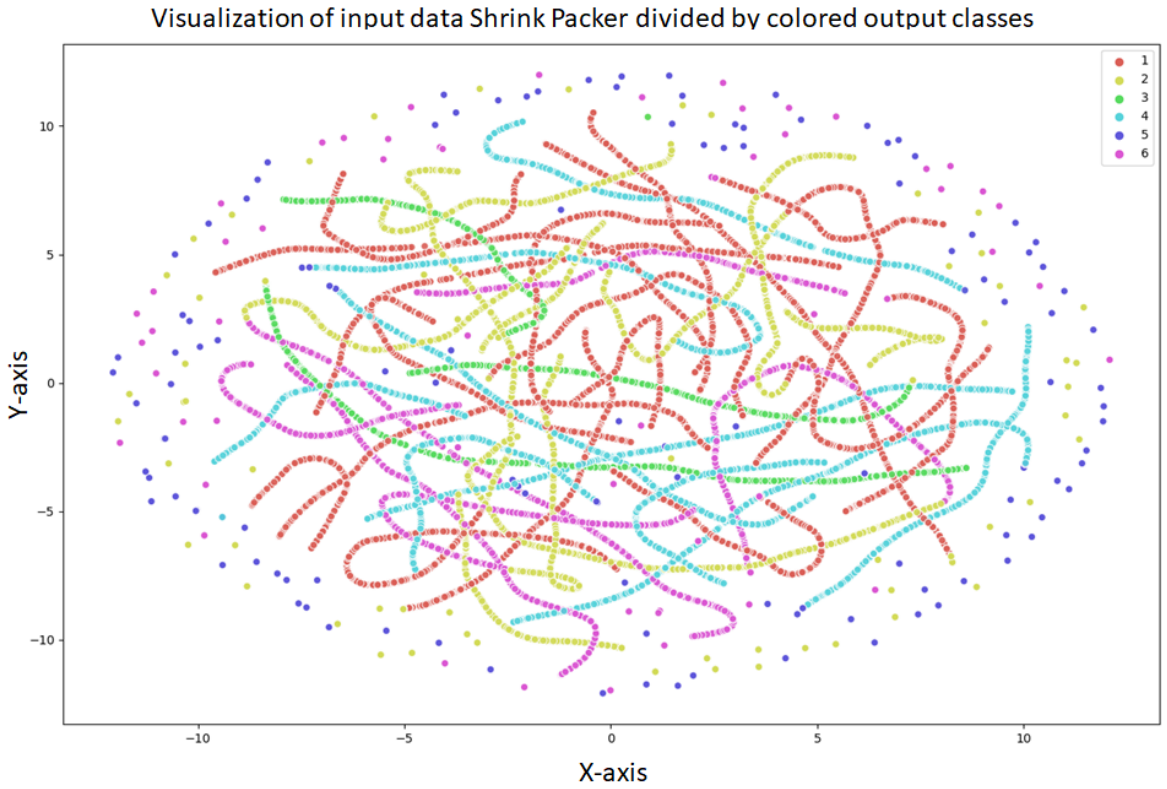


Figure 5.2: Visualization of input data Shrink Packer after dimensionality reduction

fold are shown in Figure 5.3. Both show a stable converging curve to the optimal performance.

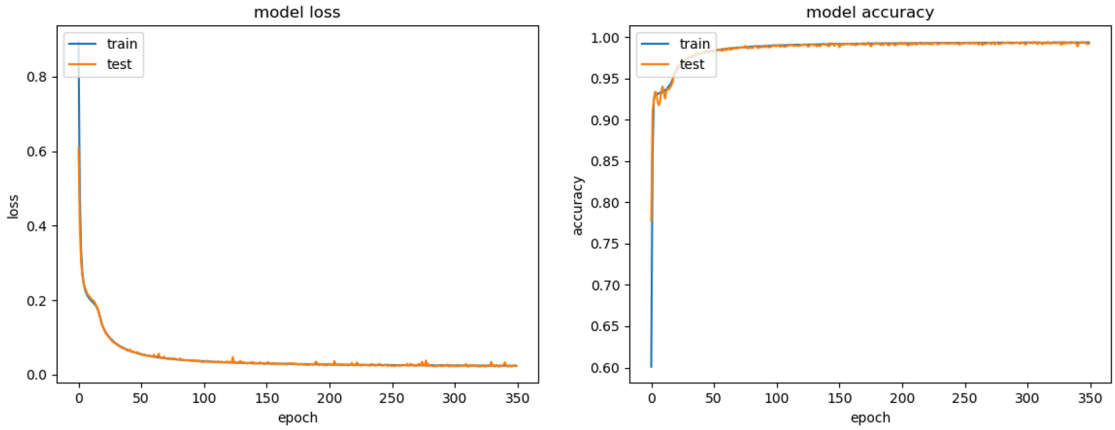


Figure 5.3: Logarithmic loss and classification accuracy for one fold of Shrink Packer model

From the 4th up until the 10th of May 2020, 121 errors occurred at the Shrink Packer, which resulted in 56 minor stops. The minor stops took on average 5 minutes which is a little less compared to the average time of a minor stop at the Filler. It is assumed that the time to solve an error in the proposed situation still takes 4,5 minutes. This results in a similar simulation than that of the Filler which is summarized in Table 5.10.

From Table 5.10 can be concluded that the absolute improvement for the Shrink Packer is smaller

		Current situation	Proposed situation			
Class	Nr. of errors/stops	Total time [min]	Recall	First attempt [min]	Second attempt [min]	Total time [min]
1	245/36	0	0,99	0,0	0,0	0,0
2	44/6	0	1,00	0,0	0,0	0,0
3	20/3	15	0,96	14,6	0,2	14,8
4	29/4	21	0,98	19,0	0,2	19,2
5	0/0	0	1,00	0,0	0,0	0,0
6	46/7	34	1,00	30,2	0,0	30,2
Total time [s]		69				64,2

Table 5.10: Overview of results of simulation of Shrink Packer in both situations

due to the smaller difference in time to resolve an individual error. Furthermore, it can be seen that countermeasure one is proposed most often to errors, and countermeasure five is not proposed to any error. However, countermeasure one and two are both countermeasures which do not require any actions of the operator. Due to the combination of the number of errors solved by countermeasure one and that countermeasure one does not require any action of the operator also results in only a small improvement in the proposed situation.

Error propagation
As already discussed in Section 1.2.3, errors can propagate through the packaging line and causes errors or minor stops at other machines. Error propagation occurs when the buffers cannot compensate for the starvation of blockage of the machine. From the minor stops at the Shrink Packer, two are caused by the Filler, five by the Tray Packer and one by the Palletiser. The minor stops of the Filler are mainly caused internally, and the Depalletizer causes only four minor stops, see Table 5.11.

		Caused by				
		Depalletiser	Filler/Pasteurizer	Shrink Packer	Tray Packer	Palletiser
Affected	Depalletiser	0	0	0	0	0
	Filler/Pasteurizer	4	28	0	0	0
	Shrink Packer	0	2	54	5	1
	Tray Packer	0	0	2	86	2
	Palletiser	0	0	0	0	1

Table 5.11: Error propagation between machines of the packaging line

From Table 5.11 can be concluded that the error propagation between the Filler and the Shrink Packer is minimized to two minor stops of the Shrink Packer caused by the Filler. As discussed in Section 1.2.3 and further stressed by Elst et al. (2020), line balancing is in place to protect the bottleneck machine (slowest machine) from any minor stops caused by other machines. Typically, the Filler and the Pasteurizer are the slowest machine(s) (Elst et al., 2020). Table 5.11 indicates that the line balancing of the packaging line of Alken-Maes is working correctly as only the Depalletiser caused four minor stops at the Filler/Pasteurizer.

The implementation of the proposed decision-making algorithm on each machine will lead to a further reduction of error propagation as the errors are solved faster. The buffers have a specific capacity, and when the capacity is reached, the error propagates to another machine. If the errors are solved faster, the capacity of the buffers is less exceeded, and fewer errors propagate through the packaging line.

5.6. Discussion on results

First, assumptions are made to make simulation possible. These assumptions are made because the information from the real-environment is missing. The first assumption, the inlet speed of the Filler, is not available from the packaging line and therefore, the assumption of Equation 5.1 is made. Further-

more, the number of input variables is small because other input variables are not available. The small number of input variables led to a small model, which needs further expansion for implementation in the brewery.

The input variables are existing values, while the countermeasures are fictional but realistic. However, the relation between the input variables and the countermeasures is also fictional. Many countermeasures are now distinguished by the difference in Filler speed, Shrink Packer speed, or by the inlet speed of the Filler. This is the result of a lack of input variables. It is more realistic to distinguish countermeasures based on several different variables of a machine. However, the model proves the decision-making requirement of the model based on input variables.

As mentioned in Section 1.1, a stable start-up of greenfield projects is not reached at Heineken due to a disconnection between theory and reality. The experience of operators is proved to be important in achieving a stable start-up. However, the equipment of breweries around the world consists only of several different types. For example, a brewery in Mexico can have the same packaging line as the brewery of Alken-Maes. The introduction of the developed service at Alken-Maes can be translated to any other brewery with the same equipment relatively easily. The results achieved in the brewery of Alken-Maes can be much higher if the service is applied in a greenfield project where the operators do not have the experience to solve problems on their own. This results in a knowledge-transfer between the breweries, which can improve the productivity of every affiliated brewery. However, the relatively easy translation from the developed service to any other brewery with the same equipment requires experts to implement the service in these breweries. The research of Elst et al. (2020) shows that two similar packaging lines in the same brewery perform different, which indicates that although the equipment is similar, local factors are in play as well. Therefore, the implementation of this service in any new brewery requires verification and validation by experts.

5.7. Conclusion

In this chapter, the model is tested against the physical environment. The sub-question which is answered in this chapter is: *How can real-time decision making be applied to an industrial asset?* The developed model is a simplification of a real-world problem because it does not contain every aspect of the packaging line of Alken-Maes. However, most of the variables correspond to the real variables of the packaging line.

First of all, data preparation is an essential step in the integration of the model. Data is retrieved from the physical environment in different formats and different databases. Combining all information is a case-specific job and depends on the specifications of the data. Furthermore, labelling is essential in classification problems because it is not possible to classify data without proper labels in a training dataset.

The developed decision model replaces the intelligent thinking of the operator once the model is trained. Based on the physical properties at the moment of an occurring error, the model decides which of the available countermeasures is most effective according to the training data. Once the data is prepared and the training dataset is covering the possible situations in the physical environment, it is possible to predict output classes with 98,85% accuracy. These countermeasures are proposed to the operator immediately when the errors occur. It only takes 2,044 seconds to retrieve data from the packaging line, predict a countermeasure, and communicate the proposal back towards the brewery. This enables a fast prediction of countermeasures to the operator. Compared to the current situation, errors can be resolved 1,5 minutes faster when the decision algorithm is applied.

The simulation of the current and proposed situations show the potential improvement of implementing the decision-making algorithm. However, the simulation is a simplification of the real-world problem, but it indicates the potential improvement. A weekly reduction of 16 minutes of unplanned downtime is achieved. Which translates into an estimated extra profit of € 31.200 annually.

