JULY 2019 | BSC THESIS *CREATIVE TECHNOLOGY*

JESPER PROVOOST

SHORT-TERM PREDICTION AND VISUALIZATION OF PARKING AREA STATES IN REAL-TIME

A MACHINE LEARNING APPROACH

dat. mobility

UNIVERSITY OF TWENTE.

SHORT-TERM PREDICTION AND VISUALIZATION OF PARKING AREA STATES IN REAL-TIME: A MACHINE LEARNING APPROACH

A thesis submitted to the University of Twente in partial fulfillment of the requirements for the degree of

Bachelor of Science in Creative Technology

Supervisors
University of Twente
dr. ir. M. van Keulen
dr. A. Kamilaris

DAT.mobility
ir. S.J. van der Drift
dr. ir. L.J.J. Wismans

by

Jesper C. Provoost
s1789198

July 2019

Abstract

Public road authorities and private mobility service providers need information about the future traffic states to act pro-actively upon the spatial and temporal dynamics of the urban road network. In this research, a machine learning methodology for predicting influx, outflux and occupancy rate of parking areas on a horizon of up to 60 minutes has been developed using publicly available historic and real-time data sources. Based on a thorough development, optimization and selection process applied to a real-world case in the city of Arnhem, the feed-forward neural network turns out to outperform the random forest on all assessed performance measures, even though the differences are small and both are outperforming a naive (seasonal random walk) model. Although the performance degrades with increasing prediction horizon, the model shows an overall performance gain of 235% (considering all horizons up to 60 minutes ahead) in comparison with the naive model. Furthermore, it is shown that predicting the in- and outflux is a far more difficult task which needs more training data than occupancy rate. At the same time, however, their respective performance is still $33\frac{1}{3}$% and 25% better than a naive model and is less sensitive for the prediction horizon. In addition, the research demonstrates that real-time information of current occupancy rate is the independent variable with the highest contribution to the performance, although time, traffic flow and weather variables also deliver a significant contribution. Also, it is shown that relatively little training data is needed to maintain satisfactory predictive performance. This is a promising finding regarding the ease of implementing other parking areas into the system, especially in cases where the availability of data is substandard. During real-time deployment, the model shows to perform 172% better than the naive model. As a result, it can provide valuable information for pro-active traffic management as well as mobility service providers.

Acknowledgements

I would like to express my sincere gratitude to my supervisor Maurice van Keulen for his enthusiasm and knowledge which he has shown during the execution of my research and writing processes. His open and supportive attitude has stimulated me to tackle new challenges within this research and to strive for academic excellence.

Secondly, I would like to thank Andreas Kamilaris for giving me all the needed support during the initial formation of the assignment as well as execution of the research. His encouragement, patience and insightful feedback were incredibly important during the process of establishing this thesis.

A special mention goes to Sander van der Drift, my main supervisor at DAT.mobility. By providing fruitful feedback and inspiration, Sander has been a large influence on this thesis. His openness, helpfulness and dedication were an essential part of the pleasant working environment which I experienced at DAT.mobility. Moreover, Sander's personal approach helped me to quickly feel at home within the organization. I could not have imagined having a better mentor during this process.

Last, but certainly not least, this result would not have been possible without the support and commitment of Luc Wismans. In the initial stages, his efforts have paved the way for a fantastic period at DAT.mobility. I am grateful for the trust and autonomy which were given to me during these phases, since it allowed me to fully engage in the forming process of an assignment which suited myself and the company optimally. During the execution phase, Luc knew exactly how to further motivate me with insightful and in-depth feedback. I thoroughly enjoyed our teamwork and (sometimes intense) discussions, which have undoubtedly contributed to the determination and pursuance of quality within my research.

CONTENTS

1 INTRODUCTION

The main objective of this thesis is to develop a method for predicting and visualizing the influx, outflux and occupancy rate of parking garages in real-time in order to enhance existing short-term prediction methods regarding traffic conditions. Such a method could contribute to the effectiveness and efficiency of traffic management processes, aiming to increase societal benefits by alleviating contemporary parking and traffic problems. In addition, it could provide authorities and policy-makers with fruitful insights about overall influence of parking on traffic networks.

This chapter will provide a brief introduction on contemporary parking problems and their effect. Then, after identifying the limitations of existing research and technologies, the objectives and challenges will be discussed. Subsequently, the research questions are defined, followed by an outline of the contents of this thesis.

1.1 MOTIVATION

1.1.1 Problem statement

Finding an available parking space is often a difficult task, especially in dense urban areas. This is understandable, considering that the number of passenger cars in The Netherlands alone has grown 32% since 2000 [1]. Due to ongoing population growth, vibrant economies and urbanization, it is unlikely that this problem will solve itself soon. Over the last decades, authorities have implemented public off-street parking, such as garages and lots, as an effective measure in order to keep up with the demand. Off-street facilities can generally offer higher capacity while inducing lower stress on surrounding traffic flows, as opposed to traditional on-street parking. However, these facilities are usually distributed sparsely across a city and therefore require drivers to search more proactively for a suitable parking location [2]. Especially when a driver is unfamiliar in the area, or when traffic is heavy, this process wastes time and fuel while inducing additional traffic load on the surrounding road network [3].

Searching for a vacant parking space thus imposes a significant burden on drivers and the wider economy, as valuable resources are wasted in the process. According to research by *INRIX* [4], a leading

(a) Convential road-side system **(b)** State-of-the-art mobile system

Figure 1.1: Implementations of PGI systems

provider of traffic and navigation services, U.S. drivers spend an average of 17 hours searching for a parking spot every year. This amount is even higher in the U.K. and Germany with 44 and 41 hours per year, respectively. In Germany alone, *INRIX* estimates that the average driver wastes €896 per year on the hunt for a parking space. This aggregates to a yearly burden of €40.4 billion on the German economy. Furthermore, a survey of 17,968 drivers from 30 cities shows that 64% of participants experience stress while trying to find parking [4]. Altogether, it is obvious that this problem has a large impact on economy, society and quality of life.

1.1.2 Status quo

Traffic management applies measures to adjust the demand and capacity of the traffic network in time and space, such that ideal traffic demands and supplies are satisfied. To battle contemporary parking problems, as well as traffic problems altogether, traffic management is a highly relevant instrument [5]. The advance of modern technologies, particularly in the form of *intelligent transportation systems* (ITS), has supported authorities to execute their traffic management tasks more effectively and efficiently [6]. Within this context, many applications of ITS are targeted at extrinsically managing traffic by controlling infrastructure and access thereof, e.g. using lane management and signal control.

However, ITS is also used as a means to directly inform or influence road users such that they make 'smarter' use of traffic networks [6]. With regard to parking, a relevant example is the *parking guidance and information* (PGI) system which supplies drivers with dynamic parking information within controlled areas. There are multiple variations of PGI systems, each with their own respective method of communicating and presenting information to drivers [7]. A conventional implementation is a static sign which displays the current number of available parking spaces (as illustrated in Figure 1.1a), but recent developments have also led to the integration of dynamic parking

information in mobile apps and in-car infotainment systems (as illustrated in Figure 1.1b) [5]. Overall, the relevance of ITS regarding (parking-induced) traffic problems is two-fold:

1. Providing dynamic parking information to authorities and service providers
   - Generates better understanding of parking phenomena and therefore the overall traffic situation
   - Facilitates the extrinsic implementation of dynamic traffic management measures
2. Refining and communicating this information directly to drivers
   - Using state-of-the-art technology, such as integration in navigation apps and in-car infotainment systems
   - Enables operators to exert direct influence (i.e. via service provider) within the vehicle

What these ITS applications have in common, is their dependence on adequate and high-quality information. Producing and supplying this information is a challenging task which requires knowledge, dedication and expertise.

1.1.3 Introduction to DAT.mobility

DAT.mobility, part of the Goudappel Groep, is a Deventer-based company with expertise in IT solutions and data analysis in the field of mobility. Its main customers are consultants, planners, policy makers, transport operators and construction firms. Using extensive knowledge about mobility, IT and traffic modelling, DAT.mobility is able to generate the correct information to ensure that the optimal decisions are made. Notably, the company has a long track record of developing solutions for traffic prediction on the short and medium term, aiming to provide stakeholders with more insights into traffic

Figure 1.2: Example of an existing short-term traffic prediction tool

situations. For instance, predictions are visualized in an online environment and communicated to traffic management centers, in which they can be beneficial for ITS applications and decision-making processes. An example use case of such an application is shown in Figure 1.2. Overall, the mission of DAT.mobility is to make decisions effective, reliable and insightful. This is done by combining expertise and societal value with user-friendly solutions and advises.

1.1.4 Current limitations

Traffic management needs accurate and complete information on traffic conditions, especially when non-regular traffic conditions occur [8]. Until now, authorities have mostly depended on real-time or historic data for these purposes. However, due to the highly dynamic nature of traffic, current information could already become obsolete within a matter of minutes. This, combined with prevailing latency in data availability, limits the effectiveness of contemporary traffic management measures.

Wismans et al. conclude that stakeholders, public road authorities and private mobility service providers need information on and derived from the current and predicted traffic states "to act upon the daily urban system and its spatial and temporal dynamics" [8, p. 2]. A similar stance is taken by Vlahogianni et al. [9], who state that accurate parking predictions may lead to better management of the system by transport operators and thereby congestion mitigation due to avoidance of queue formation. Moreover, predictions could be used as instrument to timely inform drivers, such that the effectiveness of their decisions is maximized upon arrival at their destination.

As mentioned in Section 1.1.3, DAT.mobility is developing short-term prediction tools which aim to provide stakeholders with such insights. A prevailing drawback, however, is the absence of parking as input for their underlying predictive models, especially given the high influence of parking areas on the surrounding road network. Given the fact that 40% of traffic in urban areas is attributed to the search for a parking space [10], it is obvious that parking state predictions will add substantial value to the existing tools and therefore provide new opportunities for pro-active traffic management.

1.2 OBJECTIVES

Section 1.1 affirms the need for a system which can reliably predict the future state of off-street parking areas in real-time and communicate this to stakeholders. Such a system would further enhance the existing short-term traffic prediction tools of DAT.mobility. This, in turn, would empower pro-active traffic management processes, such

that traffic flows can be anticipated and regulated in pursuance of reducing congestion, time waste, stress and fuel exhaustion [9].

In order to provide a useful input for existing traffic models, the *in- and outflux* (i.e. the number of cars which enter and leave within a specified time unit) of the parking area are the most important variables to predict, considering that they best describe the induced loads on the surrounding traffic network. The occupancy rate (%) of the parking area, which is directly related to the in- and outflux, is of secondary importance. It could mainly be helpful for operators and service providers to directly inform drivers, e.g. in the form of a PGI system (see Figure 1.1b) which would facilitate routes towards parking areas with (expected) vacant spaces while possibly diverting traffic from highly occupied parking areas.

The aim of this thesis is therefore to:

- Acquire and select input data features based on an extensive assessment of their predictive power
- Determine the most suitable machine learning method among multiple candidates to predict influx, outflux and occupancy rate
- Compile, train and validate a machine learning model which can accurately predict influx, outflux and occupancy rate based on the defined input features
- Interactively visualize the real-time predictions in order to provide useful insights for stakeholders
- Develop a system architecture which can execute the prediction and visualization processes continuously and autonomously using real-time data feeds

1.3 CHALLENGES

Before all objectives can be satisfied, there will be some hurdles to overcome. First and foremost, it will be challenging to find a suitable approach for handling time series data within the machine learning domain. Errors and uncertainty will naturally grow when the predictive time horizon becomes larger. It will be demanding to develop a model which does not only predict the upcoming five minutes reliably, but also the next 60 minutes. It is therefore crucial to minimize further propagation of errors within the model itself.

Another challenge is missing and erroneous data, either in the training set or in the real-time data feed. For instance, when one of the data feeds is malfunctioning, the system should be able to remain

operational without significant deviations in its output. Since a continuous stream of time series data is required, a method should be found for the optimal imputation of data, such that the model's predictive performance does not suffer.

Lastly, the scalability of the system is also a potential challenge. At present, extensive and accessible databases containing historical parking data are scarce [11]. Adding to this, machine learning models perform best when trained to a distinct set of training data. It will therefore be burdensome to make the resulting model perform well on other garages and lots. Hence, this thesis should mainly focus on developing a concrete methodology rather than developing a *'one-size-fits-all'* model.

1.4 RESEARCH QUESTIONS

The main research question for this thesis can be defined as follows:

> *How can an accurate and efficient machine learning methodology be developed for predicting and visualizing the influx, outflux and occupancy rate of parking areas in real-time on a horizon of up to 60 minutes ahead?*

In order to answer the above question, the following subquestions should be answered first:

> *Which data features are most significant as input for the predictive model?*
>
> *Which machine learning techniques, among multiple candidates, are most suitable to predict occupancy rate, influx and outflux?*
>
> *Which configuration of model parameters, built upon the previously defined techniques, yields the best performance when predicting occupancy, influx and outflux on a horizon of up to 60 minutes ahead?*
>
> *What is a suitable system architecture for executing the prediction processes continuously and autonomously using a real-time data feed?*
>
> *How can the output predictions be visualized, such that useful insights are provided to stakeholders both in retrospective and in real-time?*
>
> *To what extent is the resulting system transferable towards other parking areas?*

1.5 REPORT OUTLINE

This thesis consists of five chapters which will gradually build towards answering the main research question. Chapter 2 contains background information on machine learning and a careful assessment of the specific techniques applied to the parking and traffic domains. This will be done by means of a literature study. Subsequently, the third chapter will elaborate on the practical implementation of the system. Here a methodology for data collection and preparation will be discussed, as well as a procedure for training and testing the resulting model. Furthermore an overarching system architecture is developed. Chapter 4 will describe the realization of the core machine learning model and the complete system, after which the test results are presented and discussed. Here the scalability and transferability of the solution are evaluated as well. Ultimately, chapter 5 concludes the thesis by answering the research questions and defining future work.

2 THEORY AND BACKGROUND

This chapter contains an assessment of existing machine learning methodologies and their predictive power within the parking domain. First of all, background information is provided about the field of machine learning. A comprehensive literature study is then performed to analyze existing knowledge, aiming to identify relevant outcomes as well as gaps and remaining problems which provide opportunities for further research. Since the performance of a predictive model is highly characterized by the input data it is fed [12], the first step of the literature review is to define the relevant input variables based on a study of existing research on parking prediction. Subsequently, a complete analysis and pre-selection are performed of the available machine learning techniques, followed by an assessment of metrics for evaluating and comparing the models. Lastly, the pre-selected techniques will be described and explained in more detail.

2.1 INTRODUCTION TO MACHINE LEARNING

Machine learning is an application of artificial intelligence where a system autonomously learns from prior experience without the use of predefined equations as a model. Training data is fed stepwise to the machine, after which algorithms gradually build a mathematical model which optimally fits this data. Using this model, the machine can then produce predictions or decisions without being explicitly programmed to complete the intended task. [13]

2.1.1 Types of learning

The domain of machine learning consists of *supervised learning* and *unsupervised learning*. In supervised learning, the dependent variable is present to guide the learning process, whereas in unsupervised learning there is no knowledge of the desired output since discovering patterns is the main objective [12] . Supervised learning is therefore the most optimal way to predict an accurate output based on future input variables (which are defined in Section 2.2.1). In the context of parking, this approach is thus desired.

Supervised learning problems can be further divided into *classification* and *regression* problems. Classification is a technique for pre-

dicting discrete responses where the output is classified as one of the qualitative targets. On the contrary, regression is used when the output variable is quantitative, such as the influx, outflux and occupancy rate which this research is focusing on [12]. Hence, it is indisputable that a regression technique should be chosen in the context of predicting parking occupancy rates.

2.2 LITERATURE STUDY

The goal of the literature study is to identify relevant outcomes, gaps and remaining problems in current knowledge and research about parking prediction. It provides empirical insights into the input variables, machine learning techniques and validation methods, as well as a comparative assessment of their relevance according to existing research. This entails a pre-selection of machine learning techniques, which are later assessed more thoroughly using empirical tests on the relevant datasets. Altogether, the identified outcomes and shortcomings are used as basis for the further course of this research.

2.2.1 Relevant variables

In the real world, there are many factors which influence parking behaviour. Within machine learning, these factors can be quantitatively translated to input variables (or *independent variables*) which, based on their respective values in time, ultimately determine the predicted output (or *dependent variable*) of the model. According to Guyon and Elisseeff [14], the predictive power of a model is highly dependent on the chosen variables. *Feature selection* is therefore a crucial task, not only to optimize performance, but also to provide a better understanding of the underlying processes. A selection of eleven articles was therefore made to determine the most promising predictive variables. This being said, it should be noted that all selected articles solely consider the occupancy rate as dependent variable in their research. For the purposes of this research, this is acceptable since the in- and outflux simply determine the change of occupancy rate, as visible in the following equation. At time t, the change of the occupancy rate O_t is determined by subtracting the outflux $f_{out,t}$ from the influx $f_{in,t}$:

$$\Delta O_t = \Theta(f_{in,t} - f_{out,t})$$

Parking flows are highly dynamic over time, and therefore temporal variables are among the most prominent candidates in terms of predictive ability. Chen et al. [15] demonstrate that seasonal variables, such as time and date, lead to dramatically improved prediction accuracy. This is supported by others, for instance by Badii, Nesi and

Article	Variable							
	Time of day	Weekday	Temperature	Rain	Holiday	Event	Traffic flow	Historic occupancy
Vlahogianni et al. [9]	X	X			X			X
Badii, Nesi and Paoli [16]	X	X	X				X	X
Hampshire et al. [17]	X	X		X		X		
Chen [18]	X	X				X		
Zheng, Rajasegarar and Leckie [19]	X	X						X
Camero et al. [20]	X	X						
Chen et al. [15]	X				X			
Lijbers [21]	X	X	X	X	X			X
Monteiro and Ioannou [22]		X						X
Reinstadler et al. [23]	X		X	X	X	X		
Pflügler et al. [24]	X	X	X		X	X	X	

Table 2.1: Matrix of independent variable utilization

Paoli [16] who regard time variables as the baseline for their model. As a matter of fact, the variable *time of day* is mentioned unanimously in almost every article, as visible in Table 2.1. This is comprehensible, as the occupancy might rapidly increase during the morning during rush hour, while staying low at night.