Unfortunately, the model is not tested in a real-time environment due to the implementation possibilities

at the moment of model development. Therefore, it is hard to answer the sub-question from a practical point of view. However, in Chapter 3 the model requirements are discussed. The model requirements are met because the model can predict the correct countermeasure based on the input variables with high accuracy. Furthermore, the time to predict a countermeasure is significantly low, which enables resolving errors fast.

The NN in this research can predict the output classes for the top three errors of the Filler. However, the classification of data is a labour-intensive task, and it requires expert knowledge of the physical environment. When a service like this is extended to a complete machine or even an entire production line, the NN has to be scalable. This will be discussed in Chapter 6.

The application of the model on a second machine of the packaging line shows how the model deals with a more complex and extensive dataset. The model is able to predict with 99,26% accuracy, which is comparable to the accuracy of the Filler. However, due to the selection of countermeasures, the absolute improvement is smaller due to the smaller difference in time to resolve an individual error in both situations.

The implemented system is an excellent example of AI applied in the supply chain, which is already discussed from the literature in Section 2.5. The implemented system contains the reactive aspects from the research of Patel et al. (2018) and the predictive aspects from the research of Rossit et al. (2019) which are combined in this research. This enables the system to react to unexpected events and propose/predict counteractions.

6

Multi-machine environment

The two final steps of the System Engineering methodology are combined in this chapter. The previous chapter described testing and validating the NN on the considered machine in the physical environment, specifically. The final step is to describe the integration and validation of the complete system into the production environment. The fourth sub-question is answered in this chapter: *How can real-time decision making be applied in a production environment?*

First, Section 6.1 describes the failure propagation in the physical production environment. Then, Section 6.2 describes the implementation of the service on an entire packaging line. In Section 6.3 and Section 6.4, an architecture for implementing the service on an entire packaging line is proposed. Finally, Section 6.5 discusses the challenges of implementing the service in a production environment.

6.1. Physical production environment

Section 1.2 describes the physical environment of a packaging line. A packaging line consists typically of multiple machines connected by conveyors. Every machine contains typically between the 200 and 1.200 different errors. The study of Elst et al. (2020) shows that the performance of one machine affects the performance of the other machines in the same production line. If a problem appears at one machine, other machines might experience side effects; such as errors will raise at other machines in the packaging line. For instance, if a *can block* happens at the Filler, after a while, the machines downstream of the Filler will experience a *can lack*, and the machines upstream will experience a *can outlet too full*. This is also shown in Table 5.11 for the packaging line of Alken-Maes. Van der Elst et al. also analyzed the failure propagation of blockage events, which is shown in Figure 6.1.

This Figure shows that from 106 blockage observations at the Filler, 82 can be related to downstream failure occasions. From the 82 observations can 63 occasions be related to problems at the Labeller, and so on. The chapter discusses the importance of collaboration between machines in multi-machine environments. From the research of Van der Elst et al. can be concluded that machines in one production line are dependent on each other and events at one machine affect other machines sequentially. Machines affecting each other indicates the importance of addressing problems across an entire production line instead of focussing on one machine.

6.2. Monolithic Neural Network

The developed NN in Chapter 4 is a decision algorithm, trained explicitly for one machine. However, the failure propagation of one other machine is taken into account in the NN. The output classes of the NN consists of one countermeasure addressing a problem at the Depalletiser, see Section 3.2.1. The developed NN can be expanded to address all errors of the Filler or any other machine. However, it is already mentioned that many errors are an effect of problems elsewhere in the packaging line, which can be encountered by expanding the model with several input variables from other machines

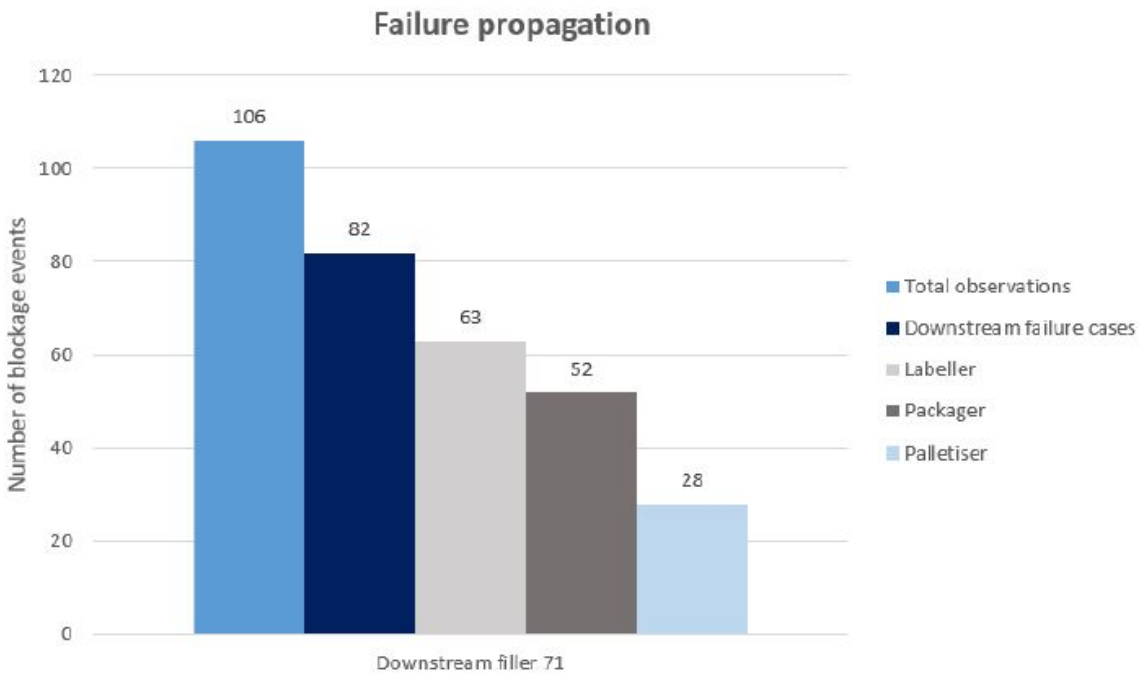


Figure 6.1: Failure propagation of blockage events (Elst et al., 2020)

combined with output classes that mention these machines. Ultimately, all variables from the entire packaging line can be combined into one NN together with expanding the number of output classes according to all countermeasures possible. This is called a monolithic NN.

In the real-life situation, multiple errors of different machines may arise at the same moment while they all have the same cause. This requires operators to decide what the cause of these errors is and which error needs to be solved first. However, this problem can also be addressed by the developed NN. Then, the final output of the model for an entire packaging line is a sequence of countermeasures which is executed by the operator. The sequence of countermeasures is dependent on all the input variables of the production line and the occurring errors.

Most of the used NNs have a monolithic structure and perform well on a small set of input variables (Wasilewska, 2018). A NN is already computational expensive compared to other ML algorithms. Expanding a NN with many more input variables requires much more data which again increases the required computational capacity. As mentioned in Chapter 5, the dataset of the Shrink Packer is more complex and contains more samples than the dataset of the Filler. The time to train the model of the Filler with a dataset of 12.000 samples is 15 seconds. The dataset of the Shrink Packer contains 57.000 samples and training the model took 70 seconds. If the dataset of the Shrink Packer is reduced to 19.000 samples, it only takes 22 seconds to train the model. Figure 6.2 shows the relation between the dataset size and the time to train the model. It shows that the time to train the model increases almost linear. However, increasing the complexity is not taken into account because the number of input variables and output classes has not increased.

Expanding the number of input variables and output classes further increase the complexity of a NN and the performance will decrease. The limit of using such a NN is the computational power to train the NN, which heavily depends on the size of the data (Donges, 2019). Adding more variables for one machine is possible, and the time to train the model will increase but adding entire machines to the NN increases the complexity massively. Therefore, Modular Neural Networks (MNNs) are studied and show excellent performance compared to monolithic NNs. The next section discusses the concept of MNNs and shows the benefits of MNNs in a multi-machine environment.

Figure 6.2: Time to train the model vs size of the dataset

6.3. Modular Neural Network

MNNs are based on the modularity of the human brain (Chen, 2015). The human brain uses modularity to use specific modules together to solve complex tasks. Besides modularity, the human brain also hierarchically processes information. Hierarchical information processing enables the human brain to solve complex tasks in an effective way.

MNN consists of multiple separated NNs, each working independently on its domain. Each network is built and trained for their specific task. The combination of the individual networks results in the final decision. The decision system can be implemented by a logical majority vote function, another NN, or a rule-based expert system (Wasilewska, 2018).

6.3.1. Benefits Modular Neural Networks

First of all, MNNs learn according to an alternative methodology (Chen, 2015). The complex optimization problem can be solved by an ensemble of simple NNs, which avoids the complicated optimization encountered in monolithic NNs without decreasing the performance. Furthermore, MNNs can flexibly use prior knowledge and integrates this in learning. A problem faced by monolithic NNs during learning is temporal and spatial cross-talk while MNNs are immune to this phenomenon (Haykin et al., 2009). Next, MNNs yields a better generalization than monolithic NNs (Auda and Kamel, 1999). Finally, modularity in MNNs enables efficient and robust computation (Auda and Kamel, 1999). This computational advantage makes MNNs scalable to large-scale implementation.

6.3.2. Architectures

There are many different MNN models which typically differ in there decomposition or aggregation. Common used MNN architectures are decoupled modules, other-output modules, hierarchical networks, multiple experts network, Adaptive Resonance Theory-Back Propagation networks and ensemble networks (Auda et al., 1996; Chen, 2015; Chris Tseng and Almogahed, 2009; Wasilewska, 2018).

Another MNN architecture is constructive MNN, see Figure 6.3 (Chen, 2015). Constructive modularization learning is developed for supervised learning problems. The divide-and-conquer principle is explicitly applied in this architecture. The idea behind this method is to divide a complex problem into several subproblems. These subproblems are easily solvable by NNs matching the requirements of the subproblems. The original problem is solved by combining the solutions to the subproblems. The NNs for the subproblems are relatively simple compared to a monolithic NN, and the architectures might differ per subproblem.

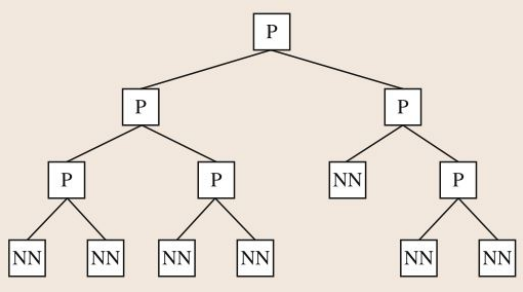


Figure 6.3: Self-generated tree-structured MNN (Chen, 2015)

The divide-and-conquer principle first divides the input space into overlapping subspaces, which facilitates dealing with various uncertainties. Next, in the conquering space, the NN works on the given subspace to complete the subproblem. These principles create a tree-structured MNN with a learnable partitioning mechanism that is placed at all intermediate levels of the MNN.

6.4. Architecture for service integration

Due to the multi-machine environment of the packaging line, multiple NNs are developed for different machines. The divide-and-conquer principle does not apply in this case because the main problem is already divided by the physical environment. Every subproblem considers its machine and predicts a countermeasure to resolve the problem on the respective machine. The output of the main problem consists of consecutive countermeasures for each machine defined by the general decision algorithm.

This architecture is close to that of a Mixture of Experts (MoE), see Figure 6.4. A MoE architecture consists of small clusters of neurons, each specialized on a part of the problem (Moussa, 2004). However, a MoE architecture decides dynamically which experts are used based on the input variables. This is done by a gating network that decides what expert to use.

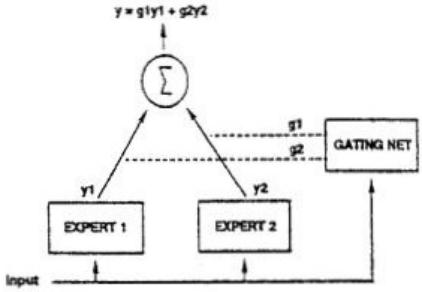


Figure 6.4: MoE architecture (Auda et al., 1996)

The outputs of the different NNs are combined as inputs in the general NN. This general NN is trained to predict a sequence of countermeasures according to the separate countermeasures proposed in the subproblems. An example of a MNN with expert NNs for each machine is shown in Figure 6.5.

An example of subproblems is discussed in the previous chapter, where a model is developed for the Filler and the Shrink Packer. Together with Table 5.11, it is possible to handle the propagation of errors and propose the right sequence of countermeasures to operators.

6.5. Challenges of implementation

The challenges of the system requirements are split into two parts, from which the challenges of the service requirement is already discussed in the previous chapter. The requirements of the infrastructure consist of access to sensors, an online database, and connections between the components. These requirements are partly met because the implementation of the IoT platform on the packaging line of Alken-Maes is still under construction.

The first requirement is also discussed in 5.6. The developed algorithm will benefit from access to more sensors of the packaging line. Now, output classes are distinguished on differences in speed while in

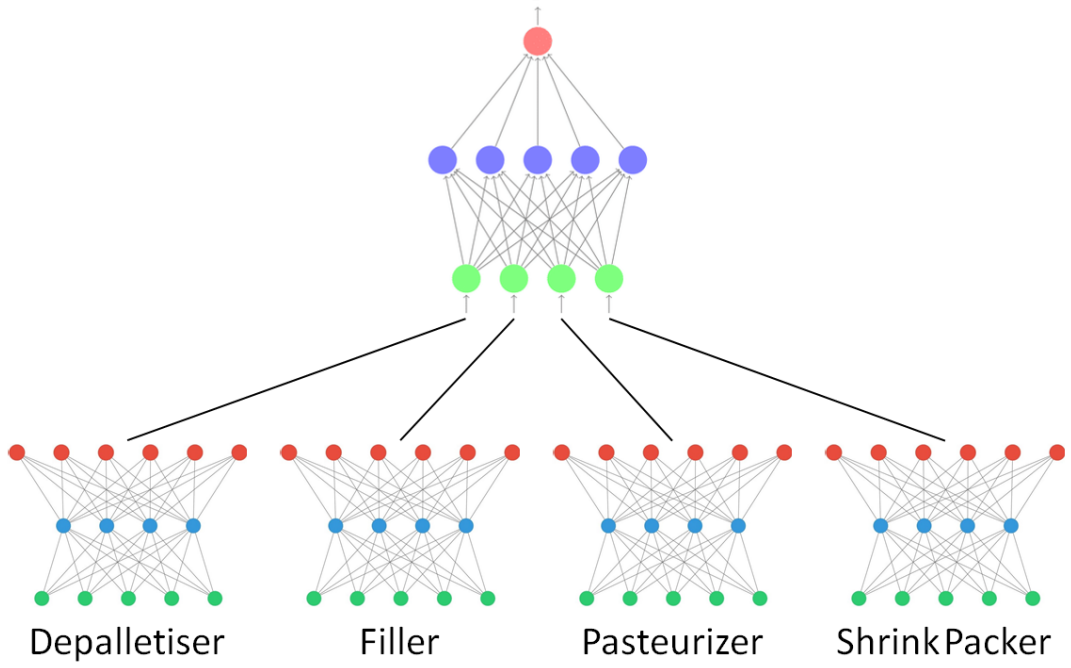


Figure 6.5: Example of a MNN with an expert NN for each machine

real production environments, this will be difficult. However, the current IoT platform offers access to the available sensors and information is retrieved from the packaging line.

The latter two system requirements are closely connected as the online database is currently accessible, and the research is based on the information in this database. However, the implementation of this model requires real-time access to the database and sensors to support operators in real-time. This also requires a connection between the online model and the operator in the brewery as the output of the model needs to be communicated to the operator. Due to these missing connections, it was not possible to test the model in the real-time production environment of Alken-Maes.

According to the five-dimensional model in Section 2.3.3, the fifth requirement is a virtual model that represents the physical model. A virtual model can consist of geometric models, physics models, behaviour models, or rule models. However, a virtual model goes beyond the scope of this research and does not contribute to the user requirements of this research. However, a virtual model provides insights into the physical asset and the output of the model.

Furthermore, the labelling of data is a massive task and requires much time of experts. Especially, labelling data from the entire packaging line in every possible situation is an enormous task. This problem can be encountered by starting to label every occurring error during operation. When an error occurs, the operator starts resolving the error as they usually do. When the operator is finished, and the error is resolved, operators indicate which countermeasure is applied. The corresponding label is given to the data sample at that moment. This method enables the model to learn on the spot, further improving its performance.

6.6. Conclusion

The defined user requirement in this research is to reduce the unplanned operational downtime by supporting operators to resolve errors faster to increase machine utilization. As discussed in Section 1.2, an operator decides which countermeasure is performed to resolve the occurring error. The operator takes a decision based on his experience and knowledge of the production line. If the countermeasure

does not resolve the problem, a second countermeasure is performed. This continues until the error is resolved.

The sub-question answered in this chapter is: *How can real-time decision making be applied in a production environment?* The application of the decision model on one machine is already discussed in the previous chapter. The application of the proposed model in a complete production environment requires the integration of multiple decision models. Every machine has a dedicated decision model, and these decision models are experts on their subproblem. The output of each decision model is combined in one general decision model that determines a sequence of countermeasures.

A suitable architecture for this MNN is a combination of a constructive MNN and a MoE architecture. However, the integration of multiple NNs in one MNN is not applicable to this research and is one of the recommendations for further research.

Furthermore, the implementation of the service requires different aspects of the infrastructure. Without extending the number of available sensors of the packaging line, it is not possible to implement the service on the entire packaging line. Furthermore, real-time connections need to be established between the (online) decision-making algorithm, the sensors of the packaging line, and the operators working in the brewery. Without these two requirements, it is not possible to apply the service on the packaging line because not all the errors and countermeasures can be distinguished by the current variables and without real-time connections, it is not possible to communicate any decision made.

7

Conclusion and Recommendations

This research studies how operational productivity can be improved using intelligent intervention proposals in case of unplanned downtime occasions. The user requirements define this goal. Then, system requirements are determined to define the requirements of such a system in a production environment. One of these requirements is to design a decision model to propose interventions to operators. This chapter presents the final conclusions in Section 7.1 and recommendations for further research in Section 7.2.

7.1. Conclusion

The never-ending strive to reduce waste in production processes and maximize machine utilization benefits from the introduction of Industry 4.0 and its tools. These tools are used to support the goal of this research to improve operational productivity in a production environment. One of these tools is a digital twin which enables the development of real-time services. This service is required to answer the main research question: *How to improve the operational productivity by using artificial intelligence-driven intervention proposals in case of unplanned downtime occasions?*

From this research can be concluded that only four of the five requirements of a digital twin are necessary for the implementation of an intelligent real-time algorithm. These four requirements are a physical asset, the access to real-time data of this asset, a service model, and the connections between these three components. The fifth requirement, a virtual model, is not required for the development of a decision-making service.

If the digital infrastructure between the physical asset, databases, and cloud computing services is present, it is possible to develop a service that improves operational productivity. Furthermore, it is possible to develop any other service based on this infrastructure and real-time or historical data.

The operational productivity is improved by implementing a service that decides which countermeasure an operator has to execute in case of an occurring error. This decision algorithm is a multi-class classification algorithm that is trained based on a predefined training dataset. The chosen multi-class classification algorithm is a neural network. The performance of the neural network is heavily dependent on the compiled training dataset. Therefore, compiling a training dataset requires experts that can label input samples. When the training dataset is compiled correctly, it is possible to predict correct countermeasures to operators with high accuracy. This enables operators to reduce the time to resolve errors faster which reduces the unplanned downtime and improves operational productivity.

The neural network is applied to the Filler of the packaging line of Alken-Maes. Simulation of the proposed method shows the potential operational productivity improvement. The neural network predicts the correct countermeasure with 98,98% accuracy, which results in a weekly reduction of unplanned downtime of 16 minutes. This indicates a yearly improvement of the operational productivity of 624.000 cans, which corresponds to 22,2% reduction of unplanned downtime for downtime occasions where

the operator has to resolve the problem. Implementing the service on each machine of the packaging line and considering every error of the machines further reduces the unplanned downtime.

Furthermore, the use of a neural network enables solving an infinite amount of possible multi-class classification problems, which makes this algorithm suitable in the case of an always-changing physical environment. Future expansions of the physical environment by any asset or sensor can be addressed by this algorithm as well. These expansions enable the continuous improvement of operational productivity in the future.

7.2. Recommendations

In this section, recommendations are proposed to extend the current research. The recommendations are divided into two sections. First, the recommendations for further academic research are proposed, and Section 7.2.2 discusses the recommendations for Heineken.

7.2.1. Recommendations for further academic research

As already mentioned in the previous chapter, labelling input data is a labour extensive task. Operators or other experts have to label the input data with a particular countermeasure. Therefore, it is recommended to study the possibilities of active learning to reduce this task of operators. Active learning is a form of learning where the learning algorithm controls input samples by a specific sample selection process. It is a combination of supervised and unsupervised learning where operators only have to label subsets of the training dataset and label specific occasions during operation.

Furthermore, Chapter 5 discusses the implementation and testing of the model on real data of the packaging line. This implementation showed the importance of proper training dataset because the training dataset did not match all occurring situations in real-life. The training dataset in this research has (almost everywhere) clear boundaries between the classes which enable the model to predict with almost 99% accuracy. It is expected that real data from the physical environment contains more overlap between classes which makes it harder to predict correctly. Therefore, it is recommended to study novelty detection in the dataset. Novelty detection detects both standard samples as novelties in the dataset. Novelty detection results in a division of data samples in known classes and unknown classes which result in better and more realistic results as new classes are found.

Another interesting recommendation is to include a time component in the input dataset. A different model has to be developed, which takes this time component into account. This model can find any trends in time and predict future errors according to these trends. If these predicted errors are correct, countermeasures can be proposed to prevent these errors. It is interesting to see if any trends can be found because packaging lines are typically high-speed lines where events occur spontaneously. An example of an algorithm that can remember time series is a Recurrent neural network.