Another time-related variable is the *weekday*, i.e. ranging between Monday until Sunday. Lijbers illustrates that "whether it is a working day or a non-working day (like in the weekend) might influence occupancy", and claims that the *weekday* variable would therefore enhance the model's response to such phenomena [21, p. 21]. Most articles support this stance, even though Hampshire et al. [17] and Badii, Nesi and Paoli [16] suggest that the actual importance of this variable is quite low. Overall, however, the *weekday* is mentioned in almost every article and can therefore be regarded as a potentially influential variable.

Additionally, *historic occupancy* is also regarded to be a strong predictor. Vlahogianni et al. demonstrate using genetic optimization that "a lookback time window of 5 minutes in the past may be efficiently used to predict parking occupancy (%) up to 30 steps in the future with high accuracy" [9, p. 198]. Similarly, Zheng, Rajasegarar and Leckie [19] argue that a 30% performance gain can be achieved by including several steps from the past, in addition to just the *time of day* and *weekday* variables. Badii, Nesi and Paoli [16], as well as Monteiro and Ioannou [22], suggest a similar effect. On the contrary, some of the other articles do not endorse the *historic occupancy* as input variable. The reason for this seems to be that these articles do not use a data source which supplies measurement points up to the last minute. For instance, Reinstadler et al. [23] define their research as a 'data-mining problem', which entails that their data points are independent and unordered over time, unlike time series data. Considering that multiple authorities and municipalities disclose complete historical time series data as well as a real-time feed [25], it can be concluded that inclusion of the *historic occupancy* variable is both feasible and potentially beneficial for upcoming research.

Other variables which are often cited in research are related to weather. In the majority of relevant articles, a weather variable such as temperature or rain is used as input of the model. Reinstadler et al. [23] argue that, because weather data has a high weight in their resulting model, these variables are very important for the accuracy of predictions. Nevertheless, Chen et al. [15] challenge this by stating that weather conditions such as rain and fog have little impact on parking occupancy. Their statement was based on analysis of daily parking patterns in the city of Dublin. However, Badii, Nesi and Paoli [16] demonstrate that the importance of temperature and rainfall varies significantly per distinct parking location. It can therefore be argued that the statement by Chen et al. does not hold firm ground. Overall, one can conclude that the temperature and rain variables are important to consider, even though it is uncertain whether they will actually increase the predictive performance of the model.

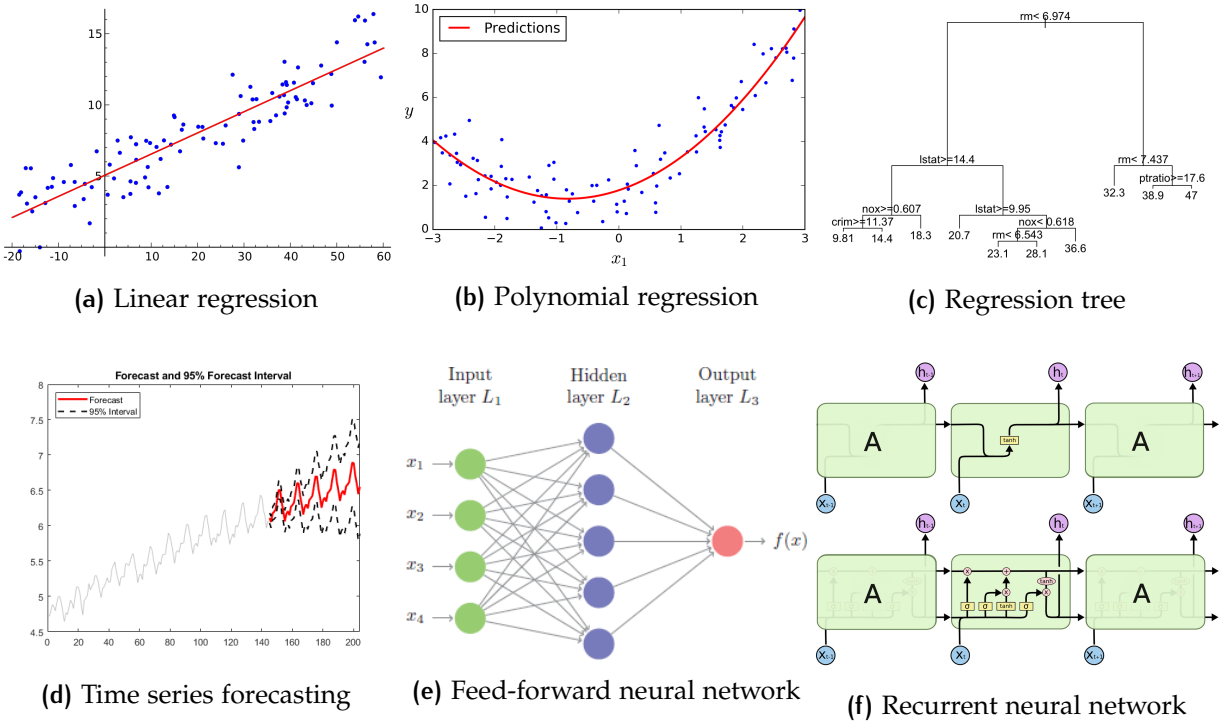
The variables *event*, *holiday* and *traffic intensity* could supposedly provide a useful addition to the model. Even though Pfügler et al. [24] claim that they are of secondary importance for modeling parking flows, they mention that "traffic information is an important factor for the availability of parking spaces" [24, p. 364]. This stance is supported by Badii, Nesi and Paoli [16], who however remark that the traffic flow variable is only relevant when sensors are located on streets leading to the parking garage, and when measurement data is "available for the previous hour with respect to the time of prediction" [16, p. 8]. On the premise that relevant and comprehensive data streams from nearby sensors are available both in real-time and historically, *traffic flow* should definitely be considered as input variable for the model. Last-mentioned is also applicable to *event* and *holiday* since a majority of articles mention these variables. For instance, Reinstadler et al. [23] state that external attributes like events and holidays are extremely important since they influence parking occupancy. Chen et al. [15] support this by demonstrating how the predictive error and standard deviation spike during the Christmas holidays.

All in all, it has become evident that time variables, namely *time of day* and *weekday*, are the most prominent predictors for a machine learning model on parking occupancy. The *historic occupancy*, provided that a lookback window is feasible with the given data feed, is also a very important predictor. Secondary to this, the variables *temperature*, *rain*, *holiday* and *event* could increase predictive power because of their supposed relationship with traffic flows, and consequently also parking flows. Lastly, it remains uncertain whether *traffic flow* variables are good predictors for parking models, even though this seems to be mainly related to a lack of research and reliable data sources. Hence, there is still a clear opportunity for these variables to be successfully applied onto the predictive model.

2.2.2 Analysis of contemporary techniques

In Section 2.1, supervised regression was determined to be the most suitable type of machine learning for predicting parking states. State-of-the-art machine learning provides many such techniques. In order to determine the optimal technique, it is best to assess the possibilities in order of their computational complexity and ease of implementation. Stolfi, Alba and Yao [26] performed tests using six predictive techniques on parking data from the city of Birmingham. Out of these techniques, which were selected based on their simplicity and ease of use, they observed that *polynomial regression* and *time series prediction* (illustrated in Figure 2.1b and 2.1d, respectively) provide the best results. Camero et al. [20] acknowledge this, but remark that there are more sophisticated techniques which can help to enhance the predictive accuracy.

In particular, *regression trees* (see Figure 2.1c) allow for higher model complexity while remaining accessible and flexible. Reinstadler et al. [23] claim that this technique generates better predictions than the ones mentioned by Stolfi, Alba and Yao. The authors argue that regression trees are "more flexible and often also more powerful" than time series techniques such as *ARMA* and *ARIMA* [23, p. 6]. This follows from the fact that time series forecasting techniques only consider the temporal seasonality patterns in the parking occupancy data [23]. As a result, they are unable to cover the eight variables which



(a) Linear regression **(b)** Polynomial regression **(c)** Regression tree

(d) Time series forecasting **(e)** Feed-forward neural network **(f)** Recurrent neural network

Figure 2.1: Illustration of machine learning techniques

were proposed in Section 2.2.1. The positive attitude of Reinstadler et al. regarding regression trees is shared by Hampshire et al. [17]. Based on analysis of four machine learning techniques, the authors conclude that the performance of the regression tree "is superior to the other measures" [17, p. 296]. According to the authors, this is caused by the fact that ordinary linear regression and time series techniques assume that all features are independent. A regression tree, on the other hand, is able to expand the tree branches such that any correlation can be handled properly. Notably in the case of parking, where input variables are often correlated, regression trees prosper [17].

Additionally, neural networks appear to produce promising results. Hampshire et al. [17] performed an analysis on two types of *feed-forward neural networks* (illustrated in Figure 2.1e), both of which proved to be more successful than 'ordinary' linear regression. The authors suggest that a hybrid of neural networks and regression provides a robust prediction platform. The use of neural networks is further supported by Pfügler et al. who state that "neural networks are particularly suitable for predicting events where little or nothing is known about the underlying relationships and features of the events, but enough training data or observation values are available" [24, p. 367]. Furthermore, the authors argue that neural networks enable continuous learning where the model can be consistently retrained, in case that a real-time data feed is available. A preliminary exploration of possible data sources shows that real-time feeds are available for the previously defined variables, which definitely advocates for the use of neural networks. Yet, Snellen [11] highlights that neural networks are inconvenient due to their 'black-box' concept which prevents stakeholders from knowing the effect and influence of each variable. Furthermore, the author maintains that neural networks are often unacceptable for real-time predictions due to their computational complexity. While research by Badii, Nesi and Paoli [16] indeed confirms that training times are longer than regular regression methods, it proves that the actual time to make a prediction is only 0.0031 seconds, which is even less than the 0.0052 seconds it takes for a linear regression model. For a real-time application, prediction times are far more meaningful than training times, predominantly since there is no need to retrain the model very frequently. [16] Overall, despite some shortcomings, neural networks seem to be a very promising technique in the context of parking occupancy prediction.

One should add that there are more variations of neural networks. Previously mentioned articles mostly used traditional *feed-forward neural networks*, i.e. networks where nodes do not form a cycle. There is also a variant of neural networks where nodes can form cycles and hence contain feedback loops. This is called a *recurrent neural network*, of which an example is illustrated in Figure 2.1f. Connor and Atlas

state that for some processes "feedback allows recurrent networks to achieve better predictions than can be made with a feed-forward network with a finite number of inputs" [27, p. 301]. Recurrent networks are able to interpret sequences of inputs which rely on each other for context. For instance, the parking occupancy of one minute ago relies also on the occupancy of the occupancy two minutes ago, and so forth [27]. Li, Li and Zhang [28] acknowledge this and demonstrate that *LSTM* (a specific kind of recurrent neural network) outperforms a regular neural network on prediction of available parking spaces. The authors however remark that prediction times are significantly longer than traditional feed-forward neural networks, which forms a bottleneck for a real-time predictive application.

In conclusion, it has become apparent that time series forecasting techniques are unsuitable for the input variables defined in Section 2.2.1. Traditional linear and polynomial regression techniques are feasible, but are regarded as being lightweight compared to other, more sophisticated methods. On the contrary, regression trees are positively regarded by multiple authors because of their transparency as well as their ability to perceive correlations between variables. Neural networks have the potential to perform even better, even though they lack in their ability to provide transparent insights about the internal structure due to their black-box concept. Feed-forward neural networks seem to outperform recurrent neural networks in term of prediction speed, which makes them more suitable for a real-time predictive system. All in all, regression trees come forward as the safest choice in terms of predictive power and explainability to stakeholders, with feed-forward neural networks being another crucial technique to examine because of their additional performance boost and ability to continuously retrain the model.

2.2.3 Performance evaluation

In order to validate machine learning models and compare their predictive performance, a standardized evaluation metric should be defined. According to Caruana and Niculescu-Mizil [29], many of the available metrics are unsuitable for comparison across multiple datasets. This is especially caused by the fact that the range 0 to p of their values depends on the used dataset. On top of that, for some metrics lower values indicate better performance, while higher values are better for others. The authors therefore define a normalized scale with range [0,1] as a means "to permit averaging across metrics and problems" [29, p. 3].

The *coefficient of determination*, or R^2, is a popular metric for assessing predictive models. It indicates the strength of the relationship between the model and the dependent variable, and has a range of [0,1]. Kvalseth points out that many data analysts utilize R^2 to assess

the "goodness of fit of the models" [30, p. 281]. In the context of parking occupancy prediction, it is mentioned by both Zheng et al. [19] and Badii, Nesi and Paoli [16]. However, while being a useful metric in itself, Kvalseth argues that it is often misused [30]. Willmott [31] acknowledges this and demonstrates that the R^2 score of a model is often unrelated to the actual size of the error between the predicted and actual value. Additionally, Pelánek [32] argues that it can be interpreted differently for different regression techniques. It is therefore difficult for stakeholders to compare multiple models and techniques using R^2. Willmott eventually concludes that a conventional error metric, such as the *mean absolute percentage error* (MAPE), would provide better insight into the actual performance of a model [31]. Badii, Nesi and Paoli [16] take a similar stance but challenge the notion that MAPE, which is normalized and thus ranges from 0 to 1, is suitable for the parking domain. The authors namely state that this metric has the disadvantage of becoming infinity or undefined when the parking occupancy approaches zero.

This problem can be solved by applying the *mean absolute scaled error* (MASE). The MASE, which is obtained by dividing the tested model's *mean absolute error* by that of an arbitrary naive model. Like MAPE, it is also independent of the scale of the data but will never encounter the problem of zero division [16]. Hyndman endorses this, and even argues that MASE should become "the standard metric for comparing forecast accuracy across multiple time series" [33, p. 43]. In contrast to the simpler *mean absolute error* and *mean squared error* metrics, Hyndman demonstrates that MASE is suitable even when the data exhibit a trend or a seasonal pattern. Since parking occupancy is characterized by several seasonal patterns, as reasoned in Section 2.2.1, MASE arguably suits best in this context [33].

$$MASE = \frac{MAE}{MAE_{naive}} \text{ where MAE is defined as:}$$

$$MAE = \frac{1}{n}\sum_{j=1}^{n} |y_{pred,j} - y_{actual,j}|$$

Because of its scale-invariant nature, it should be noted that MASE is a more demanding metric for stakeholders (e.g. road operators and traffic controllers) to understand and communicate than the more common *mean absolute error* (MAE) and *mean squared error* (MSE) [33], the latter of which penalizes large errors more than small errors. Besides, MASE is more computationally expensive than its simpler counterparts MAE and MSE. Arguably, MASE is only beneficial when comparing model performances with each other and with a naive model, and less when validating a model itself during the training phase [16]. A balanced combination of these metrics would therefore be optimal: MAE and MSE would then be used to provide a tangible and intelligible performance measure for stakeholders, such that they

are able to understand how good or bad the model predicts in which situation. Also, MSE is used as the essential loss function during the training process of the individual models. Given the resulting models, MASE would then be useful to observe how the models perform against a naive model, and therefore provides stakeholders with a strong insight into the actual added value of the model. Also, it opens the possibility to empirically compare the performance of the influx, outflux and occupancy rate models, respectively.

Overall, a combination should therefore be used of MAE and MSE as during the training and validation phase, and MASE during the inter-model comparative testing phase.

$$MSE = \frac{1}{n}\sum_{j=1}^{n}(y_{pred,j} - y_{actual,j})^2$$

2.2.4 Conclusions

Selecting and validating a powerful model is crucial in order to accurately predict the influx, outflux and occupancy rate of parking areas in real-time. The goal of this literature review was to determine the relevant input variables, choose the most suitable machine learning technique to accommodate these variables and finally select a fair and reliable metric to assess the resulting models.

Using a systematic review of relevant sources, it was determined that temporal variables were the most important for modeling the parking occupancy, together with a lookback window of historic occupancies. The weather and event variables are of secondary importance. Even though the influence of traffic flow has not been thoroughly researched yet, preliminary results are sufficiently promising to regard it as a tertiary variable. Overall, it is recommended to sequentially add these variables to the model and evaluate their effect on the metric (defined in Section 2.2.3) in their order of potential importance.

With the chosen input variables in mind, the next step was to assess multiple machine learning techniques based on their predictive power, computational complexity and suitability with the aforementioned variables. Because of their potential predictive performance in the parking domain, as well as their capability to operate in a real-time environment, both neural networks and regression trees were found to be solid machine learning techniques for building the predictive model. A conclusive testing procedure will be carried out to make a final decision on which technique performs best on the available datasets corresponding to the defined input and output variables.

Finally, a balanced combination between several metrics was determined to be the most suitable method to train, validate and compare

the neural networks and regression trees. The *mean squared error* provides a computationally efficient way to validate and optimize the model to maximize its performance on the training set. Moreover, together with the *mean absolute error*, it provides a comprehensible and precise way for stakeholders to understand the model's actual errors on a natural unambiguous scale. After compiling and training multiple models, i.e. several mutations of neural networks and regression trees, MASE can be used to empirically compare their predictive performance. Especially regarding its tolerance towards temporal data containing trends and seasonalities, as well as its suitability for model comparison across multiple datasets, MASE is arguably the most suitable metric for inter-model comparative testing. Additionally, it provides a practicable insight into a model's performance with reference to a naive model, which is especially advantageous to find out whether the model actually possesses any added value.

A critical limitation of contemporary research in the machine learning domain is its highly fragmented nature: many different data sources and parameters are utilized to assess models, input variables and validation metrics. Directly comparing methodologies on a quantitative basis is therefore a challenging task which, in turn, complicates the decision-making process. This literature review has therefore attempted to perform a comprehensive and objective selection of methodologies based on their factual suitability within the specific context. A careful process of training, validating and testing the resulting methodologies with the relevant datasets is recommended in order to precisely and definitively examine which one is most relevant within this specific context.

2.3 PRE-SELECTED TECHNIQUES

2.3.1 Regression trees

Decision trees, which are generally applied to classification problems (see Section 2.1.1), utilize a tree structure to recursively classify input variables to a fixed set of output variables [13]. Upon training a decision tree model, the dataset is split into smaller and smaller subsets while an associated tree structure is incrementally built at the same time. As illustrated in Figure 2.2, the resulting tree contains three types of nodes, all of which have their own function within the model:

- A single **root node**, which has one or more outgoing branches and no incoming branches. This node corresponds to the strongest input variable of the model.

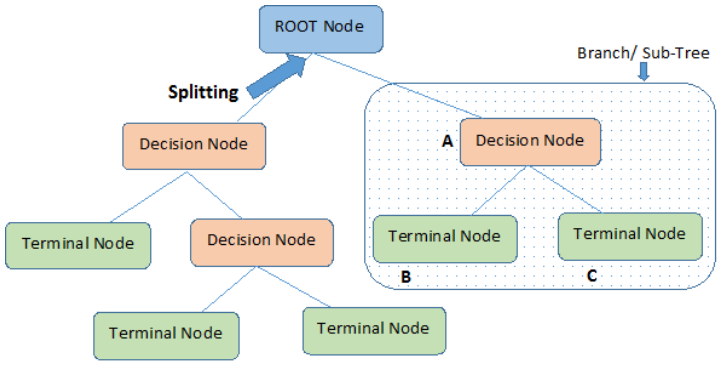


Figure 2.2: Illustration of a decision tree structure [34]

- **Decision nodes** which are fed by one incoming branch and one or more outgoing branches.
- **Terminal nodes** (or *leaves*) are fed by a single incoming branch and have no outgoing branches. They terminate the tree structure, and therefore represent a classification (or decision).

Essentially, these nodes perform logical operations on the incoming branch and guide it to another node based on a set of criteria which were defined during the training phase of the model. The predictive ability of a decision tree model is therefore highly characterized by the complexity of these criteria and relations between nodes. Overall, the strength of this technique is its ability to model complex relationships using fundamental logic rules.

Regression trees are a variant of conventional decision trees, with the obvious difference of being applicable to regression problems. Instead of classifying an outcome to a predefined set of categorical variables, regression trees output a numerical continuous value, e.g. the influx, outflux or occupancy rate of a parking area. When training a regression tree, every input variable (i.e. independent variable) is recursively partitioned based on minimization of the error between the predicted value and the actual value in the training set. New data can be filtered and lands into one of the leaf nodes which corresponds to a numerical value. This makes it possible to generate predictions.

Nevertheless, it should be noted that classification and regression trees are known to suffer from bias and variance. Generally speaking, simple trees will result in a large bias, while complex trees result in large variance (i.e. overfitting). *Ensemble methods* combine multiple trees in pursuance of increased robustness and better predictive performance. They are implemented in the form of *bagging* and *boosting*, which both produce new subsets of the training data by random sampling with replacement. Subsequently, each collection of subset data is used to train their respective decision trees, which results in an ensemble of models. Bagging techniques are used to make the resulting model less prone to individual trees overfitting the training

data. A widely used implementation is **random forest**, which takes one extra step as opposed to regular bagging techniques: in addition to randomly selecting subsets of data, it also takes the random selection of features to grow trees. Its prediction is given based on the aggregation of predictions from all trees in the model. The main advantage of random forests is the potentially high performance while maintaining relative ease of implementation, especially since the tuning of hyperparameters is fairly easy. Generally speaking, finding the optimal balance between the number of trees in the model and decent computational performance is the most important aspect of hyperparameter tuning. Above all, random forests generally provide good scalability and suitability to a wide range of machine learning problems. [12] [13]

2.3.2 Feed-forward neural networks

An *artificial neural network* (ANN) is a computational model which is inspired by the way a human brain processes information. This technique has proved to be successful across many applications of machine learning, including regression problems [12].

The fundamental unit in a neural network is a **neuron**, often called a *node*. It receives an input from one or multiple other neurons, or from an external data source. Each input has an associated *weight*, which is assigned based on its relative importance to other inputs. Subsequently, in order to produce an output value, an activation function is applied to the given inputs. Additionally, a *bias* input contributes a constant value to the function, which may be critical for successful learning. Frequently used activation functions are *ReLU*, *Softmax*, *Sigmoid* and *Tanh*.

The *feed-forward neural network* (FFNN) is the conventional type of neural networks. As visible in Figure 2.3, it contains multiple neurons which are arranged in layers. The specific property of feed-forward neural networks is that the connections between neurons do not form a cycle. Hence, information can only flow in forward direction. Neurons from adjacent layers have connections between them (each with

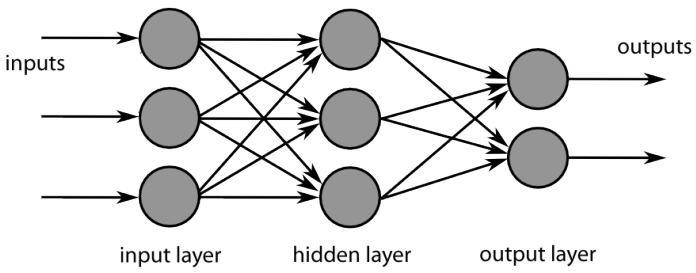


Figure 2.3: Illustration of a FFNN structure [35]

an associated weight), such that the outputs from one layer of neurons serve as inputs for the next layer. A feed-forward neural network can consist of three types of neurons:

- The first layer consists of **input neurons**, which feed the data from external sources to the rest of the model. No computation is performed in any of these nodes - they only pass the given information to the hidden nodes in the next layer.
- **Hidden neurons** are not directly linked to the outside world. Their function is to transfer information from the input layer towards the output layer. A network can contain multiple hidden layers.
- **Output neurons** are located in the last layer, i.e. the output layer of the network. They are responsible for the final computations, as well as the transfer of the information to the outside world.

Feed-forward neural networks are very useful to overcome the problem of non-linearity in some machine learning problems. In combination with their flexible structure, i.e. the ability of adding or removing neurons and hidden layers to the model, this makes them applicable and scalable to a wide range of tasks. By the same token, the output layer can contain an arbitrary number of neurons, which makes this technique suitable for multi-output predictions. This is particularly useful when predicting time series, where each predictive horizon (i.e. 1 minute ahead, 5 minutes ahead, 10 minutes ahead and so on) can be represented by its own output node. It it therefore obvious that feed-forward neural networks are theoretically very suitable to the task of predicting flows and occupancy rates of parking areas on a horizon of up to 60 minutes. [12] [36]

3 METHOD

To reach the goal of this thesis, i.e. developing and implementing a methodology to predict the influx, outflux and occupancy rate of parking areas, the project is divided into multiple phases. Together these phases and corresponding steps form the method for further execution of this research.

3.1 STRUCTURE AND PROCESS

The main structure of the method can be described using the following phases and their corresponding substeps:

- Collecting the relevant data from external sources
  - Preliminary exploration of the datasets
  - Specifying the definitive input and output features as established in Section 2.2.1
  - Selecting and describing the historical and real-time data sources
- Preparing and pre-processing the data before feeding it to the candidate models as training, validation and testing data
  - Translation of data attributes to their corresponding input and output features
  - Partitioning the datasets in training, validation and testing subsets
- Training and validating the candidate models
  - Determining and optimizing the structure of the models
  - Model optimization using hyperparameter tuning
  - Development of final models using the optimal configurations
- Inter-model comparative testing before selecting the definitive model to implement into the predictive system
  - Defining a naive model for benchmarking purposes
  - Comparing both candidate models using the benchmark metric and selecting the one which performs best

- Visualization of real-time measurements and predictions
  - Ideation oriented towards prospective stakeholders
  - Creation of the visualizations derived from ideas defined during the prior step
- Implementing a comprehensive predictive system in practice
  - Designing a suitable system architecture for predicting and visualizing in real-time
  - Realization of a prototype and evaluating the quality of predictions over time
  - Assessing the transferability of the system towards other parking areas

3.1.1 Tools

In order to execute this research thoroughly, multiple software tools were used. The Python programming language, with its extensive range of libraries for data science and machine learning, provides a solid basis for this purpose. All used libraries are open-source and come with extensive documentation as well as an active user base.

The Python libraries *Pandas* and *Numpy* were used to process and prepare the datasets, and *Seaborn* and *Matplotlib* were used to visualize the data during the exploration and measurement phases, respectively.

Additionally, the *Scikit-learn* and *Keras* libraries were utilized to build, train and test the machine learning models. Scikit-learn provides a wide range of 'traditional' machine learning algorithms as well as tools for training and testing. It therefore provided the necessary interface to implement the random forest model. Keras is a high-level library which facilitates deep learning, i.e. it can be used to construct, train and validate multiple types of neural networks, including feed-forward neural networks. The library provides a large number of frameworks and hyperparameters which can be tuned to maximize model performance. The feed-forward neural network was therefore implemented using Keras.

To store and process the incoming data of the real-time predictive application, the server was equipped with an *SQLite* database which could be accessed using the designated *Sqlite3* library for Python. To deploy a web server and stream predictions in real-time, a combination of the *Flask* and *SocketIO* libraries was utilized. The JavaScript-based libraries *MetricsGraphics.js* and *Chart.js* were used to visualize the predictions for end users.

Independent variables		Dependent variables
Weekday	Rainfall	Occupancy rate
Time	Preced. occupancy rate	Influx
Air temperature	Traffic flow	Outflux

Table 3.1: Overview of independent and dependent variables

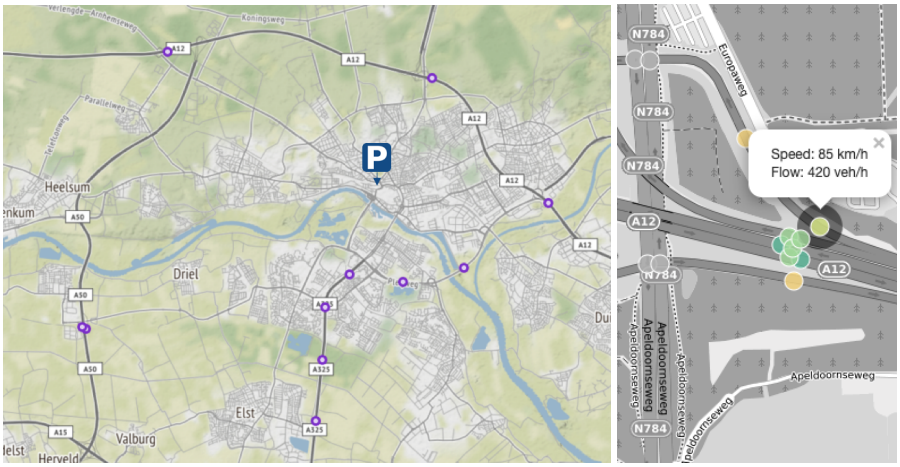
3.2 DATA COLLECTION AND EXPLORATION

The relevant independent and dependent variables for the proposed model were previously deducted and defined in Section 2.2.1. To provide a definitive overview, these variables are listed in Table 3.1. In order to develop and operate a functional predictive model, both *historical* and *real-time* data sources should be available and operational. Based on the concept that the newly created model should learn from the situations of the past, historical data sources serve as the basis to develop and tune the model (i.e. training, validating and testing). This historical dataset should comprise a vast number of entries which contain a value for each independent and dependent variable. Such a dataset can therefore be established using *data fusion*, which is "the process of integrating multiple data sources to produce more consistent, accurate, and useful information than that provided by any individual data source" [37]. Subsequently, to actually make predictions with the resulting model, real-time data sources should be accessible in order to provide actual values to the input of the model.

3.2.1 Historical data

Within the historical dataset, **time** and **weekday** are *key* variables which are connected to every other variable. To illustrate, the occupancy rate, traffic flows and air temperature are all characterized by a certain timestamp. These temporal variables can therefore be regarded as interconnecting variables which bound the other variables together to form entries in the dataset. As a result, the time and weekday variables are not collected from individual data sources, but are rather composed by collecting the historical data for the other variables.

Parking data is inevitably the most crucial data source within this research, given the fact that the intended model aims to predict the three parking variables **occupancy rate**, **influx** and **outflux**. Unfortunately, historical open data sources for parking areas are still scarce today [38]. The decision was finally made to collect the parking transaction data from the *Open Parkeerdata* portal of the Municipality of Arnhem [39], which arguably provided the most extensive historical database of parking transactions while also maintaining a real-time feed (which will be further explained in Section 3.2.2). Using the



(a) Measurement locations (marked purple)

(b) Placement near highway exit

Figure 3.1: Selection of locations for traffic flow data source

transaction entries, the dataset enables us to derive the three dependent variables and the **preceding occupancy rates**. The data source provides transaction data of the Centraal, Musis and Rozet parking garages in Arnhem, The Netherlands. Hence, the scope of the data collection (and therefore the research as a whole) becomes the city of Arnhem. In total, 233 MB of parking transaction data was retrieved from this source, ranging from August 2017 until April 2019.

Traffic data was gathered from the *Nationale Databank Wegverkeersgegevens* (NDW) using its *Dexter* [40] platform. In total, eleven measurement locations were selected, all of which are part of the *MoniCa* loop detection system operated by *Rijkswaterstaat*. All locations can be distinguished by their own identifier (formatted as `RWS01_MONICA_...`). As visible in Figure 3.1, the sensors are located on the orbital highways and freeways around Arnhem - specifically on highway exits and access roads. Hence, they measure traffic driving towards the city center (where the garages are located), such that the data gives an adequate indication of the intensity of incoming traffic. Overall, after considering the availability and validity of the measurement sensors, 15.96 GB of traffic flow data was retrieved from NDW Dexter, ranging from November 2017 until April 2019.

To gather weather data, the open databases of the Dutch meteorological institute *KNMI* [41] were utilized. Using a web service, the hourly data of several variables can be queried. The measurements of the Deelen weather station were chosen because of its close proximity (i.e. 10 km) to the city center of Arnhem. The KNMI data source provided the possibility to obtain the **air temperature** at 1.5 meter height (measured in 0.1 °C) and **rainfall** (a binary variable denoting whether rain has fallen in the past hour) variables. The hourly data from August 2017 until April 2019 were downloaded locally.

3.2.2 Real-time data

In order for the models to make predictions, values for all independent variables should be fed to the model. For instance, to predict the occupancy rate, the model expects an input consisting of the current weekday, time, temperature, rainfall, traffic flows and the preceding occupancy rates. This is where real-time data comes into play.

To provide the model with inputs of **preceding occupancy rates**, a real-time feed of relevant parking data is crucial. Based on the approach taken in Section 3.2.1, an essential task is thus to obtain a live feed of data from the parking areas in Arnhem. The *Open Data Portaal* [42] of the Municipality of Arnhem provides a section with dynamic parking data of all parking areas, including the aforementioned Centraal, Musis and Rozet garages. The data, which can be fetched in JSON format, is dynamically updated every 11 minutes and 20 seconds. After retrieving the JSON file, the occupancy rate is derived from the values of the `vacantSpaces` (the real-time number of free spaces in the garage) and `parkingCapacity` (the total number of parking spaces in the garage) attributes. Unfortunately, the absolute values for influx and outflux cannot be retrieved from this data source. This limits the knowledge and memory of the model to merely the preceding occupancy rates.

Similar to the historic traffic data, the real-time **traffic flow** data was also retrieved from the NDW. However, since the Dexter platform is only meant for exploring and exporting historical data, the *Open Data Service* [43] by NDW was used for this purpose. This platform provides a set of files which are updated in real-time. The `trafficspeeds.xml.gz` file, which is a compressed XML file, contains the current speeds and flows of the *MoniCa* and *MoniBas* loop detection sensors in The Netherlands, including the measurements locations which were previously defined in Section 3.2.1.