The research focussed mainly on the service model of a digital twin. The service model is a decision-making algorithm that can propose intelligent countermeasures to operators. The research only focussed on the development of one decision algorithm in the context of a multi-class classifier. A neural network is chosen as the multi-class classifier based on the different studies and understanding of the problem. It can be concluded that a neural network is a suitable algorithm for multi-class classification problems. However, it is founded that a neural network is also over-qualified in the context of this research. Neural networks can solve complex problems such as image recognition, text recognition, and video analyses. In the end, the provided dataset in this research is relatively easy compared to these complex problems. Therefore, it is interesting to study other machine learning algorithms which can solve this multi-class classification problem. The results of the different algorithms can be compared to choose the best suitable algorithm for this problem.

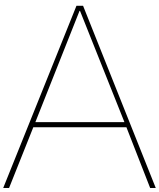
The final recommendation concerns an extension of the proposed neural network to multiple other machines of the packaging line. Different architectures to combine multiple neural networks are briefly discussed in Chapter 6. The output of these individual neural networks can be used in a general

decision algorithm to predict a sequence of countermeasures. In Chapter 6, these architectures are discussed from a theoretical point of view. It is interesting to combine the results of individual neural networks in one decision algorithm because this is useful information for implementing such a decision algorithm on a real packaging line.

7.2.2. Recommendations for Heineken

The first recommendation concerns the extension of sensors available in the database. If the training dataset is extended with new errors, more sensors are needed to distinguish outputs from each other. These sensors can vary from binary sensors to sensors that measure analogue values such as tensor sensors. Then, the training dataset can be extended with many more input variables. The problem becomes more complex but also offers opportunities to improve operational productivity further.

The second recommendation for Heineken is to start building a platform where operators can start labelling errors. Even if a decision algorithm is not yet in place, it is precious to start labelling errors because this research showed the importance of a proper training dataset. Especially labelling errors which do not occur often takes time. When a training dataset is available, it is relatively easy to develop this decision algorithm and implement the service for operators.



Research Paper

Design of a decision making algorithm to support operators in a real-time production environment

S.N.J. Koot, Dr. W.W.W. Beelaerts van Blokland and Dr. ir. D.L. Schott

***Abstract*—Production environments experience unplanned downtime occasions that have to be resolved by operators. An intelligent decision algorithm is developed to support operators in deciding which countermeasure will be successful. A neural network was chosen as this multi-class classifier. The model is applied to data from a production environment and simulated according to a real-time environment. The model is trained on two different machines with different datasets. The proposed system, including the decision algorithm, improves operational productivity in case of unplanned downtime by proposing correct countermeasures.**

Keywords: Multi-class classification problem, Real-time production environment, Digital twin, Decision algorithm

I. Introduction

Production environments are volatile environments and experiences downtime due to unexpected events. Every time the equipment in a brewery experiences downtime, it is not able to produce any goods. Downtime which occurs unexpectedly or as a result of a failure is called unplanned downtime (Immerman, 2018). Therefore, the reduction of unplanned downtime is one of the main goals in a production environment. The reduction of unplanned downtime can be split into two focus areas, the first one is to reduce the number of errors and the second one is to reduce the time to resolve an error. Different lean manufacturing methods such as zero-defect production (Wang, 2013) and Total Productive Maintenance (Pascal et al., 2019; Nakajima, 1984) describe the reduction of downtime in their methods. The focus in this research is on reducing the unplanned downtime from the moment the machine comes to a standstill.

It is necessary to take rapid decisions regarding resolving problems to reduce the unplanned downtime from a practical point of view (Miškuf and Zolotová, 2016). Taking rapid decisions can be established by dealing with real-time and historical data from the packaging line, combined with introducing intelligent decision algorithms. Such an algorithm decides which countermeasure an operator has to perform based on several parameters from the packaging line.

Both lean manufacturing and Industry 4.0 are promising production paradigms to solve future manufacturing problems. Mayr et al. (2018) did research about these two developments and how they support each other. From this research can be concluded that digitalization contributes to different lean manufacturing methods. Mainly, real-time computing in combination with a Digital Twin (DT) enables many different lean manufacturing methods to become more intelligent.

The technological basis of Industry 4.0 is the development of Internet of Things (IoT) (Ashton, 2009). The development of IoT resulted in many more sensors and devices connected to the internet, data acquisition systems, and computer networks. Managing these interconnected systems between physical assets and computational capabilities is called Cyber-Physical Systems (Lee et al., 2015). The controlling software part of Cyber-Physical System is called a DT. The physical devices of Cyber-Physical Systems communicate with each other with the use of a software replica of the physical devices.

Tao et al. (2019) presented a five-dimensional DT model to enable the use of a DT in more fields. Figure 1 shows the five-dimensional DT model. The five components of the model are (Qi et al., 2019):

- **Physical entities -** can be machines, products, devices, etc., which are the starting point of a DT.
- **Virtual models -** are the virtual replicas of the physical entity and consist of geometric models, physics models, etc.
- **Service models -** consist of optimization, simulation, diagnoses, etc.
- **DT data -** is obtained from the different components
- **Connections -** are crucial to communicate between the different components

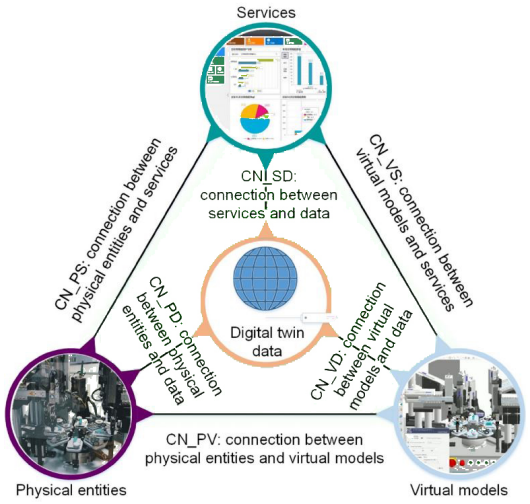


Figure 1. Five-dimensional DT model (Qi et al., 2019)

In this research, a digital service in a production environment based on real-time data is developed. This is typically provided in the service domain of a DT. Service plays an increasingly more critical role in manufacturing as manufacturing evolves toward socialization and servitization

(Lightfoot et al., 2013). The developed service is defined as a multi-class classification problem (Terry-Jack, 2019). Multi-class classification algorithms predict something into one of n classes. Multi-class classification algorithms are used in many real-world problems like speech recognition, face recognition, medical diagnosis, fraud detection, and fault detection (Bhardwaj et al., 2016). In the context of this research, the multi-class classification algorithm decides which countermeasure an operator has to execute based on specific real-time input parameters from the production line.

The remainder of this research is focussed on applying a multi-class classification algorithm on the packaging line of Alken-Maes brewery, which is one of the breweries of Heineken. From the packaging line of Alken-Maes, the model is first developed on the Filler. The Filler first rinses the cans quickly after which they are loaded on a circular star-wheel, which fills the cans volumetrically. Subsequently, the Seamer places a lid on the can and closes the seam of the lid. Finally, the Fill Height Inspector measures the filling height of the can and removes faulty cans from the line.

Section II discusses the development of the ML model according to a seven-step framework. The results of this section are used in Section III, which discusses the use case of this research. Then, the use case is expanded to another machine of the packaging line in Section IV. Finally, Section V presents the conclusion of the research and several recommendations for further research are presented.

II. Model development

The development of machine learning (ML) algorithms has been studied extensively over the last decades. These studies all use a seven-step framework for designing machine learning algorithms (Brownlee, 2013; Guo, 2017; Mayo, 2018; van Rijmenam, 2019). The seven steps are:

1) Data collection: The data collection consists of defining and obtaining data. From the Filler, the created dataset contains the following input parameters: the active program, the top three errors, the speed of the Filler, the inlet speed of the Filler, and the state of the Filler. The output classes are six predefined countermeasures to the errors.
2) Data preparation: The data is loaded into a suitable format and prepared for use in the ML algorithm. Data preparation is important because it will affect the result positively or negatively depending on the taken steps.
3) Choose a model: The third step consists of selecting the right model. The use of a NN as a multi-class classifier is chosen because of the natural suitability for this kind of problems. Moreover, a NN is effective for high dimensional data, and as the problem in real-life can expand to a high dimensionality problem, this algorithm is very suitable for this research.
4) Train the model: The goal of training the model is to incrementally improve the prediction of the model based on the training data. By iteratively comparing the predicted output with the corresponding output, the model can adjust its prediction. After training the model extensively, it can predict the right output corresponding to the input variables.
5) Evaluate the model: A test dataset is created containing new data which has not been seen by the model and is, therefore, used to determine the skill of the model on new data.
6) Hyperparameter Tuning: This step refers to hyperparameter tuning intending to improve the training of the model further. The tuning of the hyperparameters is done experientially and heavily depends on the specifics of the dataset, model and training process.
7) Prediction: Finally, when the model satisfies the needs, it is possible to predict based on the dataset and the model.

As required in the first step of the framework, a dataset is developed on which the model is trained. The dataset to train the model contains the following input variables:

- Program of the Filler
- Speed of the Filler
- Inlet speed of the Filler
- Error
  - Error 1: External 3527 - Can lack at inlet 1
  - Error 2: Operator Guard 0043 - Low speed inlet conveyor
  - Error 3: External 3549 - Outlet can too full

Furthermore, the output classes of the dataset consists of six countermeasures. The six countermeasures to resolve the errors are:

- Countermeasure 1: Unknown cause: investigate cause
- Countermeasure 2: Stop Filler
- Countermeasure 3: Solve a problem with Depalletiser
- Countermeasure 4: Solve can block at Filler
- Countermeasure 5: Slow down Filler
- Countermeasure 6: Solve can block at outlet Filler

Following the described seven steps resulted in a final model with accuracy, and logarithmic loss of 98,98%, 0,0368, respectively. The curves of both performance measures are shown in Figure 2. The optimal hyperparameters of the NN can be found in Table I.