In contrast to the historical data source, the KNMI unfortunately does not provide an accessible API for real-time weather data. For this reason, real-time **temperature** and **rainfall** data was obtained from the *Weerlive API* [44]. This third party also obtains its data from the KNMI, but distributes it as an API in JSON format which makes it more accessible and convenient to process. It was assured that measurements from same weather station were used. After obtaining an API key and specifying the location (i.e. Deelen), data was retrieved in real-time in a ten minute interval.

3.3 DATA PREPARATION

As described in Section 3.2, historical and real-time data sources were queried to obtain reliable input streams for every input variable. All

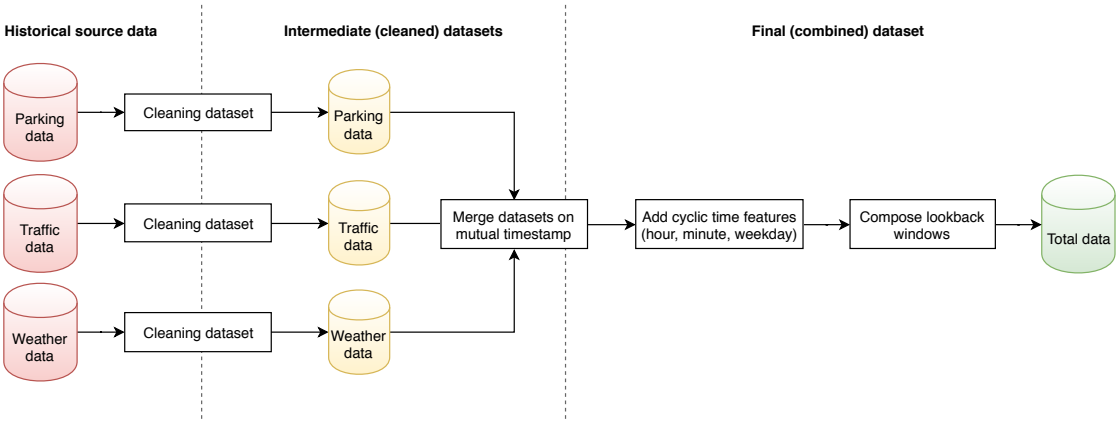


Figure 3.2: Process of cleaning and processing the historical data

sources have their own file format and data structure, and therefore the data should be refined before being supplied to the training, validation and testing processes. As a consequence, the decision was made to clean and process all incoming datasets, and combine the relevant data features into one file called `totalData.csv`. The `.csv` (*comma separated file*) format was chosen because of its interpretability, low time-space complexity and convenience in the Pandas library. First of all, the source data files are imported and appended to a Pandas DataFrame (the main multi-dimensional data structure in Pandas), after which they are cleaned. *Cleaning* here entails: the removal of obsolete columns from the dataset, deleting erroneous rows, filling missing values and transforming the structure of the data. The resulting `.csv` files of the cleaned parking, traffic and weather datasets are then merged based on their mutual `timestamp` column, after which this column is converted into three other columns `hour`, `minute` and `weekday`. A comprehensive overview of the process is shown in Figure 3.2.

3.3.1 Cleaning the historical data

The raw parking transaction data were retrieved as multiple files - one for each month. As a result, 21 files were used, spanning from August 2017 until April 2019. Using a Python script, these different files were all appended to a Pandas DataFrame `df`. Then, based on the fact that the *Centraal* garage (with a capacity of 1050 parking spaces) is the largest of the three aforementioned parking areas, only the transaction data from this garage was extracted. After this, the obsolete columns `garage_nm` (i.e. since all of its values now equal *'Centraal'*), `card_type_nm` and `pay_parking_dt` were dropped from the dataset, such that only the incoming and outgoing timestamps remained. Every row thus denotes the parking movement of a particular vehicle, characterized by the *in* and *out* timestamps. Hence, all

```
# Select only the Centraal garage
df = df[df['garage_nm'] == "Centraal"]

# Drop unimportant columns and rows
df = df.drop(columns=['garage_nm', 'card_type_nm', 'pay_parking_dt'])
df.columns = ['in', 'out']

# Merge the in- and outflux
df = df.melt(value_vars=['in', 'out'], value_name='timestamp',
             var_name='flux')

# Using crosstab, list in- and outflux in separate columns
df = pd.crosstab(pd.to_datetime(df['timestamp']), df.flux)

# Resample the in- and outflux for the desired time interval
df = df.resample('T').sum()

# Save output as a combined csv file
df.to_csv('fullParkingData.csv')
```

Listing 1: Cleaning process of parking data

rows with one or more NaN values were dropped in order to exclude incomplete parking movements (e.g. a car which enters the garage but never leaves would be regarded as invalid). The reason why a whole row is deleted in this case is to guarantee the stability of occupancy rates: since the occupancy rate is based on the cumulative sum of influx and outflux, incomplete rows (when not compensated for by another incomplete parking movement in opposite direction) could destabilize the occupancy rate over time. Keeping only the complete rows, which thus have both an in and out timestamp, will thus assure the stability between both ends of the dataset. Subsequently, the DataFrame was reshaped using the *melt* method, resulting in a separate `timestamp` column and a `flux` column which denotes the direction of the parking movement at that specific time. The influx and outflux quantities were then derived using the *crosstab* method, after which they were resampled on a minute basis. The resulting dataset, which hence contains the influx and outflux quantities per minute, was saved as `fullParkingData.csv`. The simplified Python code is shown in Listing 1.

Similar to the parking transaction data, the traffic flow data was combined from multiple `.csv` files into a single DataFrame. Unlike the parking data, every row in the dataset represents one measurement with a fixed interval of one minute. The data consisted of many obsolete columns with erroneous values, which had to be handled first. The column `dataError` is of Boolean type and denotes whether a measurement is valid or invalid. Therefore, the `avgVehicleFlow` (which denotes the traffic flow) was set to NaN where `dataError == True`, such that these erroneous measurements could later be filled

```
# Set erroneous values to NaN, to interpolate missing series
df.loc[df['dataError'] == 1, 'avgVehicleFlow'] = np.nan

# Filter the rows with only the number of passenger cars
df = df[df['index'] == "1001A"]

# Select only the total traffic intensity and the required columns
df = df[['measurementSiteReference', 'periodStart', 'avgVehicleFlow']]
df.columns = ['location', 'timestamp', 'flow']

# Make a separate column for every measurement location
df = df.pivot_table(index='timestamp', columns='location', values='flow')

# Apply filtering on the signal
B, A = signal.butter(2, 0.05)
df = signal.filtfilt(B, A, df)

# Save output as a combined csv file
df.to_csv('fullTrafficData.csv')
```

Listing 2: Cleaning process of traffic flow data

by interpolation. Subsequently, only the measurements concerning passenger cars had to be selected. Since the NDW documentation [40] specifies that vehicles shorter than 5.6 meters are characterized by `index == 1001A`, this was used to query the dataset. After that, the relevant columns `periodStart`, `measurementSiteReference` and `avgVehicleFlow` were kept while the other (obsolete) columns were dropped from the DataFrame. Pivoting was then applied to assign a separate column to every measurement location. Since traffic flows are sensitive to randomness and high variance, smoothing was applied. For this purpose, a 2nd order low-pass Butterworth filter (with a cutoff frequency of 0.05) was applied. This method was selected in favour of a regular rolling mean, mainly since the rolling mean introduces a lag which will be problematic when real-time data sources are used. Ultimately, the resulting dataset was saved as `fullTrafficData.csv`. The simplified Python code of the full process is shown in Listing 2.

Relatively speaking, the historical dataset from KNMI required less complex operations. Firstly, as visible the hourly data was loaded into a Pandas DataFrame from a single comprehensive `.txt` file. As a consequence of the formatting of the file, some empty and undefined rows had to be removed from the dataset. Afterwards, the `timestamp` column was converted into a Pandas *datetime* type which makes it possible to perform temporal operations (e.g. interpolating, extrapolating and resampling) on the data. Equivalent to the parking and traffic datasets, the weather data is then also resampled on a minute basis. Since the dataset was originally provided in an hourly resolution, the *forward-fill* method (propagating the last valid observation forward) was used to fill the missing values which emerged after up-

```
# Remove all redundant entries in the txt file
df.dropna(inplace=True)

# Convert the hour and minute columns to the valid Pandas timestamp unit
df[0] = df[1].astype(int).map(str) + df[2].astype(int).map(str)
df[0] = pd.to_datetime(df[0], format='%Y%m%d%H')

# Rename columns for more overview
df.columns = ['timestamp', 'temp', 'rain']

# Upsample by forward-filling values between hours
df = df.resample('T').fillna("ffill")

# Save output as a combined csv file
df.to_csv('fullWeatherData.csv')
```

Listing 3: Cleaning process of weather data

sampling. The simplified Python code of the process is shown in Listing 3.

3.3.2 Establishing the final dataset

The three resulting datasets `fullParkingData.csv`, `fullTrafficData.csv` and `fullWeatherData.csv` were imported into three Pandas DataFrame structures, respectively. With `df_parking` as the base dataset, the columns of the other two DataFrames were appended using the *merge* operation. All datasets consisted of a mutual timestamp column (with *%Y-%M-%D HH:MM* format) which was used as the pivot to perform this operation.

```
def to_cyclic_time(date):
    hour, minute = date.strftime("%H").astype(int),
                   date.strftime("%M").astype(int)
    frac_time = hour + minute/60.0

    x, y = np.sin(2. * np.pi * frac_time/24.),
           np.cos(2. * np.pi * frac_time/24.)
    z = date.weekday

    return x, y, z
```

Listing 4: Function to convert a timestamp to three (cyclic) attributes

Because temporal variables are cyclic (for instance, days and hours are recurring patterns), engineering the optimal time features requires some extra attention. The conventional method to express time would be to assign a decimal number from 0 to 24 (for instance, 20:30 would be expressed as `20.5`) to every timestamp. However, this encoding would suggest that there exists a maximum difference between 23:59 and 00:00 (i.e. `23.98 - 0 = 23.98`), even though the actual difference

is just one minute. A solution was found by modelling the hour and minute as two separate input variables, which can be interpreted as the x- and y-component of a unit circle (or the hour and minute hand of a clock). The position is determined using trigonometric functions, i.e. a sine for the hour component and a cosine for the minute component. This will make it easier for the predictive model to evolve an understanding of the cyclic patterns and seasonalities within the dependent variables. Based on the timestamp, the weekday was determined and converted to an ordinal encoding to make it interpretable for the model. The conversion into cyclic time and weekday features was implemented in practice using the `to_cyclic_time` method, as visible in Listing 4. This function is then called from the main Python script, of which a simplified version is displayed in Listing 5.

```
# Merge the two datasets on the intensity data
df = pd.merge(df_intensity, df_parking, how='inner').fillna(0)
df = pd.merge(df, df_weather, how='inner')

# Compute occupancy from cumulative sum of in- and outflux
df['occup'] = (df['in'] - df['out']).cumsum()

# Normalize the occupancy
scaler = MinMaxScaler(feature_range=(0, 100))
df['occup'] = scaler.fit_transform(df[['occup']])

# Resample to a custom time interval
df = df.resample('T').last()

# Convert timestamp to cyclic temporal features
df['hour'], df['min'], df['weekday'] = to_cyclic_time(df.index)
```

Listing 5: Process of merging and refining the final dataset

After merging the datasets and engineering the temporal features, the occupancy rate was computed and appended to the dataset. The cumulative sum of the difference between influx (the `in` column) and outflux (the `out` column) was used to compute the number of occupied parking spaces, after which the result was assigned to a new column `occup`. Based on the total capacity of the Centraal garage (i.e. 1050 parking spaces), a *MinMaxScaler* was then applied to convert these values into a percentage.

A lookback window was composed to provide the model with knowledge about the recent past. Concerning the occupancy rate, this was achieved by shifting the `occup` column by a multiple of 11 units (since the real-time occupancy is updated every eleven minutes) and creating new columns with the shifted values. A window of 60 minutes was chosen, which therefore results in five new time-shifted columns. Concerning the traffic flows, a similar approach was taken: for every measurement location (i.e. starting with the `RWS01_MONICA_00D00C12BC0A10200005` column) a lookback window

of 30 minutes was created, based on the knowledge that all locations are (in the worst case) 15-30 minutes driving from the Centraal garage. By applying a 10-minute rolling sum on every traffic flow column, three aggregated time windows (i.e. 0-10 minutes ago, 10-20 minutes ago and 20-30 minutes ago) were established for every measurement location. This way, randomness and high variance of the real-time traffic measurements will have significantly less impact on the quality of prediction results. The lookback windows were realized according to Listing 6, and a comprehensive list of resulting variables is visible in Appendix A.

```
# Process all the traffic data here
for column in df.columns['RWS01_MONICA_00D00C12BC0A10200005':]:
    df[column] = df[column].rolling(10).sum()

    for x in np.arange(0, 30, 10):
        df[x + "-" + x+10 + "_" + column] = df[column].shift(x)

# Add previous occupancy rates to the dataset
for x in np.arange(0, 60, 11):
    df['occup_prev' + str(x)] = df['occup'].shift(x)
```

Listing 6: Creating lookback windows for traffic flow and occupancy rate

To be completely certain that there are no duplicate entries, the dataset was once again resampled on a minute basis. The total dataset, which now contains 40 columns relating to all independent and dependent variables, was then saved as `totalData.csv`. The size of the resulting file is 123 MB, which is substantially smaller than the initial files originating from the data sources. This shows the impact of cleaning and restructuring the data. A sample of the final dataset is listed in Appendix A.

Note: the mutations of the dependent variables (i.e. the horizons to be predicted) have not yet been added to the final dataset. To retain maximum flexibility with these columns during the model development process, they were added just before feeding the training data to the models (as described in Section 3.4).

3.3.3 Splitting the dataset

To develop a predictive model, the total dataset should be divided into multiple subsets. This is a fundamental practice within machine learning - not just to develop the predictive model, but also to assess its performance and apply statistical reasoning [13]. In total, three subsets can be distinguished:

- The **training set** is used by the model to learn patterns from the data. It yields a preliminary model which makes near-expected predictions.
- A **validation set** is used to understand behaviour of the preliminary model and its generalizability on a previously unseen dataset. The validation set is therefore used to assess the effect of structure changes and hyperparameter tuning.
- The **testing set** is kept separate from the model until until the very end, in pursuance of obtaining a completely unbiased estimate of model performance. It provides a solid indication of the model performance in a real-world scenario.

The partitioning of the datasets is arbitrary, but literature provides some guidelines. In general, it is recommended to have a sufficiently large training set, such that the model possesses enough data to learn from. Train-test divisions of 75%/25% and 80%/20% are commonly used, depending on the size of the total dataset. The initial training set is then again partitioned into an actual training set and a validation set. Usually, test and validation sets are kept small in relation to the training set. [12] [13] [36] Note that cross-validation would not be effective within the context of this research, given the sequential nature of the input data (i.e. time series). Moreover, it would arguably be more effective to maintain the chronological order of data, such that the model's sensitivity to seasonal patterns will become more evident during the validation and testing phase.

Ultimately, given the fact that our dataset is considerably large, the total set was divided into 80% training data and 20% test data. The training subset was then again partitioned into 90% training data and 10% validation data (as visible in Figure 3.3). Considering that the total dataset contains 756,500 entries, the end result is a training set of `80% * 90% * 756500 = 544,680` entries, a validation set of `80% * 10% * 756,500 = 60,520` entries and a test set of `20% * 756500 = 151,300` entries.

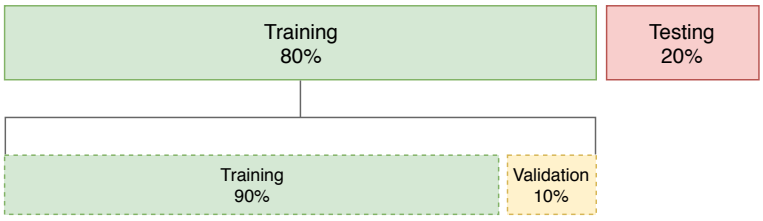


Figure 3.3: Partitioning of datasets for model development

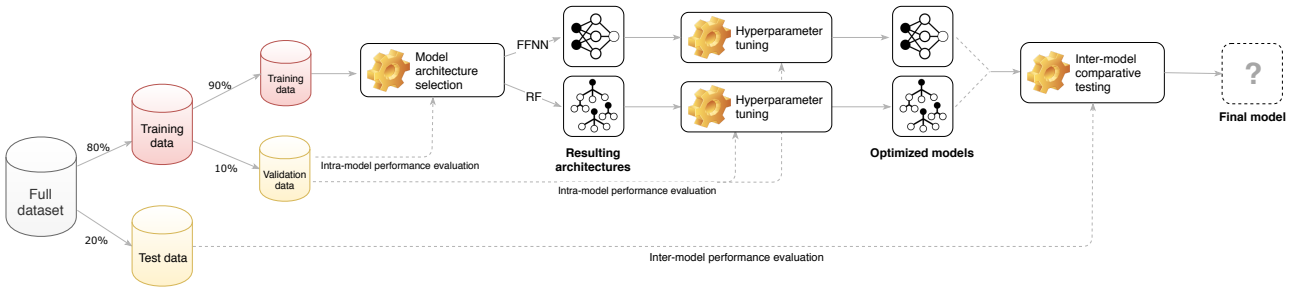


Figure 3.4: Complete process of model development and selection

3.4 MODEL DEVELOPMENT

After having composed and partitioned the final dataset, the process of training, validating and testing the models can commence. In this chapter, a systematic method is described which will be used to develop and optimize the two eventual models (i.e. both the *feed-forward neural network* and the *random forest*). The overall process (which is visualized in Figure 3.4) can be subdivided into three phases:

1. During **model architecture selection**, a systematic search is applied to establish the optimal internal architecture for both model types. This is achieved by repeatedly evaluating the model performance on the validation set. This phase results in two preliminary models.