Table I
Overview of optimal hyperparameters

Hyperparameters	Value or option
Nr. of training epochs	350
Batch size	256
Optimization algorithm	Nadam
Learning rate of optimization algorithm	0.0015
Activation function	Tanh
Dropout regularization	0
Nr. of hidden layers	2
Nr. of neurons in each layer	16

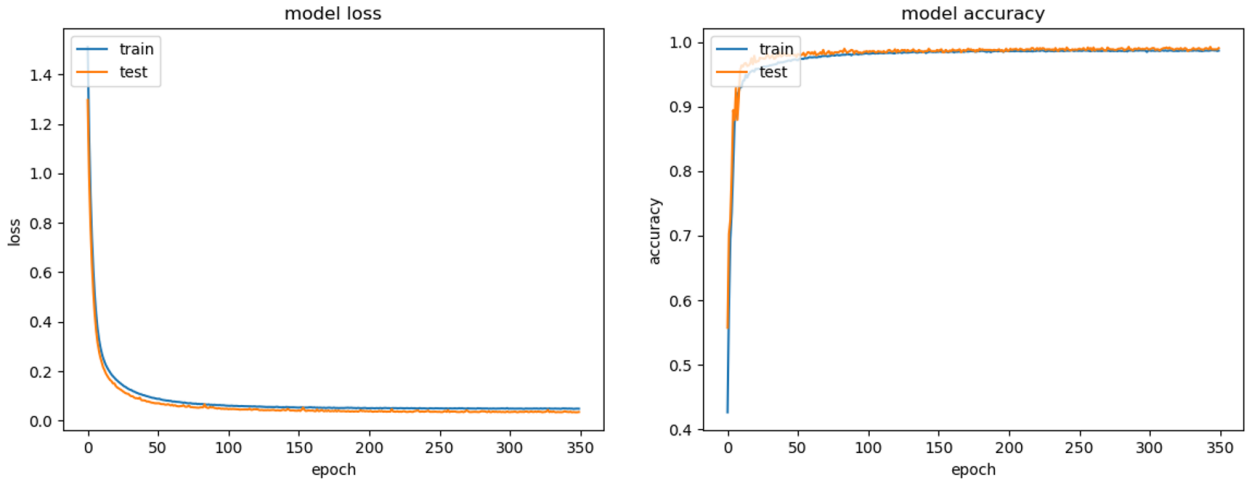


Figure 2. Logarithmic loss and classification accuracy for one fold of final model with two hidden layers and 16 neurons

III. Case study

For the case study, data points from the packaging between the 4th up until the 10th of May 2020 are used. First, the data is collected and prepared according to step 1 and 2 from the 7-step framework. Then, the defined and prepared dataset is fed into the model. The result of the model is a predicted output class for each sample of the dataset. It took 0,034 seconds to predict the entire dataset of 1071 samples, and it took only 0,012 seconds to predict one sample. The result of the prediction can be found in Table II, which shows the number of predicted outputs per class.

Table II
The number of predicted outputs per class

Output class	Number of samples
1	3
2	502
3	18
4	369
5	173
6	6

A. Calculation of time to resolve an error

The time to resolve an error in the current situation is based on an analysis of the data from multiple periods which resulted in a distributed time to resolve an error according to a normal distribution. From this analysis can be concluded that the average time to resolve an error is approximately 6 minutes. These 6 minutes are divided over different steps in the process of resolving an error. First, the operator responds typically in approximately 30 seconds from the moment he is notified about an error. Then, the operator walks to the PLC of the machine to determine which error causes the problem. Walking to the PLC takes on average 100 seconds. Then, the operator decides which countermeasure is executed to resolve the problem and walks to the location of the problem. This decision takes on average 60, and 30 seconds, respectively. Finally, the operator resolves the problems in 120 seconds. An overview of these steps can be found in Table III.

The times in Table III are assumed based on expert knowledge in combination with the information available about average times to resolve an error.

The time for the proposed model to decide which countermeasure the operator has to perform consists of multiple components. First, once every second, the database is updated with the newest data inputs from the packaging line. Then, the data is prepared, which takes on average 0,032 seconds. The prediction of output classes takes 0,012 seconds. Furthermore, it is assumed that the communication back to the brewery consumes the same amount of time (1 second) as the communication between the brewery and the database.

The total time to predict the correct countermeasure is approximately 2 seconds from the moment the error occurs. The operator receives a notification with the proposed countermeasure, walks towards the location of the problem, and resolves the error. However, it is assumed that an operator does not respond immediately to a notification (similar to the current situation). The time it takes for every step in the process to resolve an error is summarized in Table III.

Table III
Time for every step between the moment an error occurs until the error is resolved

Steps to resolve an error	Current situation [s]	Proposed situation [s]
Notification	30	30
Walk to PLC	100	-
Decide on counter measure	60	-
Walk to problem	30	100
Resolve problem	140	140
Total time	360	270

B. Results

As mentioned earlier, the total amount of errors in the considered time window is 1071. From these 1071 errors, 32 led to a minor stop of the Filler, which is considered as unplanned downtime. Therefore, the numbers of errors from Table II are multiplied by this ratio of minor stops per error.

The recall of the model is specified as the percentage of output classes that are recognized correctly. The recall of each class is considered in this simulation. For each class that is not predicted correctly, another 2 minutes are taken into account to solve the problem. An overview of the results of the simulation is given in Table IV.

The simulation resulted in an improvement of the proposed situation of 16 minutes compared to the current situation, which indicates a reduction of downtime of 22,2% under the assumptions made. This improvement of 22,2% is according to the expected difference. The difference between 6 and 4,5 minutes is 1,5 minutes which is an improvement of 25%. Considering the accuracy of the model, the expected difference is a little less than 25%. Compared to the real situation of the packaging line in Alken-Maes, this is plausible.

C. Discussion on results

The uncertainty of the assumed times to resolve an error is addressed by applying the same calculation on 5 and 7 minutes to resolve an error in the current situation. This results in a difference of 11 and 23 minutes between the current and proposed situation, respectively. This is a change of 31% and 44%, respectively, which indicates that the uncertainty in the assumptions affects the results of this simulation significantly. However, the actual downtime due to minor stops in that period is 84 minutes. This differs 12 minutes from the total simulated downtime (72 minutes), which is close enough to consider the initial results as realistic.

In the entire problem, only the three most occurring errors are considered. If more errors are considered, this will lead to a higher absolute difference. The same applies to expand this decision model to multiple other machines of the packaging line. Now, only the Filler is considered. However, other machines experience errors as well, and the application of this model on these machines can achieve comparable results. Furthermore, in the simulation, it is assumed that the operator makes the correct decision regarding the countermeasure because there is no information available about the ratio between taking the correct or wrong decision. This ratio will further increase the difference between the current and proposed situation in favour of the proposed situation.

The implementation of this system into the production environment of Alken-Maes is close to the suggested implementation of IoT systems together with AI techniques in the research of Patel et al. (2018). However, the implementation of the model in this research goes beyond the research of Patel et al. because the operator does not have to troubleshoot an occurring problem.

IV. Expansion of use case

The model is also applied to a second machine of the packaging line of Alken-Maes. The second machine on which the model is applied is the Shrink Packer. After the Filler, first the Tunnel Pasteurizer and then the Shrink Packer is located. The Shrink Packer shrinks a plastic wrap around a specified number of filled bottles.

A. Model training

A completely new dataset is generated which contains the four following input parameters:

- Program of the Shrink Packer
- State of the Shrink Packer
- Speed of the Shrink Packer
- Error
  - Error 1: ALL306 - Minimal accumulation layer
  - Error 2: ALL321 - Slow down outlet
  - Error 3: ALL233 - Deformed pack at uphill conveyor
  - Error 4: ALL273 - Minimal level reel film

The output classes of the dataset consists of six countermeasures. The six countermeasures to resolve the errors are:

- Countermeasure 1: Slow down Shrink Packer
- Countermeasure 2: Stop Shrink Packer
- Countermeasure 3: Remove deformed pack at uphill conveyor
- Countermeasure 4: Fill reel film
- Countermeasure 5: Reset sensor reel film
- Countermeasure 6: Fill accumulation layer

The dataset is created and contains many more different combinations compared to the dataset of the Filler. Where the dataset of the Filler contains only 13 combinations, the dataset of the Shrink Packer contains 57 different combinations. The time to train the model took more time due to the more extensive and complex dataset. Training the model took 70 seconds compared to 15 seconds of training the model for the Filler. However, after training the model, the logarithmic loss, and accuracy of the model are 0,0248, and 99,26%, respectively. The logarithmic loss and accuracy curves for one fold are shown in Figure 3. Both show a stable converging curve to the optimal performance.

B. Results

Data is retrieved from the Shrink Packer over the same period as for the Filler. Over the period from the 4th up until the 10th of May 2020, 121 errors occurred at the Shrink Packer, which resulted in 56 minor stops. The minor stops took on average 5 minutes which is a little less compared to the average time of a minor stop at the Filler. It is assumed that the time to solve an error in the proposed situation still takes 4,5 minutes. This results in a similar simulation than that of the Filler, which is summarized in Table V.

From Table V can be concluded that the absolute improvement for the Shrink Packer is smaller due to the smaller difference in time to resolve an individual error. The improvement is 4,8 minutes compared to the 16 minutes at

Table IV
OVERVIEW OF RESULTS OF SIMULATION OF PROBLEM SOLVING IN BOTH SITUATIONS

		Current situation	Proposed situation			
Class	Nr. of stops	Total time [min]	Recall	First attempt [min]	Second attempt [min]	Total time [min]
1	0	0	1,00	0	0	0
2	15	0	0,98	0	0,6	0,6
3	1	6	1,00	4,5	0	4,5
4	11	66	0,98	50	1	51
5	5	0	1,00	0	0	0
6	0	0	1,00	0	0	0
Total time [min]		72				56

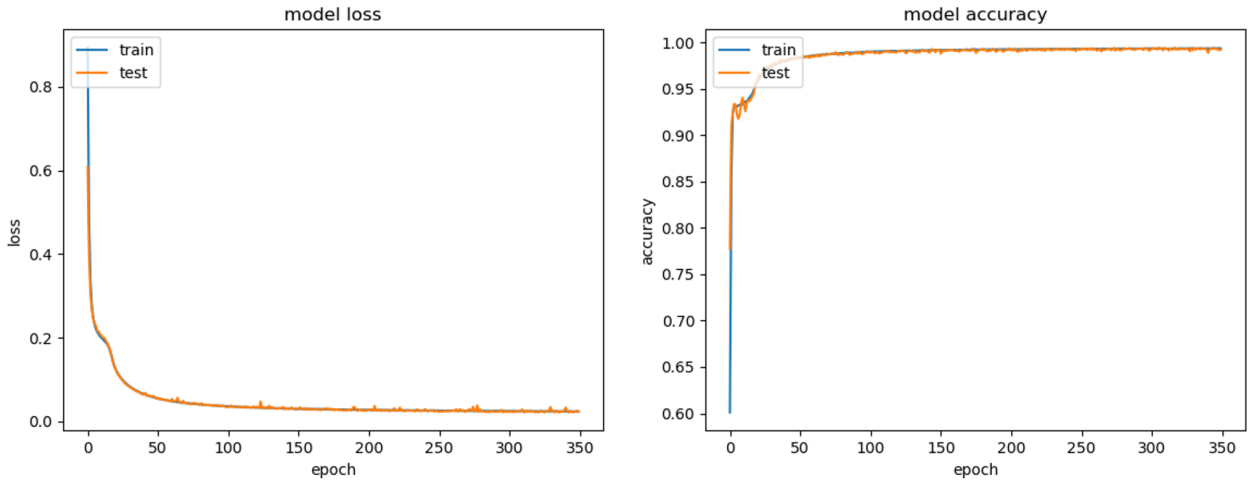


Figure 3. Logarithmic loss and classification accuracy for one fold of Shrink Packer model

Table V
OVERVIEW OF RESULTS OF SIMULATION OF SHRINK PACKER IN BOTH SITUATIONS

		Current situation	Proposed situation			
Class	Nr. of errors/stops	Total time [min]	Recall	First attempt [min]	Second attempt [min]	Total time [min]
1	245/36	0	0,99	0,0	0,0	0,0
2	44/6	0	1,00	0,0	0,0	0,0
3	20/3	15	0,96	14,6	0,2	14,8
4	29/4	21	0,98	19,0	0,2	19,2
5	0/0	0	1,00	0,0	0,0	0,0
6	46/7	34	1,00	30,2	0,0	30,2
Total time [s]		69				64,2

the Filler. Furthermore, it can be seen that countermeasure one is proposed most often, and countermeasure five is not proposed for errors. However, countermeasure one and two are both countermeasures which do not require any actions of the operator. Due to the combination of many errors solved by countermeasure one and that this countermeasure does not require any action of the operator also leads to a small improvement in the proposed situation.