2. Both models are then optimized during **hyperparameter tuning**. Relevant parameters of both model types are systematically tweaked, followed by repeated evaluation of the model performance on the validation set. This results in two candidate models, i.e. a feed-forward neural network and a random forest.

3. In the **inter-model comparative testing** phase, the candidate models of both types are assessed using the test set against a naive model. The methodology behind this phase is further described in Section 3.6.

The selection and optimization processes were executed using *Scikit-learn* and *Keras*. First, after initializing the training set into a DataFrame `df`, all predictive horizons of the dependent variables were added. This was done by shifting the `in`, `out` and `occup` columns in forward direction in steps of five minutes, after which the results were assigned to new columns corresponding to each output variable and the timeshift. The reason why the mutations of the dependent variables are added *after* creating the final dataset (and not *before*) is because it allows for more flexibility: before feeding the data to the model, additional data processing can be applied on the in/outflux

```
# Create 60 min. horizon + 30 min. buffer
for x in np.arange(5, 90, 5):
    df['occup' + str(x)] = df['occup'].shift(-x)
    df['in' + str(x)] = df['in'].shift(-x)
    df['out' + str(x)] = df['out'].shift(-x)

# The subset of all input/independent variables is formed here
X = df['weekday', 'hour', 'min', 'temp', 'rain', 'occup_prev',
       'occup_prev11', ..., '20_RWS01_MONICA_01D14503D400D0050009']

# The subset of all output/dependent variables is formed here
y_inout = df['in5', 'in10', ..., 'in90', 'out5', 'out10', ... 'out90']
y_occup = df['occup5', 'occup10', ... 'occup90']
```

Listing 7: Preparation of input set X and output sets y

and occupancy models individually. Even though the goal of this thesis is to predict up to 60 minutes ahead, the actual horizon was extended to 90 minutes in order to establish a *buffer*. Since the real-time occupancy of the Centraal garage is disclosed every 11 minutes, the 30-minute buffer assures that the real-time predictive system is always able to predict for 60 minutes ahead - even in the worst case where the last occupancy rate was received 10 minutes ago. Lastly, a common subset X was formed to accommodate all independent variables. Since in/outflux and occupancy rate are modeled separately, two subsets `y_inout` and `y_occup` were formed to accommodate the dependent variables. The implementation is visible in Listing 7.

To systematically find the optimal configuration for both model types, subsets of the relevant parameter spaces were defined. Subsequently, all possible combinations of those subsets were tested successively by compiling, training and validating a new model. The performance could then be compared between all parameter configurations. This process is called a *grid search* [13]. Listing 8 and Listing 9 shows an example of how both model types were built, compiled and fitted to the training set using the aforementioned Python libraries. The dashed lines indicate the places where parameters were defined. All FFNN configurations were trained under the same circumstances: for 200 epochs, using *Adam* as optimization algorithm. A `ModelCheckpoint` was used to save (i.e. create a checkpoint of) the model only when its performance had improved after an epoch, such that the best performing model is preserved after training. In Section 3.4.1.2, more will be explained about epochs and optimization strategies.

Since the training process of machine learning models is a time-consuming and computationally demanding task, it was determined that the first two phases were only focused on predicting the *occupancy rate*. Since the occupancy rate is directly determined by the *in-and outflux* (as explained in Section 2.2.1), the relative performance

```
# Build and fit FFNN using Keras
ffnn_model = Sequential()

# Adding new hidden layers determines the model architecture
ffnn_model.add(Dense(...., activation='relu'))
....

mc = ModelCheckpoint('best_ffnn_model.h5', monitor='val_loss',
                     mode='min', save_best_only=True, verbose=1)

ffnn_model.compile(loss='mse', ....)
ffnn_model.fit(X, y_occup, validation_split=0.10,
               epochs=200, verbose=2, callbacks=[mc])
```

Listing 8: Building a new FFNN occupancy model using Keras

```
# Build and fit the random forest using Scikit-learn
rf_model = RandomForestRegressor(....)
rf_model.fit(X, y_occup)
```

Listing 9: Building a new RF occupancy model using Scikit-learn

of the occupancy model is arguably a strong indicator for the performance of the in- and outflux model. As a consequence, the in- and outflux models were only developed after the first two phases (i.e. just before inter-model comparative testing) using the selected architectures and parameters.

3.4.1 Feed-forward neural network

3.4.1.1 *Architecture selection*

The model architecture of a feed-forward neural network was selected using two parameters: the *number of nodes* and the *number of hidden layers*. According to Goodfellow, Bengio and Courville [36], "the dimensionality of hidden layers determines the *width* of the model". In similar fashion, the *depth* of the model is determined by the number of hidden layers. Together, these parameters determine the internal structure of the model and therefore have fundamental impact on its functioning and performance.

A **2D grid search** was executed to find the optimal combination of the aforementioned parameters. For the number of nodes n in the network, a range from 10 to 100 was chosen, equally spaced by an interval of 10. This approach was mainly based on the commonly applied rule-of-thumb which specifies that the number of nodes in the hidden layers should be "between the input layer size and the output layer size" [45].

For the number of hidden layers m, a range of 1 to 10 was chosen. The nodes are divided equally across the hidden layers. In case of a

remainder after evenly distributing the neurons, the remaining neurons are added to the first hidden layer (i.e. after the input layer), based on the fact that the input layer of size 40 is significantly larger than the output layer of size 18. To illustrate, a configuration with $n = 50$ and $m = 4$ would result in the following composition (obviously excluding the input and output layer):

$$[14, 12, 12, 12]$$

The grid search can be regarded as a matrix, where every combination of n nodes distributed across m layers is used to develop a unique mutation of the feed-forward neural network. Afterwards, the loss $L_{n,m}$ (on the validation set) of every mutation is examined and compared in order to determine the optimal configuration. As previously explained in Section 2.2.3, the *mean squared error* was used as loss function. A representation of the parameter grid can be seen in Figure 3.5.

$$\begin{array}{c} \\ \\ 10 \\ 20 \\ 30 \\ \vdots \\ 100 \end{array} \begin{array}{c} \text{\# hidden layers} \\ \begin{array}{ccccc} 1 & 2 & 3 & \dots & 10 \end{array} \\ \begin{bmatrix} L_{10,1} & L_{10,2} & L_{10,3} & \dots & L_{10,10} \\ L_{20,1} & L_{20,2} & L_{20,3} & \dots & L_{20,10} \\ L_{30,1} & L_{30,2} & L_{30,3} & \dots & L_{30,10} \\ \vdots & \ddots & & & \vdots \\ L_{100,1} & L_{100,2} & L_{100,3} & \dots & L_{100,10} \end{bmatrix} \end{array} \text{\# neurons}$$

Figure 3.5: Matrix of grid search for FFNN architecture

3.4.1.2 *Hyperparameter tuning*

After having determined the optimal composition of hidden layers, the next phase is to perform hyperparameter tuning. In their book *Deep learning*, Goodfellow, Bengio and Courville state that "the learning rate is perhaps the most important hyperparameter. If you have time to tune only one hyperparameter, tune the learning rate." [36] The *learning rate* α was therefore chosen as the main hyperparameter to optimize during this phase. Once again, a grid search was executed to find the optimal value of α. The corresponding grid, which ranges from 0.01 to 0.00001 on a logarithmic scale, is defined as:

$$\alpha = [0.01, 0.001, 0.0001, 0.00001]$$

Gradient descent is the most conventional optimization strategy for neural networks: to find the minimum loss, the weights of the network are updated against the slope (or gradient) of the loss function at the current point [36]. *Adam*, the optimization algorithm which was

used to train the models in Keras (see Listing 8), is also based on this principle. Its main goal is to decrease the loss (which, in this context, is the mean squared error) with regard to the validation set, since this provides the best indication of whether underfitting or overfitting has taken place. The *learning rate* determines the size of the steps which are taken when 'descending the slope'. If the learning rate is too high, it will be more difficult to find the global minimum and the model might not converge at all. On the other hand, if the learning rate is too low, it will take very long for the model to converge [13]. A balance must therefore be established.

Hence, the *number of epochs* is also a crucial factor. An *epoch* is a single step in the training process of the neural network: it means that the whole training set has been passed through the network once and that the weights have been updated accordingly. The optimal learning rate was found by plotting the validation loss of each learning rate α (from the parameter grid) against the number of epochs. In case the α is too high, the validation loss decreases quickly during the first epochs, but the improvement then stagnates (i.e. the global minimum loss is never found) or even worsens. When the α is too low, the validation loss will decrease steadily but extremely slowly. The optimal α is therefore characterized by a steady and enduring decrease of the validation loss, which is neither too slow nor too abrupt.

3.4.2 Random forest

3.4.2.1 *Architecture selection*

Considering architecture and structural properties, random forests require a different approach than feed-forward neural networks. Random forests are rather simple to optimize, since there are relatively little parameters to modify. The main parameter which determines the architecture of the model is the *number of trees*. Generally speaking, maximizing the number of trees means that the predictive power is maximized as well [13]. However, the improvement decreases as the number of trees increases, i.e. at a certain point the benefit in predictive power from incorporating more trees will be lower than the cost in computing time for learning these additional trees, which poses a potential threat to a real-time system. A **1D grid search** was therefore executed to find the optimal balance between the number of trees and computational complexity.

$$
\begin{array}{c}
\text{\# trees} \\
\begin{array}{cccccccccc}
1 & 5 & 10 & 15 & 20 & 25 & 50 & 100 & 150 & 200 \\
\left[L_1 \right. & L_5 & L_{10} & L_{15} & L_{20} & L_{25} & L_{50} & L_{100} & L_{150} & \left. L_{200}\right]
\end{array}
\end{array}
$$

Figure 3.6: Array of grid search for RF architecture

As visible in Figure 3.6, a range between 1 and 200 was chosen for the number of trees n. Gradually increased spacing was used between the amounts, mainly since the loss will initially decrease quickly before reaching an equilibrium (i.e. asymptotic behaviour). The optimum is then located at the point where almost no improvement is taking place anymore. Again, the *mean squared error* was used as loss function L_n.

3.4.2.2 *Hyperparameter tuning*

After the optimal number of trees in the random forest has been identified, there are other hyperparameters of interest which could lead to an increase of performance. According to Koehrsen, the *maximum tree depth* (which sets a limit for the depth of each tree in the forest) and *maximum features* (the number of features to consider when looking for the best split) are two of the most important hyperparameters [46]. The tree depth determines the flexibility of the model: a deeper tree can fit more complicated functions. However, additional flexibility can also lead to overfitting. By evaluating the loss on the validation set, an optimal balance can be found. A **2D grid search** provided an accurate and thorough method to optimize both parameters. For the max. features the Scikit-learn library provides a fixed set of options which generally satisfy the optimization needs: *the number of features* (i.e. the 40 independent variable columns), the *square root of this number* as well as the *log base 2 of this number*. For the max. depth of the trees, a range of 1 to 30 was chosen.

$$
\begin{array}{c}
\\
\\
1 \\
2 \\
3 \\
\vdots \\
30
\end{array}
\begin{array}{c}
\text{max. features} \\
\begin{array}{ccc}
\#\,features & \sqrt{\#\,features} & \log_2(\#\,features)
\end{array} \\
\left[\begin{array}{ccc}
L_{1,n} & L_{1,sqrt} & L_{1,log2} \\
L_{2,n} & L_{2,sqrt} & L_{2,log2} \\
L_{3,n} & L_{3,sqrt} & L_{3,log2} \\
\vdots & \vdots & \vdots \\
L_{30,n} & L_{30,sqrt} & L_{30,log2}
\end{array}\right]
\end{array}
\text{max. depth}
$$

Figure 3.7: Grid search matrix for RF hyperparameter tuning

The grid search can be regarded as a matrix, where all combinations of max. depth d and max. features f are used to build and validate a model, after which the loss $L_{d,f}$ (i.e. from the epoch with the lowest MSE on the validation set) of the corresponding configuration is evaluated. The overall parameter grid is visible in Figure 3.7. Ultimately, the combination with the lowest $L_{d,f}$ is selected as the optimal hyperparameter configuration. Together with the previously selected model architecture, i.e. the optimal number of trees (see Section 3.4.2.1, this results in the final RF model which is ready to compete against the final FFNN model.

3.5 COMPILING AND FITTING FINAL MODELS

After the optimization phase, the final models were compiled and trained definitively using the identified configurations. Along with the occupancy rate models, this entails that the in- and outflux models were now also developed. For the final round, the feed-forward neural network was trained under different circumstances: a training duration of 2000 epochs was chosen instead of 200. In combination with a `ModelCheckpoint`, a larger number of epochs facilitates a further decrease of validation loss.

Ultimately, the final versions of the models were saved as the following files:

- `final_ffnn_occup.h5` (the FFNN model which predicts the *occupancy rate*)
- `final_ffnn_inout.h5` (the FFNN model which predicts the *influx* and *outflux*)
- `final_rf_occup.pckl` (the RF model which predicts the *occupancy rate*)
- `final_rf_inout.pckl` (the RF model which predicts the *influx* and *outflux*)

Note that different file formats are used - this is because Keras exports models as *Hierarchical Data Format* (`.h5`) while Scikit-learn uses the *pickle* (`.pckl`) format by default.

3.6 INTER-MODEL COMPARATIVE TESTING

Now that the final candidate models were developed and exported, their performance should be assessed and compared. In order to decide upon the optimal model for the predictive system, both the quality and efficiency of predictions were assessed. As explained in Section 3.3.3, the test set was kept separate from the training and validation datasets (both of which were previously used to develop the individual models). Hence it provides a completely unbiased estimation of how the models perform. Regarding the actual performance assessment, the literature study in Section 2.2 has shown that a combination of *mean squared error* (MSE), *mean absolute error* (MAE) and *mean absolute scaled error* (MASE) would provide the best insights. MSE and MAE provide a comprehensible and precise way of understanding the magnitude and distribution of the model's errors using a natural, unambiguous scale. MASE compares the model's MAE to that of a naive benchmark model, which makes it robust to scaling differences. This accommodates the comparison between in/outflux

and occupancy rate models. Moreover, it demonstrates the added value of each model with regard to a naive model.

3.6.1 Naive prediction benchmark

In order to use MASE, a naive benchmark model must be defined first. According to Gilliland, a naive model should be "simple to calculate" and easily implementable such that no computational power is needed whatsoever [47]. Two commonly used naive models are the *random walk* and the *seasonal random walk*:

- The **random walk** model uses the last known observation to predict the future values. To illustrate, the last known occupancy rate from the Centraal garage will be used as prediction for 5 minutes ahead, and also 90 minutes ahead.
- The **seasonal random walk** model incorporates seasonal and temporal patterns in order to make predictions. For instance, the influx from one year ago would be used to predict the influx for the upcoming minute.

Considering that this thesis aims to predict for a multitude of horizons simultaneously (i.e. a multi-output approach, ranging from 5 to 90 minutes ahead), the seasonal random walk was chosen as the most suitable naive model. In case of the 'regular' random walk, the same value would namely be predicted for every horizon. As a result, the accuracy of the 1-minute-ahead prediction would likely be excellent, but performance would drop drastically as soon as the horizon moves further away. With this in mind, the seasonal random walk was regarded to be a more suitable benchmark model.

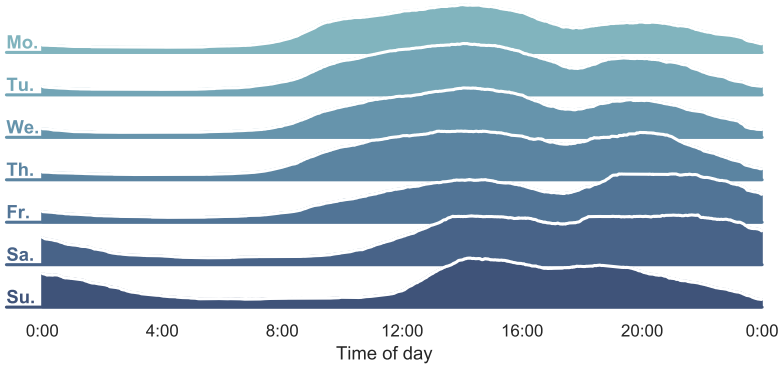


Figure 3.8: Daily and weekly patterns of the occupancy rate

An important consideration for the seasonal random walk is choosing a suitable seasonality. Parking data is obviously sensitive to temporal patterns: for instance during morning rush hour, the influx

tends to be very high when compared to other times of day. Similarly, the influx tends to increase dramatically during the morning rush hour. In Figure 3.8, the occupancy rate during a regular week (from 22-10-2018 until 28-10-2018) is visualized. Not only daily seasonalities (like the ones described earlier) are clearly visible, but also weekly seasonalities such as the difference between weekdays and weekend. All things considered, a period of 7 days (= 10.080 minutes) was chosen to be the most suitable seasonality since it covers both the daily and weekly patterns of the dependent variables. The selected benchmark model is therefore a seasonal random walk which uses the historical data of 10.075 minutes ago till 9.990 minutes ago as predictions from 5 to 90 minutes ahead.

3.6.2 Quality of predictions

Regarding the quality of predictions, both feed-forward neural network and random forest were assessed using the test set. After feeding the independent variables of the test set to the input of both models, the corresponding output predictions were generated and stored in a DataFrame `y_predict`. By comparing them to the actual values `y_true` of the dependent variables, the metrics MAE, MSE and MASE were computed. Scikit-learn contains several built-in functions which can be used to implement these metrics. The code snippet in Listing 10 shows abstractly how this was done.

```
from sklearn.metrics import mean_absolute_error, mean_squared_error

mse = mean_squared_error(y_true, y_predict)
mae = mean_absolute_error(y_true, y_predict)

mase = mae / mean_absolute_error(y_true, y_naive)
```

Listing 10: Obtaining test metrics for both model types

The assessment of predictive quality was commenced by a listing of the MSE, MAE and MASE values per candidate model for every dependent variable. Accordingly, three tables (`influx`, `outflux` and `occupancy rate`) with two columns (`FFNN` and `RF`) and three rows (`MSE`, `MAE` and `MASE`) were created. Before performing a more detailed assessment of errors, the tables provide a comprehensive understanding of the overall performance.

MASE was previously defined as the principal metric for inter-model comparison. Hence, the MASE of the influx, outflux and occupancy rate models was visualized using Matplotlib. This resulted in a three-fold line plot with the predictive horizon (i.e. a range of 0 to 90 minutes ahead) on the x-axis and the MASE on the y-axis. By definition of the MASE metric, the naive model always satisfies $MASE = 1$. Therefore a dashed horizontal line was placed on each subplot at

$y = 1$, such that the predictive performance of the influx, outflux and occupancy can be visually compared with that of the naive model. Instinctively, the candidate model with the lowest values of MASE is the best. This visualization provides an efficient method to assess the performance of the FFNN and RF models on all three dependent variables at the same time. Also, it shows the development of MASE as soon as the predictive horizon moves further away - which is a meaningful aspect to take into consideration. For instance, a model which initially makes many mistakes but starts to perform robustly after 30 minutes might be preferable above a model which predicts perfectly for the first 15 minutes but then suddenly weakens.

In order to obtain more insight into the distribution of the individual errors from which the MASE was computed, a violin plot was created. A violin plot is similar to a conventional box plot, but has the additional benefit of displaying a rotated density plot on both sides. This way, specific characteristics in the distribution of the errors can be perceived and interpreted. For instance, a model with many outliers but a relatively low MASE might be regarded as inferior to a model with a higher MASE and a compact distribution of errors. In order to compare the influx, outflux and occupancy rate, all individual absolute errors were scaled using the MAE of the naive model. Note that the mean of these individual error components is therefore synonymous to the MASE.

With regard to the quality of predictions, the tables and plots facilitate the final decision between both candidate models.

3.6.3 Efficiency of predictions

For a system which aims to produce a consistent stream of predictions in real-time, it is crucial to consider the predictive efficiency. In order to compare the efficiency of both model types, the *prediction time* (in seconds) was measured. Obviously, the lower the prediction time, the higher the efficiency. After recording the start time using the *time* function in Python, the predictions were made using the test set (i.e. `X_test`) as input. As soon as the output predictions were established, the end time was recorded as well. The prediction time was then calculated by subtracting the start time from the end time. This process (as abstractly displayed in Listing 11) was executed 100 times per model in order to establish an adequate sample size.

```
start = time.time()
model.predict(X_test)
end = time.time()
```

Listing 11: Measuring prediction times of a model

After the prediction times were collected for all models, the mean was computed for both the feed-forward neural network and the random forest. The resulting numbers were then compared in order to discover which model type is most efficient.

3.7 REAL-TIME PREDICTIVE SYSTEM

Even though the optimal machine learning model has been developed and tested in conformity with Section 3.4, the main goal is to implement this model into a comprehensive system which can continuously generate predictions based on a real-time data feed.

3.7.1 System architecture design

Designing an appropriate system architecture requires a careful process where all elements of the chain are taken into account: gathering the data, generating predictions, evaluating the predictive performance and communicating information to end users. Figure 3.9 shows the architecture design which was ultimately selected.

First and foremost, the machine learning model (i.e. the central element of the system) should be provided relevant data to satisfy all its input variables. This can be translated to the *data collection* component of the system architecture, in which the three real-time data sources (as identified in Section 3.2.2) are queried on a recurring basis. In the same process, the input data are shaped and prepared, after which they can be saved into a database of input data. This database is a part of the comprehensive system database.

Every minute, the machine learning model retrieves the recent input data from the database and uses this to generate a set of pre-

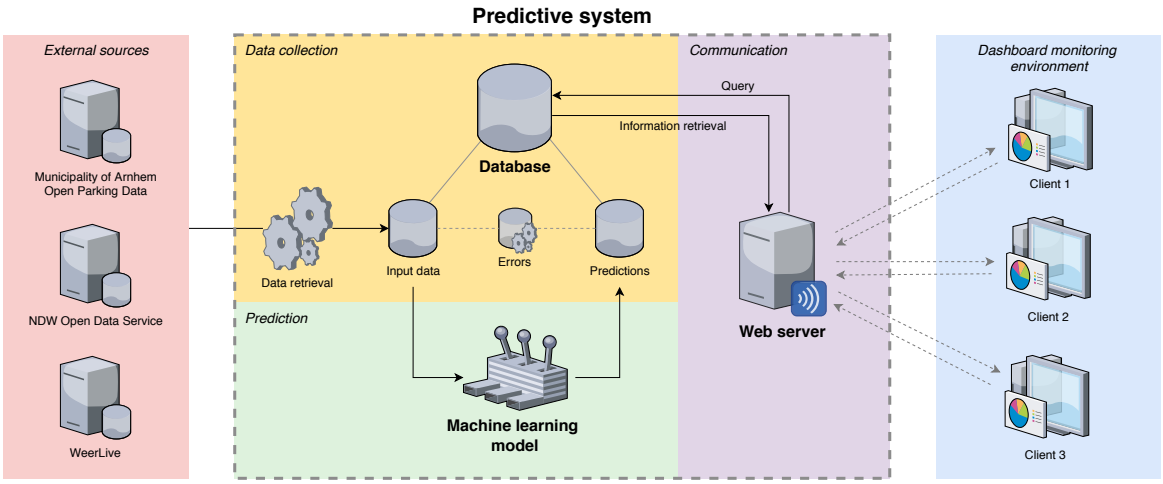


Figure 3.9: Resulting system architecture

dictions for the upcoming 60 minutes. These are then fed back to a database which stores all predictions that the model has generated over time. Together, these processes belong to the *prediction* component of the system, and are thus responsible for the core predictive functionality of the system.

As soon as a batch of input data is received, the errors can be computed for all predictive horizons by comparing the incoming occupancy rate with the predictions of the past hour. Hence, this process utilizes both the input database and the prediction database, and finally saves the errors into another database. This results in a self-sustaining collection of errors per predictive horizon, from which the system performance can be evaluated.

A web server forms the link between the system core and the end user. Hence, it belongs to the *communication* component of the system architecture. After establishing a connection with a client, the server will push the real-time data to the client as soon as a new prediction is available. This process ensures that the client receives the information in real-time, without any delay. The dashboard application runs in the web browser of the client. Hence, this is the *front-end* of the system where the incoming data stream from the server is actually visualized and displayed to the end user.

3.7.2 Back-end

The core functionality of the system revolves around gathering real-time data and predicting the influx, outflux and occupancy rate with this input data. The corresponding processes are invisible to the end user, and can therefore be regarded as the *back-end* of the system.

The Python `multiprocessing` library was used to run all processes simultaneously from one Python program. This program is called `mainProcess.py`. All processes have access to the SQLite database which is stored locally on the system machine as `database.sqlite`. Six tables (as visible in Table 3.2) were added to store all relevant incoming data and the output predictions for every timestep ahead. A *reference* table was added with the average occupancy rate per weekday and time of day. This facilitates comparison of the real-time predictions with the historical averages of the same point in time.

Table	Columns
occup	*time, occup, err_5, err_10, err_15, ..., err_60*
weather	*time, temp, rain*
traffic	*time, RWS01_MONICA_00D00C12BC0A10200005, ...*
pred_inoutflux	*time, pred_in_10, pred_in_20, ..., pred_in_60, pred_out_10, pred_out_20, ..., pred_out_60*
pred_occup	*time, pred_5, pred_10, pred_15, ..., pred_60*
reference	*time, weekday, occup*

Table 3.2: Composition of system database

3.7.2.1 *Data retrieval*

The input data was retrieved in real-time using three separate processes, i.e. for the occupancy rate, weather data and traffic data, respectively. This approach was mainly chosen to maintain a higher computational efficiency, but also to make the system less susceptible to a malfunction caused by a single data source (e.g. a connection error). In this case, the other processes would continue to run without being impaired or delayed. All processes implement a `while(True)` loop combined with a `time.sleep()` function in order to retrieve the data repeatedly on the desired interval.

In the first place, an important process is the retrieval of real-time occupancy rate data. This process is named `get_occupancy()` and is continuously executed with an interval of 60 seconds in order to keep measurement delays as low as possible. As mentioned in Section 3.2.2, the data is queried in JSON format from the parking data portal of the Municipality of Arnhem [42]. Using the `requests` library, the response from the URL is loaded, after which the page is interpreted as a JSON file and parsed to a Python `dict` structure. The supplier distributes the data in a specific format, and therefore the relevant information should be filtered from the original JSON structure. Hence, the *parkingCapacity* and *vacantSpaces* values were used to compute the current occupancy rate, and the *lastUpdated* value was used to record the time at which the current measurement was taken. Using the retrieved occupancy rate (i.e. `occup`), the errors are then computed. This is done using the `compute_errors()` function which compares every prediction of the past 60 minutes to the current occupancy measurement. By taking the absolute value of the differences between the real-time feed and the predictions, the error magnitude can be obtained for every predictive horizon. The timestamp, current measurement and all errors are then added to the *occup* table using an `INSERT` query. When not enough predictions have been made in the past hour (e.g. when the system has just been started), the errors are filled with `NaN` values. A simplified version of the `get_occupancy()` process code is visible in Listing 12.

```
response = requests.get("http://opd.it-t.nl/Data/parkingdata/v1/arnhem/...")
data = json.loads(response.text)['parkingFacilityDynamicInformation']
                               ['facilityActualStatus']

occup = (data['parkingCapacity'] - data['vacantSpaces'])
        / data['parkingCapacity'] * 100

errors = compute_errors(occup)

c.execute("INSERT INTO occup VALUES (?,?,?,?,?,?,?,?,?,?,?,?,?,?)",
          [data['lastUpdated'], occup] + errors)
```

Listing 12: Retrieval of real-time occupancy rate

A similar method was used to retrieve the weather data in real-time. This functionality was implemented using the `get_weather()` process. The weather data is provided by Weerlive in JSON format [44]. Only the *temp* and *rain* values are of interest in this case, so only these two values are queried and inserted into the `weather` table of the database. Since the weather conditions are less prone to rapid changes, the process is executed once every 15 minutes to relieve computational complexity.

Lastly, the traffic data is retrieved using the `get_traffic` process. This process is executed every minute, based on the fact that the traffic data is fed to the model using a rolling mean. Since traffic flows are characterized by high variance (as explained in Section 3.3.2), a larger number of samples is beneficial to the consistency of these inputs. The traffic data is retrieved from the NDW Open Data portal [43] in a compressed XML file. Hence, the `GZip` library is used to decompress the file first, after which the resulting XML file is parsed into the more practicable `dict` format using the `xmltodict` library. The structure of the data is fairly complicated, which results in a large number of selections which have to be performed before the raw traffic flow integer can be obtained. Using a double for-loop, the traffic flow amounts are assigned to the corresponding measurement locations. The process is concluded by an `INSERT` query such that the current timestamp and traffic flows are appended to the *traffic* table of the database. A simplified version of the process code is visible in Listing 13.

```
with gzip.GzipFile(fileobj=f) as xml_file:
    data = xmltodict.parse(xml_file)
    data = data['SOAP:Envelope']['SOAP:Body']['d2LogicalModel']
               ['payloadPublication']

    meas = {
        'RWS01_MONICA_00D00C12BC0A10200005': 0,
        'RWS01_MONICA_00D00C15003210200009': 0,
        ....
    }

    for element in data['siteMeasurements']:
        for location in meas:
            if element['measurementSiteReference']['@id'] == location:
                meas[location] = element['measuredValue'][0]['basicData']
                                        ['vehicleFlow']['vehicleFlowRate']
                break

    c.execute("INSERT INTO traffic VALUES (?,?,?,?,?,?,?,?,?,?,?)",
              [t_now] + list(meas.values()))
```

Listing 13: Retrieval of real-time traffic flows

3.7.2.2 *Generating predictions*

The `predict()` process is used to generate influx, outflux and occupancy rate predictions in real-time. Both the in/outflux model and the occupancy rate model are first loaded into memory, either from a `h5` or `pckl` file (depending on the chosen model type of Section 3.6). Subsequently, all input data has to be collected and processed before it can be fed to the models. This is done according to the following steps:

1. Loading the last five occupancy rate measurements from the *occup* table, which together compose the lookback window which is used as a direct input for the models.
2. Loading the traffic data from the *traffic* table, and applying a rolling mean with a window of 10 minutes, after which every tenth value is selected. This results in an consistent representation of the traffic flows at each location for 0-10 minutes ago, 10-20 minutes ago and 20-30 minutes ago. The resulting array of values maps one-to-one to the traffic flow inputs of the models.
3. Selecting the last known temperature and rain values from the *weather* table. These values also map one-to-one to the weather inputs of both models.
4. The time and weekday values are computed by obtaining the time of the last occupancy rate measurement and feeding this to the `to_cyclic_time()` function (as described in Listing 4).
5. All input data are fed to the model, after which the model will deliver an output of raw predictions for a horizon of 90 minutes.

Since there is a delay of maximally 11 minutes for obtaining a fresh and up-to-date set of predictions (i.e. the interval at which new occupancy rate predictions become available), the 30-minute buffer can be used to maintain a steady stream of predictions. In order to adapt the raw predictions to the current horizon of 60 minutes ahead, the set of predictions is first resampled and interpolated on a minute-basis. *Polynomial interpolation* (of the 2nd order) was used to fit a realistic and smooth curve through the sample points. The result is a continuous series of predictions, from which a 60-minute subset (starting at the current timestamp) can be selected. Then, by selecting every 5th element of the resulting series, the actual predictions for the upcoming hour are obtained. Ultimately, these are inserted into the *pred_inoutflux* and *pred_occup* tables, respectively. The code snippet of Listing 14 shows the interpolation and selection steps with regard to the occupancy rate.

```
future_range = pd.date_range(t_last, periods=19, freq='5T', closed='right')
df_occup = pd.DataFrame({'occup': y_occup}).set_index(future_range)
df_occup = df_occup.resample('T').mean().interpolate(method='polynomial',
           order=2).loc[t_now + 5:t_now + 65][::5]

c.execute('INSERT INTO pred_occup VALUES (?,?,?,?,?,?,?,?,?,?,?,?,?)',
          [t_now] + df_occup['occup'].tolist())
```

Listing 14: Interpolating and selecting predictions for upcoming hour

3.7.2.3 *Running the server*

As explained in Section 3.7.1, the server has the task of communicating predictions to the end users. It therefore serves as the bridge between the back-end and front-end of the system. The server is established using the `Flask` library using the `serverHandler()` process. Also, since new predictions are generated every minute, it is important that the users of the front-end application can observe the changes as they occur. The `SocketIO` library provides a way to achieve real-time bi-directional communication between clients and the server.

For every client that connects to the server (i.e. a user who tries to open the application using the web browser), a new thread is started. This thread then executes a few steps in order to load the relevant data and emit it to the client's browser. These steps can be described as follows:

1. Loading the last batch of predictions from the *pred_occup* and *pred_inout* tables of the central database.
2. Prepare and/or enrich the predictions for communication towards client
   a) Regarding the occupancy rate, the historical measurements of the past hour (obtained from the *occup* table) are appended to the predictions. The combined array is then resampled and interpolated (i.e. polynomial interpolation of the 2nd order) to obtain an equally spaced range of occupancy rates. This enriches the prediction data, such that the end user will obtain a comprehensive overview of the overall development of the occupancy rate over time.
   b) Regarding the in- and outflux, the predictions (which are output by the model as a combined array) are split in half such that the influx and outflux are kept separate. Also, besides communicating the full arrays of influx and outflux predictions, the flows are also aggregated for the full 60 minutes ahead. This gives the end user a quick impression of the magnitude and direction of the flows.

3. Computing the error metrics for all predictive horizons from the individual error values in the *occup* table. The decision was made to compute the MAE (i.e. taking the mean of all individual errors) as well as the MSE (i.e. squaring the individual errors and then taking the mean) for every predictive horizon. This enables the end user to identify patterns, developments and anomalies in system performance.

4. Computing a set of summary statistics using the *occup* table which facilitates an at-a-glance overview of system performance. The MAE was chosen to be the main metric, since its natural scale makes it easier to understand for end users. In addition, dividing the naive model's MAE by that of the system will tell how many times 'better' the system performs than the previously defined naive model. Note that this this metric essentially identical to the inverse of the MASE, but will appeal more to the imagination and comprehension of the end users.