C. Error propagation

Furthermore, errors can propagate through the packaging line and causes errors or minor stops at other machines. Error propagation occurs when the buffers cannot compensate for the starvation of blockage of the machine. From the minor stops at the Shrink Packer, two minor stops are caused by the Filler, five by the Tray Packer and one by the Palletiser. The minor stops of the Filler are mainly caused internally, and the Depalletizer causes only four minor stops, see Table VI.

From Table VI can be concluded that the error propagation between the Filler and the Shrink Packer is minimized to two minor stops of the Shrink Packer caused by the Filler. Production lines, as the packaging line of Alken-Maes, are designed to be balanced along the line to protect the bottleneck machine (slowest machine) from any minor stops caused by other machines (Patti et al., 2008). Typically, the Filler and the Pasteurizer are the slowest machine(s) (Elst et al., 2020).

Table VI
ERROR PROPAGATION BETWEEN MACHINES OF THE PACKAGING LINE

		Caused by				
		Depalletiser	Filler/Pasteurizer	Shrink Packer	Tray Packer	Palletiser
Affected	Depalletiser	0	0	0	0	0
	Filler/Pasteurizer	4	28	0	0	0
	Shrink Packer	0	2	54	5	1
	Tray Packer	0	0	2	86	2
	Palletiser	0	0	0	0	1

Table VI indicates that the line balancing of the packaging line of Alken-Maes is working correctly as only the Depalletiser caused four minor stops at the Filler/Pasteurizer.

The implementation of the proposed decision-making algorithm on each machine will lead to a further reduction of error propagation as the errors are solved faster. The buffers have a specific capacity, and when the capacity is reached, the error propagates to another machine. If the errors are solved faster, the capacity of the buffers is not or less exceeded, and the fewer errors propagate through the packaging line.

V. CONCLUSION

From this research can be concluded that only four of the five requirements of a digital twin are necessary for the implementation of an intelligent real-time algorithm. These four requirements are a physical asset, the access to real-time data of this asset, a service model, and the connections between these three components.

The operational productivity is improved by implementing a service that decides which countermeasure an operator has to execute in case of an occurring error. A neural network is chosen as decision algorithm. The performance of the neural network is heavily dependent on the compiled training dataset. Therefore, compiling a training dataset requires experts that can label input samples.

The neural network is applied to the Filler of the packaging line of Alken-Maes. Simulation of the proposed method shows the potential operational productivity improvement. The neural network predicts the correct countermeasure with 98,98% accuracy, which results in a weekly reduction of unplanned downtime of 16 minutes. This corresponds to 22,2% reduction of unplanned downtime for downtime occasions where the operator has to resolve the problem. Implementing the service on each machine of the packaging line and considering every error of the machines further reduces the unplanned downtime.

REFERENCES

Ashton, K. (2009). In the real world, things matter more than ideas. That 'Internet of Things' Thing. Technical report.

Bhardwaj, A., Tiwari, A., Bhardwaj, H., and Bhardwaj, A. (2016). A genetically optimized neural network model for multi-class classification | Elsevier Enhanced Reader. Technical report.

Brownlee, J. (2013). Applied machine learning process.

Elst, W. Z. V. D., Vleugel, J. M., Blokland, W. W. A. B. V., and Negenborn, P. R. R. (2020). Towards a continuous flow in production lines Case-study at Heineken Zoeterwoude.

Guo, Y. (2017). The 7 Steps of Machine Learning - Towards Data Science.

Immerman, G. (2018). The Real Cost of Downtime in Manufacturing.

Lee, J., Bagheri, B., and Kao, H.-A. (2015). A Cyber-Physical Systems architecture for Industry 4.0-based manufacturing systems. *Manufacturing Letters*, 3:18–23.

Lightfoot, H., Baines, T., and Smart, P. (2013). The servitization of manufacturing: A systematic literature review of interdependent trends. In *International Journal of Operations and Production Management*, volume 33, pages 1408–1434.

Mayo, M. (2018). Frameworks for Approaching the Machine Learning Process.

Mayr, A., Weigelt, M., Kühl, A., Grimm, S., Erll, A., Potzel, M., and Franke, J. (2018). Lean 4.0-A conceptual conjunction of lean management and Industry 4.0. In *Procedia CIRP*, volume 72, pages 622–628. Elsevier B.V.

Miškuf, M. and Zolotová, I. (2016). Comparison between multi-class classifiers and deep learning with focus on industry 4.0. In *2016 Cybernetics and Informatics, K and I 2016 - Proceedings of the 28th International Conference*. Institute of Electrical and Electronics Engineers Inc.

Nakajima, S. (1984). *An Introduction to Total Productive Maintenance (TPM)*.

Pascal, V., Toufik, A., Manuel, A., Florent, D., and Frédéric, K. (2019). Improvement indicators for Total Productive Maintenance policy. *Control Engineering Practice*, 82:86–96.

Patel, P., Ali, M. I., and Sheth, A. (2018). From raw data to smart manufacturing: AI and semantic web of things for industry 4.0. *IEEE Intelligent Systems*, 33(4):79–86.

Patti, A. L., Watson, K., and Blackstone, J. H. (2008). The shape of protective capacity in unbalanced production systems with unplanned machine downtime. *Production Planning and Control*, 19(5):486–494.

Qi, Q., Tao, F., Hu, T., Anwer, N., Liu, A., Wei, Y., Wang, L., and Nee, A. Y. (2019). Enabling technologies and tools for digital twin. *Journal of Manufacturing Systems*.

Tao, F., Liu, W., Zhang, M., Hu, T., Qi, Q., Zhang, H., Sui, F., Wang, T., Xu, H., Huang, Z., Ma, X., Zhang, L., Cheng, J., Yao, N., Yi, W., Zhu, K., Zhang, X., Meng, F., Jin, X., Liu, Z., He, L., Cheng, H., Zhou, E., Li, Y., Lyu, Q., and Luo, Y. (2019). Five-dimension digital twin model and its ten applications. *Jisuanji Jicheng Zhizao Xitong/Computer Integrated Manufacturing Systems, CIMS*, 25(1):1–18.

Terry-Jack, M. (2019). Tips and Tricks for Multi-Class

Classification.
van Rijmenam, M. (2019). 7 Steps to Machine Learning: How to Prepare for an Automated Future.
Wang, K.-S. (2013). Towards zero-defect manufacturing (ZDM)-a data mining approach.

B

Lay-out of Alken-Maes brewery

Figure B.1: Lay-out of Alken-Maes brewery

C

Manual verification

Actual	Predicted	Actual	Predicted	Actual	Predicted	Actual	Predicted	Actual	Predicted
5	5	5	5	3	3	5	5	2	2
2	2	4	4	1	1	2	2	2	2
4	4	6	6	4	4	5	5	**4**	**2**
5	5	3	3	1	1	5	5	3	3
3	3	3	3	**4**	**2**	5	5	2	2
3	3	2	2	3	3	2	2	2	2
3	3	2	2	6	6	2	2	5	5
4	4	3	3	2	2	2	2	2	2
3	3	5	5	4	4	2	2	2	2
3	3	3	3	1	1	1	1	1	1
3	3	4	4	6	6	3	3	5	5
2	2	2	2	2	2	1	1	2	2
2	2	2	2	4	4	2	2	5	5
1	1	3	3	2	2	4	4	3	3
2	2	3	3	2	2	3	3	3	3
2	2	2	2	4	4	6	6	2	2
2	2	2	2	2	2	5	5	4	4
3	3	3	3	4	4	3	3	1	1
2	2	2	2	4	4	2	2	6	6
1	1	2	2	3	3	6	6	2	2

Table C.1: Actual and predicted classes of first 100 entries test dataset

D

Use case verification

D.1. Use case verification with initial trainings dataset

Program	State Depalletizer	Speed Filler	Inlet Speed	Error	Predicted class	Required class
1	1	333	296	3	5	5
1	1	390	314	1	4	-
1	1	0	0	1	4	-
1	1	485	431	2	4	4
1	1	485	429	1	4	-
1	1	516	447	3	5	5
1	1	439	369	1	4	-
1	1	333	308	3	5	5
1	1	435	374	1	4	-
1	1	442	377	3	5	5
1	1	400	370	1	4	-
1	1	607	550	3	5	5
1	1	559	462	3	5	5
1	1	578	522	1	4	-
1	1	476	398	2	4	4
1	1	420	402	1	4	-
1	1	420	408	2	4	4
1	1	420	378	1	4	-
1	1	420	407	1	4	-
1	1	238	194	2	4	4

Table D.1: Manual verification of use case of first 20 samples

D.2. Use case verification after expansion trainings dataset

Program	State Depalletizer	Speed Filler	Inlet Speed	Error	Predicted class	Required class
1	1	333	296	3	5	5
1	1	390	314	1	2	2
1	1	0	0	1	2	2
1	1	485	431	2	4	4
1	1	485	429	1	2	2
1	1	516	447	3	5	5
1	1	439	369	1	2	2
1	1	333	308	3	5	5
1	1	435	374	1	2	2
1	1	442	377	3	5	5
1	1	400	370	1	2	2
1	1	607	550	3	5	5
1	1	559	462	3	5	5
1	1	578	522	1	2	2
1	1	476	398	2	4	4
1	1	420	402	1	2	2
1	1	420	408	2	4	4
1	1	420	378	1	2	2
1	1	420	407	1	2	2
1	1	238	194	2	4	4

Table D.2: Manual verification of use case of first 20 samples

E

Overview of dataset Shrink Packer

Shrink packer							
Program	1	1	1	1	1	1	1
State	5	7	7	6	6	6	6
Speed Packer [cans/min]	0-1400	1380-1400	0-1385	0-730	730-1400	725-1400	0-730
Error	1	1	1	1	1	2	2
Measure/Class	1	6	1	2	6	1	2
Shrink packer							
Program	7	7	7	7	7	7	7
State	5	7	7	6	6	6	6
Speed Packer [cans/min]	0-1380	1360-1380	0-1365	0-730	730-1400	725-1400	0-730
Error	1	1	1	1	1	2	2
Measure/Class	1	6	1	2	6	1	2
Shrink packer							
Program	10	10	10	10	10	10	10
State	5	7	7	6	6	6	6
Speed Packer [cans/min]	0-1000	950-970	0-955	0-370	370-1000	355-1000	0-360
Error	1	1	1	1	1	2	2
Measure/Class	1	6	1	2	6	1	2