Having obtained all relevant data and metrics, the next step is to realize the communication of this data towards the front-end application. Multiple `socket.emit()` calls are executed to to produce events which can be perceived by the JavaScript in the client's browser. An *emit* contains the data which should be communicated from the server to client, or vice versa. In total, two *emit* events are created: the first one contains the set of new predictions. The decision was made to group all predictions in one event, mainly to increase the efficiency of communication: after all, the predictions are generated using the same interval of one minute. The second emit event contains the error values, as well as the summary metrics. An abstract representation of how the data is emitted is visible in Listing 15.

```
socket.emit('new_pred', [pred_occ, pred_in, pred_out, aggregated_inout],
                        namespace='/predict')
socket.emit('new_metrics', [errors, summary], namespace='/predict')
```

Listing 15: Emitting the relevant data to the client

3.7.3 Front-end

The back-end processes enable the system to produce predictions and communicate them via a web server. However, raw predictions will not make a lot of sense to the stakeholders which this system is aimed at. For the system to communicate important insights effectively to those stakeholders, good visualizations of the system outputs are thus essential. According to Knaflic, visualizing and communicating data "is key to turning it into information that can be used to drive better decision making" [48].

3.7.3.1 *Context*

Earlier in Section 1.1, the potential users of the system were identified. They can be defined as follows:

- **Public road authorities**, who would use the predictive system to enhance their information supplies and empower better decision-making within their *traffic management centers*
- **Private mobility service providers**, who aim to use the predictions in their own service which is then provided to the end users (e.g. drivers of vehicles)

Especially the public road authorities (i.e. road operators, planners and traffic management) would benefit from visualizing the predictions, considering that they would use the system as an instrument to enforce dynamic traffic management measures. As opposed to the private mobility service providers, who will mainly acquire and contextualize the data before propagating it to their customers, they can therefore be regarded as the main target group for visualization and communication of the predictions.

There are several essential aspects of information which are needed by traffic management centers to facilitate efficient decision-making. First, the actual predictions should be communicated clearly and within their respective context. For instance, when sudden changes in occupancy rate, influx or outflux are predicted, the visualizations should clearly communicate this, such that reliable and rational decisions can be made on short notice. It is therefore crucial that the predictions are not displayed independently, but that the changes and developments are visualized over time. Preferably, the historical measurements and predictions are fused into a continuous time series. This will provide a complete context of time, which is crucial for traffic management to anticipate on the dynamic behaviour of the parking flows.

To ensure the transparency of the system, information about the system's performance should also be clearly communicated to the stakeholders. Such performance metrics (e.g. the aggregation and distribution of the absolute or squared errors) can provide an indication of how good the model predicts, and also shows whether the current performance is better or worse than usual. Therefore it can be used as a measure for reliability and stability: another useful instrument for facilitating well-grounded decisions.

3.7.3.2 *Visualizations*

First of all, a **dashboard gauge** was implemented to give stakeholders an instant overview of the current occupancy of the garage. This was done using the *JustGage* library, which provides the functionality of creating and customizing a gauge and updating it dynamically. A

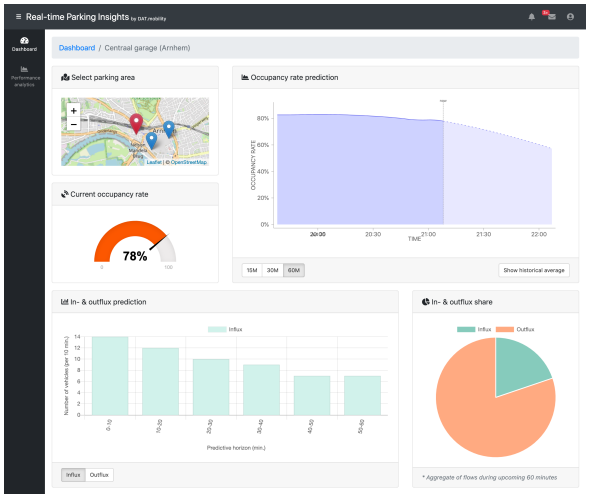
range of 0 to 100% was chosen since the gauge displays the percentage of occupancy in the garage. By making the gauge color-coded, i.e. a scale from green to red corresponding to the occupancy rate, the users of the dashboard can immediately observe the current situation in the garage. The gauge therefore provides a clear context and facilitates the interpretation of the other visualizations.

To visualize the occupancy rate over time (including the predictions), a **line graph** was created. This was done using the *Metricsgraphics* library since it provided the necessary tools to dynamically visualize time series. By placing a marker at the current point in time (annotated with *'now'*) and modifying the line style and area color in the CSS stylesheet, the predictions can be clearly distinguished from the historical observations.

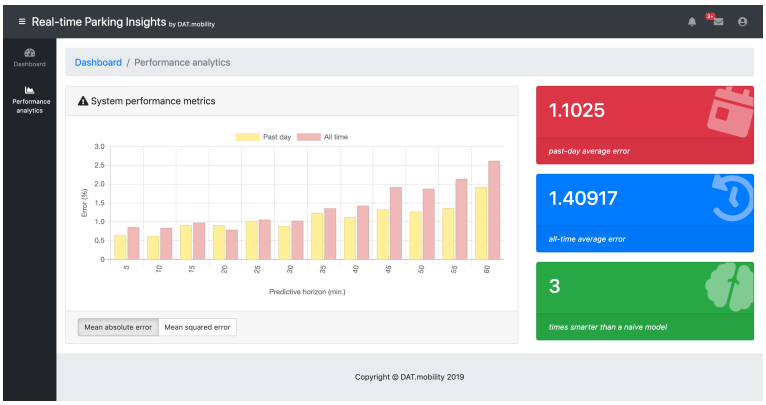
In contrast to the continuous nature of the occupancy rate time series, the in- and outflux are discrete (i.e. divided in 10-minute aggregation windows) which makes a line graph unsuitable. As a result, a **bar chart** was chosen for visualizing both the influx and outflux, with every bar representing one time window of 10 minutes. This results in six bars representing the predictions for up to 60 minutes ahead. The chart was realized using the *ChartJS* library, which facilitates user interaction with the graph (e.g. using a tooltip). A button was added such that the user can interactively switch between the influx or outflux. To obtain a single overview of the expected distribution of the influx and outflux in the upcoming hour (i.e. aggregation over 60 minutes), a **pie chart** was added. A tooltip was incorporated such that users can read the exact values of the flows when hovering over the chart.

The system performance metrics were visualized using a **grouped bar chart**, where every predictive horizon (in steps of 5 minutes) is represented by a group of bars. Each group consists of two bars: the left bar represents the mean error of the last 24 hours, while the right bar represents the mean error of all time. Again, this chart was created using the *ChartJS* library. An interactive toggle enables users to switch between the mean squared error (MSE) and the mean absolute error (MAE), which thus updates the magnitude of the bars. Ultimately, an overview of summary metrics was also added using multiple boxes which were stylized with a suitable icon. The metrics which were implemented are: the MAE of the past day, the MAE of all time and the factor by which the selected model performs better than the naive model. The latter metric is dynamically computed by dividing the MAE of the naive model by that of the selected model.

Note: a complete overview of the visualizations can be found in Appendix B.



(a) 'Parking monitoring' page



(b) 'Performance analytics' page

Figure 3.10: Dashboard monitoring environment

3.7.3.3 *Dashboard*

To communicate all the aforementioned information to the relevant stakeholders, the visualizations should be presented in a clear and insightful way. Stephen Few argues that a *dashboard* is a "unique and powerful means" to achieve this. A dashboard provides at-a-glance views of several metrics and indicators regarding information which is relevant to a particular objective [49]. The visualizations which were described in Section 3.7.3.2 were thus jointly implemented into a dashboard, which enables the stakeholders to obtain meaningful insights to support their traffic management tasks.

The *SB Admin* template, which utilizes the *Bootstrap* framework, was used as a structural basis for developing the dashboard monitoring environment. The visualizations were divided into two pages: '`Parking monitoring`' and '`Performance analytics`'. An overview of the resulting application is visible in Figure 3.10. Moreover, an enlarged version of these screen recordings can be found in Appendix C.

Having developed the dashboard, the missing link from the core system output to the end user is now established. This concludes the development of the comprehensive system.

3.7.4 Performance testing

Even though the standalone machine learning model has already been tested thoroughly in Section 3.6, it is also crucial to assess the performance of the overall real-time system. Since the system will be continuously exposed to unseen data, this provides insights about how the system performs when it is implemented in a real-world scenario.

As previously mentioned in Section 3.7.2, the system computes the errors dynamically by comparing a newly received measurement to the predictions of the past hour. The absolute error values are saved in the *occup* table of the system database. The system's predictive performance was therefore assessed by letting the system run (and thus generate predictions) continuously for seven consecutive days. Considering that measurements become available every 11 minutes and that a week consists of 10,080 minutes, this leads to a sample size of $10080/11 = 916$ error values per predictive horizon (from 5 up to 60 minutes ahead), and thus $916 \cdot 12 = 10996$ error values in total. This provides a sufficiently large basis to determine the overall performance, as well as a breakdown of performance per predictive horizon.

To facilitate direct comparison with the errors of the standalone model on the test set, the overall system performance is assessed using the MAE, MSE and MASE metrics. The former two can directly be computed from the individual errors in the *occup* table, but the MASE needs the MAE of the naive model on top of this. In order to relieve the system from extra computational complexity, the decision was made to adopt the same MAE as derived when assessing the performance of the naive model on the test set. Besides a comprehensive overview of the three metrics, a detailed breakdown of the MASE per predictive horizon is also provided. This is done by computing the MASE for every predictive horizon (i.e. each column of the *occup* table). According to Section 3.6, the same has already been done for the standalone machine learning model. Hence, they are combined into a line graph which facilitates an insight of how well the system performs in a dynamic real-time situation, compared to a static test on a historical dataset.

3.8 TRANSFERABILITY OF THE SYSTEM

So far, the research has been focused specifically on the Centraal garage in Arnhem, The Netherlands. On the long term, however, the objective of the system is unquestionably to accommodate other parking areas as well. Transferability and expansion of the system is therefore an important topic.

3.8.1 Input variable dependency

Whether other parking areas can successfully be added to the system is arguably influenced by the kinds of data which are supplied. For instance, when a parking area does not disclose a real-time feed of its occupancy rate, the resulting model would likely perform worse since it does not have any knowledge about the past hour. An insight

into input variable dependencies therefore provides a clear indication of the flexibility with which new parking areas can be added to the system.

To test the importance of input variables, a *feature elimination* strategy was used. This technique entails that variables are categorically removed from the input dataset. For every variable (or category of variables) that is removed, a model is trained with the remaining input columns. The performance of this model is then assessed using the MSE on the test set, after which this value is subtracted from the MSE of the reference model (i.e. the model without any eliminated features). This is how the *increase of MSE* is derived. Ranking these results therefore gives a reliable indication of the feature importance within the model.

Since some input variables are interrelated (e.g. the lookback windows), it was a logical choice to categorize them such that they could be eliminated collectively. During the test, all identified categories were then separately eliminated, such that only a single category of variables was absent during every test round. This facilitates the assessment of a particular category's importance. Note that the order of elimination is irrelevant since it would not alter the outcome of the test.

The following variable categories were ultimately identified:

- **Weekday**
- Both **time of day** variables, i.e. the *hour* and *minute* components
- **Rain**
- **Temperature**
- The five **previous occupancy rate** variables, i.e. the complete lookback window of the past hour
- All mutations of the **traffic flow** variables, i.e. for all locations and lookback timesteps

The MSE increase values, which were measured after individually eliminating the above variable categories, thus provided a ranking of the most influential variables of the model. By weighing this against the availability and accessibility of data sources for these variables, conclusions were drawn about the potential transferability of the system. For instance, if the *rain* variable would hypothetically have a large importance on the model while a corresponding data source is unavailable (either historically or in real-time) for most parking areas, the transferability would be at risk.

3.8.2 Impact of limited training data

Not only the kinds of data, but also the amount of training data is a crucial factor regarding the transferability of a machine learning system. As mentioned in Section 3.2.1, the Centraal garage was chosen because the municipality of Arnhem is currently the only Dutch provider of such extensive historical datasets in combination with a real-time feed. It should be mentioned, however, that there is a significant number of parking areas which disclose a real-time data feed only [38]. An opportunity would therefore be to dynamically collect data, such that an independent historical dataset is gradually developed for each parking area. However, machine learning models need a large amount of training data to learn from [13], so collecting an appropriate amount of data will be very time-consuming. In order to satisfy certain time and resource constraints, it is therefore crucial to know how much training data is actually needed to obtain a decently performing model.

This was tested by recursively dividing the training set into halves, and training a new in/outflux and occupancy rate model every time based on the resulting subset. To illustrate, after every round of the test, half of the training set remains as input for the next round. The data was not shuffled to maintain the natural order of the time series. Hence, the oldest half is removed from the set, while the most recent half remains for the next round. Given the fact that the original training set contains 544,680 samples (approx. 1 year of data), the first subset will contain 272,340 samples (i.e. the last 6 months), the second subset will contain 136,170 samples (i.e. the last 3 months), et cetera. The loss L_n of the resulting model (i.e. the MASE corresponding to fraction n of the original training set) was measured using the test set. In total, this was done 15 times, as visible in the parameter grid of Figure 3.11. The model configuration was kept constant throughout the test.

Fraction of original training set

1	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$	$\dots$	$\frac{1}{32768}$
$[L_1$	$L_{0.5^1}$	$L_{0.5^2}$	$L_{0.5^3}$	$\dots$	$L_{0.5^{15}}]$

Figure 3.11: Parameter grid of training subsets

After gathering the test results for every subset, a line graph was made of the MASE against the size of the training set. A logarithmic scale was chosen for the x-axis (i.e. to denote the training set size) since the fractions are all powers of 0.5. The $MASE = 1$ guideline was then again used to assess at which point the performance becomes unacceptable (i.e. where the model performs worse than the naive model). The corresponding training set size was then computed by multiplying the fraction with the total number of samples,

i.e. 544,680. The outcome therefore gives an indication of the amount of data which is needed to develop both the in/outflux and the occupancy rate model with acceptable performance.

4 RESULTS AND DISCUSSION

Using the previously defined research methods, the results were generated and collected. This entails the results of development and tuning of the models, as well as the twofold testing and comparison results, followed by the outcome of the final predictive application.

4.1 FEED-FORWARD NEURAL NETWORK

4.1.1 Architecture selection

The 2D grid search was executed according to the previously defined methodology of Section 3.4.1.1. The total running time of the test was approximately 1 day and 20 hours. The MSE observations $L_{n,m}$ corresponding to all configurations of the (n, m) grid were visualized using a heatmap. The results are visible in Figure 4.1.

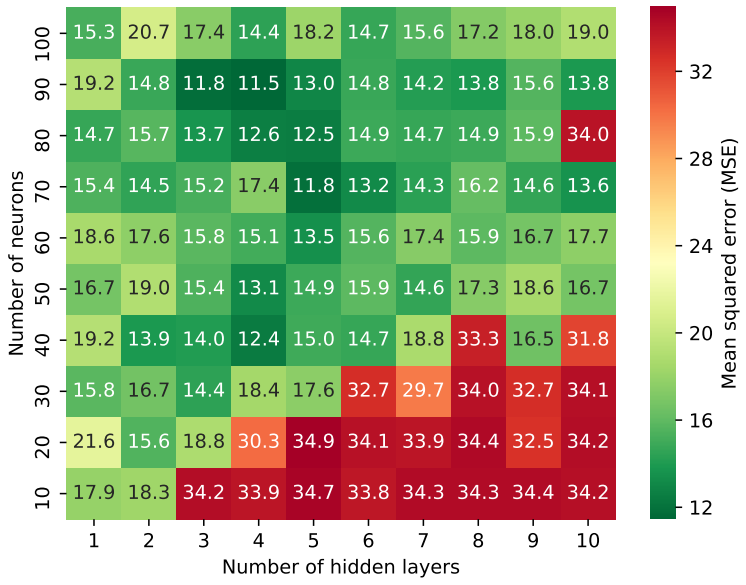


Figure 4.1: Results of the FFNN architecture selection test

The worst performance (highest loss) can be observed in the bottom-right part of the heatmap. This can be attributed to the fact that a relatively low number of neurons is spread across a high number of layers. As a result, some layers contain only one or two neurons, which leads to underfitting.

The *'sweet spot'* in terms of performance is located around the top-left and middle-left areas of the heatmap. This suggests that the feed-forward neural network performs best when a high number of neurons is divided between a relatively small number of hidden layers. The minimum MSE was observed at configuration $(90, 4)$, i.e. with 90 neurons spread across 4 hidden layers. Hence, the optimal model architecture (i.e. the composition of hidden layers in sequential order) can be schematically represented as follows:

$$[24, 22, 22, 22]$$

4.1.2 Hyperparameter tuning

The underlying structure of the FFNN has now been definitively established. Using this architecture as basis, it is therefore time to optimize the hyperparameters. The test, which aims to find the optimal learning rate, was executed according to the previously defined methodology of Section 3.4.1.2. The total running time of the test was approximately 1 hour and 45 minutes. For all learning rates α, the progression of MSE over time (i.e. the number of epochs) was visualized using a line graph. The results are as visible in Figure 4.2.