Shrink packer							
Program	1	1	1	1	1	1	1
State	7	5	6	6	6	7	5
Speed Packer [cans/min]	0-1400	0-1400	720-1400	710-730	0-710	0-1400	0-1400
Error	2	2	3	3	3	3	3
Measure/Class	1	6	3	2	1	3	1
Shrink packer							
Program	7	7	7	7	7	7	7
State	7	5	6	6	6	7	5
Speed Packer [cans/min]	0-1360	0-1360	720-1380	710-730	0-710	0-1360	0-1360
Error	2	2	3	3	3	3	3
Measure/Class	1	6	3	2	1	3	1
Shrink packer							
Program	10	10	10	10	10	10	10
State	7	5	6	6	6	7	5
Speed Packer [cans/min]	0-1000	0-1000	360-1000	350-370	0-350	0-1000	0-1000
Error	2	2	3	3	3	3	3
Measure/Class	1	6	3	2	1	3	1

Shrink packer					
Program	1	1	1	1	1
State	6	5	5	7	7
Speed Packer [cans/min]	0-1400	1380-1400	0-1385	1380-1400	0-1380
Error	4	4	4	4	4
Measure/Class	4	5	4	5	4
Shrink packer					
Program	7	7	7	7	7
State	6	5	5	7	7
Speed Packer [cans/min]	0-1360	1360-1380	0-1365	1360-1380	0-1360
Error	4	4	4	4	4
Measure/Class	4	5	4	5	4
Shrink packer					
Program	10	10	10	10	10
State	6	5	5	7	7
Speed Packer [cans/min]	0-1000	950-970	0-955	950-970	0-1000
Error	4	4	4	4	4
Measure/Class	4	5	4	5	4

Table E.1: Overview of dataset Shrink Packer

Bibliography

Ahuja, I. P. and Khamba, J. S. (2008). Total productive maintenance: Literature review and directions. Technical Report 7.

Aly, M. (2005). Survey on Multiclass Classification Methods. Technical report.

Ashton, K. (2009). In the real world, things matter more than ideas. That 'Internet of Things' Thing. Technical report.

Auda, G. and Kamel, M. (1999). Modular neural networks: a survey. *International journal of neural systems*, 9(2):129–151.

Auda, G., Kamel, M., and Raafat, H. (1996). Modular neural network architectures for classification. In *IEEE International Conference on Neural Networks - Conference Proceedings*, volume 2, pages 1279–1284. IEEE.

Battini, D., Persona, A., and Regattieri, A. (2009). Buffer size design linked to reliability performance: A simulative study. *Computers and Industrial Engineering*, 56(4):1633–1641.

Bay, S. D. (1998). Combining Nearest Neighbor Classifiers Through Multiple Feature Subsets. Technical report.

Bessis, N. and Dobre, C. (2014). Studies in Computational Intelligence Series editor. Technical report.

Bhardwaj, A., Tiwari, A., Bhardwaj, H., and Bhardwaj, A. (2016). A genetically optimized neural network model for multi-class classification | Elsevier Enhanced Reader. Technical report.

Bishop, C. M. (1995). *Neural Networks for Pattern Recognition*.

Brownlee, J. (2013). Applied machine learning process.

Brownlee, J. (2016a). How To Improve Deep Learning Performance.

Brownlee, J. (2016b). Supervised and Unsupervised Machine Learning Algorithms.

Brownlee, J. (2017a). Gentle Introduction to the Adam Optimization Algorithm for Deep Learning.

Brownlee, J. (2017b). How to Evaluate the Skill of Deep Learning Models.

Brownlee, J. (2017c). How to Train a Final Machine Learning Model.

Brownlee, J. (2017d). What is the Difference Between a Parameter and a Hyperparameter?

Brownlee, J. (2019). A Tour of Machine Learning Algorithms.

Burges, C. J. (1998). A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167.

Carbonell, J. G., Michalski, R. S., and Mitchell, T. M. (1983). AN OVERVIEW OF MACHINE LEARNING. In *Machine Learning*, pages 3–23. Elsevier.

Carrasco, O. C. (2017). Support Vector Machines for Classification -.

Chen, K. (2015). Deep and Modular Neural Networks. In *Springer Handbook of Computational Intelligence*, pages 473–494. Springer Berlin Heidelberg.

Choromanska, A., Henaff, M., Mathieu, M., Arous, G. B., and LeCun, Y. (2015). The loss surfaces of multilayer networks. In *Journal of Machine Learning Research*, volume 38, pages 192–204. Microtome Publishing.

Chris Tseng, H. and Almogahed, B. (2009). Modular neural networks with applications to pattern profiling problems. *Neurocomputing*, 72(10-12):2093–2100.

Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine Learning*, 20(3):273–297.

Delft University of Technology (2019). Track: Multi-Machine Engineering.

Deotte, C. (2018). How to score 97%, 98%, 99%, and 100%.

Department of Defense (2001). SYSTEMS ENGINEERING FUNDAMENTALS. Technical report, Defense Acquisition University Press.

Ding, C. H. Q. and Dubchak, I. (2001). Multi-class protein fold recognition using support vector machines and neural networks. Technical Report 4.

Donges, N. (2019). 4 Disadvantages Of Neural Networks.

Elm, W. C., Gualtieri, J. W., Mckenna, B. P., Tittle, J. S., Peffer, J. E., Szymczak, S. S., and Grossman, J. B. (2008). Integrating Cognitive Systems Engineering Throughout the Systems Engineering Process. *Journal of Cognitive Engineering and Decision Making*, 2(3):249–273.

Elst, W. Z. V. D., Vleugel, J. M., Blokland, W. W. A. B. V., and Negenborn, P. R. R. (2020). Towards a continuous flow in production lines Case-study at Heineken Zoeterwoude.

Er, O., Tanrikulu, A. C., Abakay, A., and Temurtas, F. (2012). An approach based on probabilistic neural network for diagnosis of Mesothelioma's disease. *Computers and Electrical Engineering*, 38(1):75–81.

Esposito, C., Castiglione, A., Martini, B., and Choo, K. K. R. (2016). Cloud Manufacturing: Security, Privacy, and Forensic Concerns. *IEEE Cloud Computing*, 3(4):16–22.

Farid, D. M., Zhang, L., Rahman, C. M., Hossain, M. A., and Strachan, R. (2014). Hybrid decision tree and naïve Bayes classifiers for multi-class classification tasks. *Expert Systems with Applications*, 41(4 PART 2):1937–1946.

Ferreira, F., Faria, J., Azevedo, A., and Marques, A. L. (2017). Product lifecycle management in knowledge intensive collaborative environments: An application to automotive industry. *International Journal of Information Management*, 37(1):1474–1487.

Foote, K. D. (2016). A Brief History of the Internet of Things.

Forsberg, K. and Mooz, H. (1994). The Relationship of System Engineering to the Project Cycle. Technical report.

Garbade, M. J. (2018). Clearing the Confusion: AI vs Machine Learning vs Deep Learning Differences.

Ginder, A., Robinson, A., and Robinson, C. J. (1995). *Implementing TPM: The North American Experience*.

Goienetxea Uriartea, A., Ng, A. H., and Urenda Moris, M. (2018). Supporting the lean journey with simulation and optimization in the context of Industry 4.0. Technical report.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning (Adaptive Computation and Machine Learning series)*, volume 521.

Goodhue, D. L. and Thompson, R. L. (1995). Task-technology fit and individual performance. *MIS Quarterly: Management Information Systems*, 19(2):213–233.

Gordon, A. D., Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). Classification and Regression Trees. *Biometrics*, 40(3):874.

Grezmak, J., Zhang, J., Wang, P., and Gao, R. X. (2020). Multi-stream convolutional neural network-based fault diagnosis for variable frequency drives in sustainable manufacturing system. Technical report.

Grieves, M. (2016). Origins of the Digital Twin Concept. *Revista de obstetricia y ginecolog??a de Venezuela*, 23(August):889–896.

Gubbi, J., Buyya, R., Marusic, S., and Palaniswami, M. (2013). Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7):1645–1660.

Guo, Y. (2017). The 7 Steps of Machine Learning - Towards Data Science.

Gupta, J. (2020). Going beyond 99% — MNIST Handwritten Digits Recognition.

Hao, P. Y., Chiang, J. H., and Lin, Y. H. (2009). A new maximal-margin spherical-structured multi-class support vector machine. *Applied Intelligence*, 30(2):98–111.

Haykin, S., York, N., San, B., London, F., Sydney, T., Singapore, T., Mexico, M., Munich, C., Cape, P., Hong, T., and Montreal, K. (2009). *Neural Networks and Learning Machines Third Edition*.

Heineken (2018). HEINEKEN opens new brewery in Meoqui, Mexico.

Heineken (2019a). HEINEKEN opens first brewery in Mozambique, a US$100 million investment.

Heineken (2019b). Who we are | The HEINEKEN Company.

Heineken (2020a). Heineken Global Projects & Engineering in Operational Set-Up | Internal document. Technical report.

Heineken (2020b). Internal document. Technical report.

Heineken (2020c). Our heritage | The HEINEKEN Company.

Immerman, G. (2018). The Real Cost of Downtime in Manufacturing.

Juszczak, P. and Duin, R. P. W. (2004). Combining One-Class Classifiers to Classify Missing Data. pages 92–101. Springer, Berlin, Heidelberg.

Kammer, P. J., Bolcer, G. A., Taylor, R. N., Hitomi, A. S., and Bergman, M. (2000). Techniques for supporting dynamic and adaptive workflow. *Computer Supported Cooperative Work: CSCW: An International Journal*, 9(3):269–292.

Kang, S., Cho, S., and Kang, P. (2015). Multi-class classification via heterogeneous ensemble of one-class classifiers. *Engineering Applications of Artificial Intelligence*, 43:35–43.

Kateris, D., Moshou, D., Pantazi, X. E., Gravalos, I., Sawalhi, N., and Loutridis, S. (2014). A machine learning approach for the condition monitoring of rotating machinery. *Journal of Mechanical Science and Technology*, 28(1):61–71.

Kis, T. and Pesch, E. (2005). A review of exact solution methods for the non-preemptive multiprocessor flowshop problem. In *European Journal of Operational Research*, volume 164, pages 592–608. Elsevier.

Köksal, G., Batmaz, I., and Testik, M. C. (2011). A review of data mining applications for quality improvement in manufacturing industry.

Kritzinger, W., Karner, M., Traar, G., Henjes, J., and Sihn, W. (2018). Digital Twin in manufacturing: A categorical literature review and classification. *IFAC-PapersOnLine*, 51(11):1016–1022.

Lee, D. and Lee, J. (2005). Domain described support vector classifier for multi-classification problems.

Lee, J., Bagheri, B., and Kao, H.-A. (2015). A Cyber-Physical Systems architecture for Industry 4.0-based manufacturing systems. *Manufacturing Letters*, 3:18–23.

Lightfoot, H., Baines, T., and Smart, P. (2013). The servitization of manufacturing: A systematic literature review of interdependent trends. In *International Journal of Operations and Production Management*, volume 33, pages 1408–1434.

Lippmann, R. P. (1987). An Introduction to Computing with Neural Nets. *IEEE ASSP Magazine*, 4(2):4–22.

Lorena, A. C., De Carvalho, A. C., and Gama, J. M. (2008). A review on the combination of binary classifiers in multiclass problems. *Artificial Intelligence Review*, 30(1-4):19–37.

Lughofer, E. (2012). Hybrid active learning for reducing the annotation effort of operators in classification systems. *Pattern Recognition*, 45(2):884–896.

Macchi, M., Roda, I., Negri, E., and Fumagalli, L. (2018). Exploring the role of Digital Twin for Asset Lifecycle Management. *IFAC-PapersOnLine*, 51(11):790–795.

Manogna, S. and Dakannagari, H. R. (2016). Internet of Things. Technical report.

Mayo, M. (2018). Frameworks for Approaching the Machine Learning Process.

Mayoraz, E. and Alpaydm, E. (1999). Support vector machines for multi-class classification. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 1607, pages 833–842. Springer Verlag.

Mayr, A., Weigelt, M., Kühl, A., Grimm, S., Erll, A., Potzel, M., and Franke, J. (2018). Lean 4.0-A conceptual conjunction of lean management and Industry 4.0. In *Procedia CIRP*, volume 72, pages 622–628. Elsevier B.V.

Min, H. (2010). Artificial intelligence in supply chain management: theory and applications. *International Journal of Logistics: Research and Applications*, 13(1):13–39.

Min, Q., Lu, Y., Liu, Z., Su, C., and Wang, B. (2019). Machine Learning based Digital Twin Framework for Production Optimization in Petrochemical Industry. *International Journal of Information Management*, 49:502–519.

Mishra, A. (2018). Metrics to Evaluate your Machine Learning Algorithm.

Miškuf, M. and Zolotová, I. (2016). Comparison between multi-class classifiers and deep learning with focus on industry 4.0. In *2016 Cybernetics and Informatics, K and I 2016 - Proceedings of the 28th International Conference*. Institute of Electrical and Electronics Engineers Inc.

MissingLink (2019). Classification with Neural Networks: Is it the Right Choice?

Moussa, M. A. (2004). Combining expert neural networks using reinforcement feedback for learning primitive grasping behavior. *IEEE Transactions on Neural Networks*, 15(3):629–638.

Nain, A. (2017). TensorFlow or Keras? Which one should I learn?

Nakajima, S. (1984). *An Introduction to Total Productive Maintenance (TPM)*.

Narendra, N. C. (2004). Flexible support and management of adaptive workflow processes. *Information Systems Frontiers*, 6(3):247–262.

Navlani, A. (2018). KNN Classification using Scikit-learn.

Newell, A. (1990). Unified Theories of Cognition.

Nikolaidou, M., Anagnostopoulos, D., and Tsalgatidou, A. (2008). Business processes modelling and automation in the banking sector: A case study. Technical report.

Nwankpa, C., Ijomah, W., Gachagan, A., and Marshall, S. (2018). Activation Functions: Comparison of trends in Practice and Research for Deep Learning.

Ou, G. and Murphey, Y. L. (2007). Multi-class pattern classification using neural networks. *Pattern Recognition*, 40(1):4–18.

Ouelhadj, D. and Petrovic, S. (2009). A survey of dynamic scheduling in manufacturing systems.

Paolanti, M., Romeo, L., Felicetti, A., Mancini, A., Frontoni, E., and Loncarski, J. (2018). Machine Learning approach for Predictive Maintenance in Industry 4.0. In *2018 14th IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications, MESA 2018*. Institute of Electrical and Electronics Engineers Inc.

Pascal, V., Toufik, A., Manuel, A., Florent, D., and Frédéric, K. (2019). Improvement indicators for Total Productive Maintenance policy. *Control Engineering Practice*, 82:86–96.

Patel, P., Ali, M. I., and Sheth, A. (2018). From raw data to smart manufacturing: AI and semantic web of things for industry 4.0. *IEEE Intelligent Systems*, 33(4):79–86.

Patti, A. L., Watson, K., and Blackstone, J. H. (2008). The shape of protective capacity in unbalanced production systems with unplanned machine downtime. *Production Planning and Control*, 19(5):486–494.

Price, D., Knerr, S., Personnaz, L., and Dreyfus, G. (1995). Pairwise Neural Network Classifiers with Probabilistic Outputs. Technical report.

Qi, Q., Tao, F., Hu, T., Anwer, N., Liu, A., Wei, Y., Wang, L., and Nee, A. Y. (2019). Enabling technologies and tools for digital twin. *Journal of Manufacturing Systems*.

Qi, Q., Tao, F., Zuo, Y., and Zhao, D. (2018). Digital Twin Service towards Smart Manufacturing. *Procedia CIRP*, 72:237–242.

Qian, B., Su, J., and Wen, Z. (2019). Orchestrating Development Lifecycle of Machine Learning Based IoT Applications: A Survey. Technical report.

Quatrini, E., Constantino, F., Di Gravio, G., and Patriarca, R. (2020). Machine learning for anomaly detection and process phase classification to improve safety and maintenance activities | Elsevier Enhanced Reader. Technical report.

Radecic, D. (2018). Softmax Activation Function Explained.

Reed, R. D. and Marks, R. J. (1999). *Neural smithing : supervised learning in feedforward artificial neural networks*. MIT Press.

Roblek, V., Meško, M., and Krapež, A. (2016). A Complex View of Industry 4.0. *SAGE Open*, 6(2).

Rokach, L. (2010). Ensemble-based classifiers. *Artificial Intelligence Review*, 33(1-2):1–39.

Rossit, D. A., Tohmé, F., and Frutos, M. (2019). A data-driven scheduling approach to smart manufacturing. *Journal of Industrial Information Integration*, 15:69–79.

Salzberg, S. L. (1993). C4.5: Programs for Machine Learning. *Machine Learning*, 16(3):235–240.

Sauser, B., Gove, R., Forbes, E., and Ramirez-Marquez, J. E. (2010). Integration maturity metrics: Development of an integration readiness level. *Information Knowledge Systems Management*, 9(1):17–46.

Scime, L. and Beuth, J. (2018). A multi-scale convolutional neural network for autonomous anomaly detection and classification in a laser powder bed fusion additive manufacturing process | Elsevier Enhanced Reader. Technical report.

Shafto, M., Conroy, M., Doyle, R., Glaessgen, E., Kemp, C., Lemoigne, J., and Wang, L. (2010). DRAFT Modeling, Simulation, Information Technology & Processing Roadmap. Technical report.

Sharma, S. (2017). Activation Functions in Neural Networks.

Shmueli, B. (2019). Multi-Class Metrics Made Simple, Part I: Precision and Recall.

Sidel (2019). Medium speed mechanical filler.

Sidel (2020). Beer bottling line.

Simon, H. A. and Mintzberg, H. (1977). The New Science of Management Decision, Revised Edition. *Administrative Science Quarterly*, 22(2):342.

Singh, A. K., Tiwari, S., and Shukla, V. P. (2012). Wavelet based Multi Class image classification using Neural Network. Technical Report 4.

Sivaram, N. M., Devadasan, S. R., and Murugesh, R. (2013). Conceptualisation for implementing total productive maintenance through the iso 9001:2008 standard-based quality management system. *South African Journal of Industrial Engineering*, 24(2):33–46.

Soni, D. (2018). Supervised vs. Unsupervised Learning - Towards Data Science.

Stewart, M. (2019). Neural Network Optimization.

Sundmaeker, H., Guillemin, P., Friess, P., and Woelfflé, S. (2010). Vision and Challenges for Realising the Internet of Things The meaning of things lies not in the things themselves, but in our attitude towards them. Antoine de Saint-Exupéry. Technical report.

Tao, F., Cheng, J., Qi, Q., Zhang, M., Zhang, H., and Sui, F. (2018). Digital twin-driven product design, manufacturing and service with big data. *International Journal of Advanced Manufacturing Technology*, 94(9-12):3563–3576.

Tao, F., Liu, W., Zhang, M., Hu, T., Qi, Q., Zhang, H., Sui, F., Wang, T., Xu, H., Huang, Z., Ma, X., Zhang, L., Cheng, J., Yao, N., Yi, W., Zhu, K., Zhang, X., Meng, F., Jin, X., Liu, Z., He, L., Cheng, H., Zhou, E., Li, Y., Lyu, Q., and Luo, Y. (2019a). Five-dimension digital twin model and its ten applications. *Jisuanji Jicheng Zhizao Xitong/Computer Integrated Manufacturing Systems, CIMS*, 25(1):1–18.

Tao, F. and Qi, Q. (2019a). New IT driven service-oriented smart manufacturing: Framework and characteristics. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 49(1):81–91.

Tao, F. and Qi, Q. (2019b). New IT driven service-oriented smart manufacturing: Framework and characteristics. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 49(1):81–91.

Tao, F. and Zhang, M. (2017). Digital Twin Shop-Floor: A New Shop-Floor Paradigm Towards Smart Manufacturing. *IEEE Access*, 5:20418–20427.

Tao, F., Zhang, M., and Nee, A. (2019b). Digital Twin and Services. In *Digital Twin Driven Smart Manufacturing*, pages 203–217. Elsevier.

Tax, D. and Duin, R. (2008). Growing a multi-class classifier with a reject option. *Pattern Recognition Letters*.

Terry-Jack, M. (2019). Tips and Tricks for Multi-Class Classification.

Thames, L. and Schaefer, D. (2016). Software-defined Cloud Manufacturing for Industry 4.0. *Procedia CIRP*, 52:12–17.

Trkman, P. (2010). The critical success factors of business process management. *International Journal of Information Management*, 30(2):125–134.

Van der Aalst, W. M. and Kumar, A. (2003). XML-based schema definition for support of interorganizational workflow. *Information Systems Research*, 14(1):23–46.

Van Der Maaten, L. and Hinton, G. (2008). Visualizing Data using t-SNE. Technical Report Nov.

van Ede, J. (2008). Hoe te kiezen tussen Lean, Six Sigma en TPM?

van Rijmenam, M. (2019). 7 Steps to Machine Learning: How to Prepare for an Automated Future.

Wang, K.-S. (2013). Towards zero-defect manufacturing (ZDM)-a data mining approach.

Wang, M. and Wang, H. (2005). From process logic to business logic-A cognitive approach to business process management.

Wasilewska, A. (2018). Modular Neural Networks.

Wen, J., Li, S., Lin, Z., Hu, Y., and Huang, C. (2012). Systematic literature review of machine learning based software development effort estimation models.

Williams, J. (2018). Deep Learning in Digital Pathology.

Zheng, P. and Sivabalan, A. S. (2020). A generic tri-model-based approach for product-level digital twin development in a smart manufacturing environment. *Robotics and Computer-Integrated Manufacturing*, 64.