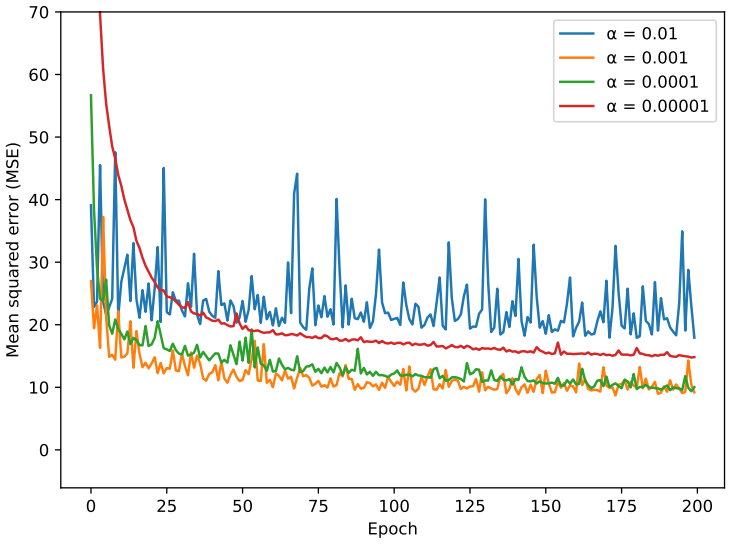


Figure 4.2: Results of the FFNN hyperparameter tuning test

The graph clearly shows that the learning rates lead to significant differences in the progression of the validation loss. As expected, higher learning rates initially show a rapid decrease of MSE, but then stagnate and show unbalanced behaviour. Evidently, the learning rate $\alpha = 0.01$ is too high: even though the loss is initially relatively low (compared to the other curves), it shows large oscillations and fluctuations and very little convergence. This suggests that the step size of

the gradient descent is too high, which makes it impossible to find the minimum loss. The lower learning rates, on the contrary, demonstrate a stable decrease but converge too slowly. Notably, the learning rate $\alpha = 0.00001$ results in a very durable and reliable decrease of MSE, even though it is still relatively high after 200 epochs. According to the previously defined criteria, the learning rate $\alpha = 0.0001$ provides an optimal balance: after 200 epochs, the corresponding loss is the lowest (compared to the other configuration) and is still descending at a substantial pace. Out of all learning rates, it therefore provides the best step size in order to descend to the lowest possible validation loss within the boundaries of temporal and computational complexity. A learning rate of $0.0001 (= 10^{-4})$ was therefore definitively selected for the final feed-forward neural network.

4.1.3 Candidate model

Based on the optimized parameters which were derived in the previous sections, the feed-forward neural network was definitively trained according to the methodology of Section 3.5. The result is a neural network with a $[24, 22, 22, 22]$ architecture (i.e. **90 neurons** and **4 hidden layers**) which was trained over the course of **2000 epochs** at a **learning rate of 0.0001**. To prevent overfitting, a model checkpoint was applied to save the model at the epoch where the lowest validation loss was measured.

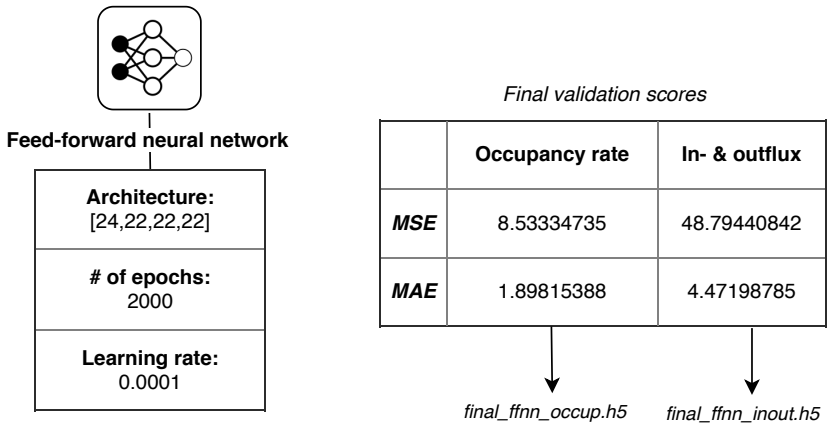


Figure 4.3: Schematic overview of the resulting FFNN

In addition to the occupancy rate model, the in/outflux model was now also developed using these parameters. An overview of both models, their mutual configuration as well as their respective validation scores can be seen in Figure 4.3.

4.2 RANDOM FOREST

4.2.1 Architecture selection

After having found the optimal architecture and hyperparameters of the feed-forward neural network, a similar process is now applied onto the random forest. Regarding the selection of the optimal architecture, the grid search was executed according to the previously defined methodology of Section 3.4.2.1. The total running time of the test was approximately 1 hour and 9 minutes. The MSE observations L_n corresponding to all configurations of the n (*number of trees*) grid were visualized using a line graph. The results are visible in Figure 4.4.

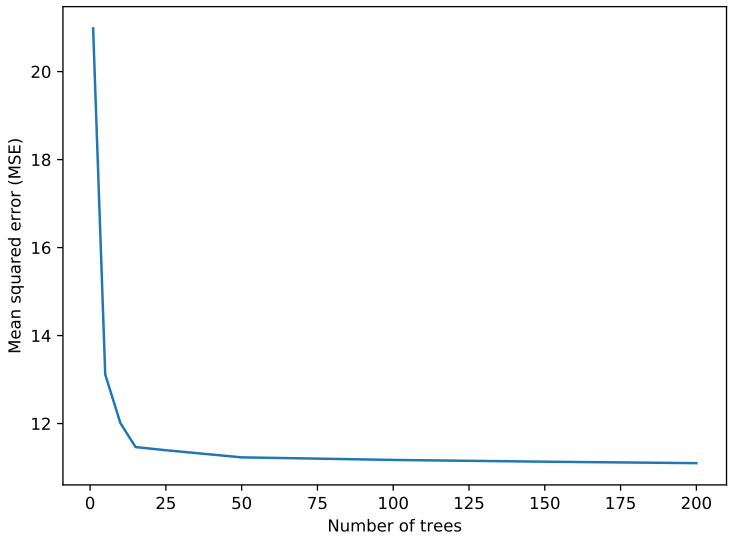


Figure 4.4: Results of the RF architecture selection test

The results suggest that the validation loss is subject to exponential decay as soon as the number of trees n increases. Initially, when there is only one tree in the ensemble, the random forest can essentially be regarded as an ordinary *decision tree*. The real power of the random forest becomes evident when the number of trees grows larger than 1. Around $n = 50$, the MSE seems to reach a plateau state. A higher number of trees would thus be ineffective: no significant performance gain will occur anymore, even though the computational complexity will rise dramatically. Therefore, 50 trees were definitively chosen as the optimal number of trees for the random forest model.

4.2.2 Hyperparameter tuning

Having found the ideal number of trees, it is time to tune the hyperparameters of the random forest. Hence, a grid search was executed

according to the previously defined methodology of Section 3.4.2.2. The total running time of the test was 10,455 seconds (approximately 2 hours and 55 minutes). The MSE observations $L_{d,f}$ corresponding to all configurations of the (d, f) grid were visualized using a multiple-line graph. Every setting of the maximum features f is therefore represented by its own line. The results are visible in Figure 4.5.

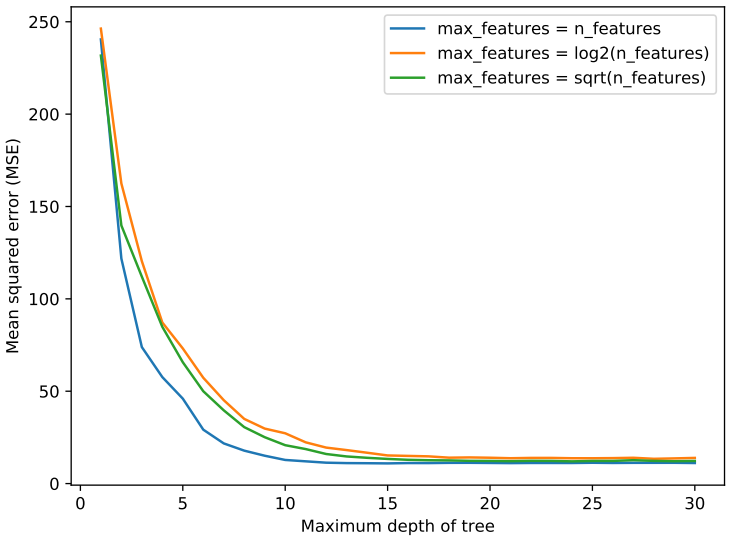


Figure 4.5: Results of the RF hyperparameter tuning test

From the results, it becomes clear that the three configurations of f roughly follow the same trend in relation to the maximum depth d. Nonetheless, in case of $f = n_{features}$, the validation loss clearly decreases faster and reaches the plateau state at a significantly lower value of d. This is the preferred option, since the maximum depth should be rather small in order to minimize the computational complexity (i.e. training and prediction times). Using this configuration, the minimum MSE is already reached at a maximum depth of 12, after which no further performance gain takes place anymore. A hyperparameter configuration where $f = n_{features}$ and $d = 12$ is therefore arguably the optimal balance.

4.2.3 Candidate model

Based on the optimized parameters which were derived in the previous sections, the random forest was definitively trained according to the methodology of Section 3.5. The result is a random forest consisting of **50 trees**, each with a **maximum depth of 12** and a **maximum features per split** which is equal to the **number of features** of the training set.

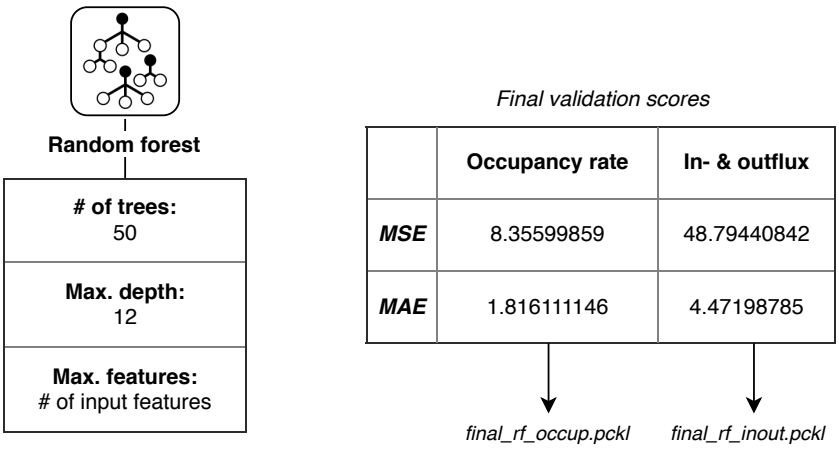


Figure 4.6: Schematic overview of the resulting RF

In addition to the occupancy rate model, the in/outflux model was now also developed using these parameters. An overview of both models, their mutual configuration as well as their respective validation scores can be seen in Figure 4.6.

4.3 INTER-MODEL COMPARATIVE TESTING

4.3.1 Quality of predictions

In order to determine the model with the best predictive performance, the test set was supplied to the two candidate models, after which the MSE, MAE and MASE metrics were computed. Before analyzing the characteristics and distribution of the MASE for different predictive horizons, the metrics are first presented in the form of a comprehensive table. The red and green color-codings denote whether the random forest or the feed-forward neural network performed better (i.e. which one had the lowest error). The resulting tables are visible in Figure 4.7.

Influx

	FFNN	RF
MSE	34.02645	36.04549
MAE	3.93467	3.99512
MASE	0.74247	0.75387

Outflux

	FFNN	RF
MSE	35.12385	37.86854
MAE	3.85855	3.89326
MASE	0.79100	0.79812

Occupancy rate

	FFNN	RF
MSE	6.37295	7.30700
MAE	1.65064	1.74443
MASE	0.38478	0.40664

Figure 4.7: Comprehensive overview of inter-model test results

Even though the differences are not very large, the results demonstrate that the feed-forward neural network outperforms the random forest in every aspect. Concerning both the occupancy rate and the

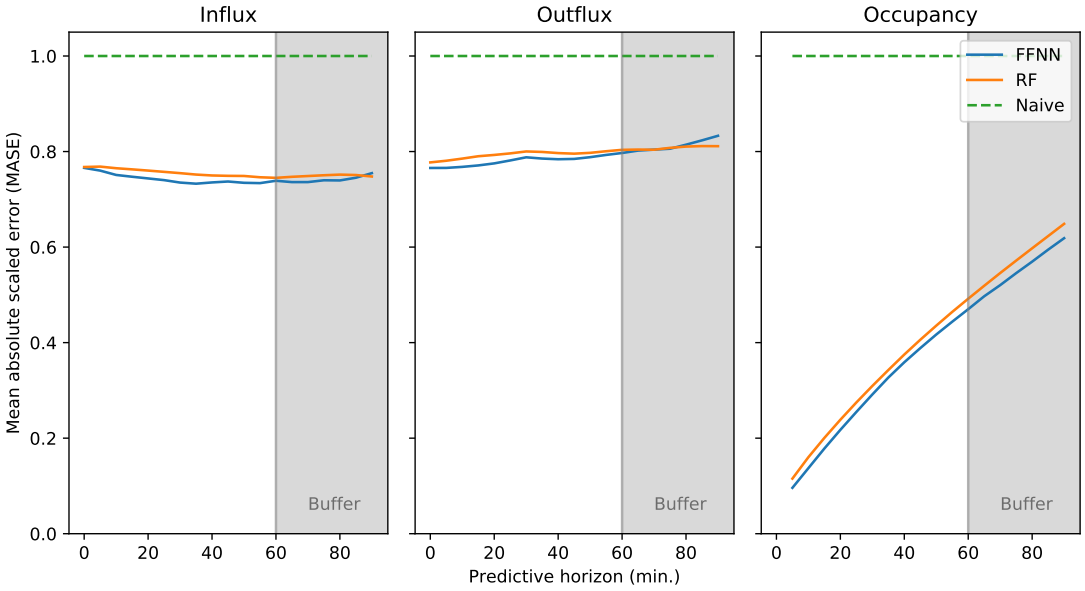


Figure 4.8: Results of MASE against the predictive horizon

in/outflux models, the FFNN predicts with lower errors (as acknowledged by all three metrics) and can therefore be regarded as the stronger model configuration. For a more precise insight into model performance with regard to the predictive horizon, the three-fold line graph is established in Figure 4.8.

First and foremost, the results show that every model predicts significantly better than the naive benchmark. In particular, the occupancy rate models (both the FFNN and the RF) perform exceptionally well: across all horizons, the corresponding MASE is around 0.38 (see Figure 4.7) which suggests that the performance is 2.6 times better than the naive model. This amounts to a performance gain of 160%. When considering only 60 minutes ahead (i.e. disregarding the buffer), the performance gain is even higher at 235%. Regarding the influx and outflux models, the overall MASE is between 0.75 and 0.8, so both models predict $1\frac{1}{3}$ to $1\frac{1}{4}$ times more accurately than the naive model. This amounts to a performance gain between $33\frac{1}{3}$% and 25%.

As the summarized overview of metrics already suggested, the MASE plots confirm that the FFNN outperforms the RF. However, it is interesting to see that the differences become smaller as soon as the predictive horizon increases. In fact, in case of the outflux, the RF starts to to outperform the FFNN after predicting approximately 65 minutes or further ahead. A similar phenomenon occurs at the influx, where the RF starts to perform better after 85 minutes. However, both cases belong to the *buffer* area and are therefore less relevant than the first 60 minutes - which is where the FFNN performs significantly better.

Other insights can also be gained from the results. Notably, predicting occupancy rates is more reliable than predicting the influx and outflux, as indicated by the fact that the MASE of the former is significantly lower. This can obviously be explained by the fact that the lookback window does not contain particular information about the influx and outflux: only the *preceding occupancy rates* are fed to the model. The occupancy rate models can therefore directly base their predictions on these inputs. The influx and outflux models do not possess this advantage, even though the interrelations between the preceding occupancy rates do provide some indication of the respective flows. Besides, the difference in performance can also be explained by the fact that the occupancy rate is a continuous variable with low variance, as opposed to the influx and outflux - both of which are discrete (aggregated per 10 minutes) and are known to suffer from high fluctuations (see Section 3.4).

It is also remarkable that the performance of influx and outflux predictions remains relatively consistent, regardless of the predictive horizon. Hypothetically speaking, the MASE would increase when predicting further away from the $t = 0$ point, given the fact that uncertainty increases when looking further into the future, and associated difficulty in that case to outperform the naive model. The reason why this is not applicable to the influx and outflux might be similar to the previous case: since the influx and outflux do not possess knowledge about their exact state before $t = 0$, the overall performance is worse even though it stays consistent over time. Even though the outflux experiences a small decrease of performance, it is clearly not as evident as with the occupancy rate models.

Even though the differences are not large (approximately 0.05 on average), another interesting observation is the fact that the influx models perform better than the outflux models. This could be explained by the fact that the traffic flow variables are less relevant to the outflux than to the influx. After all, the traffic flows facing inwards to the city center cannot be directly related to the outflux. The same goes for the traffic flows facing outwards: they only become relevant as soon as the vehicles have already left the garage.

From the violin plot in Figure 4.9 (see Section 3.6.2 for an explanation about this plotting technique), some additional insights about inter-model performance can be gained. Overall, the center of gravity of every distribution is located under the $MASE = 1$ line, which shows that all models predominantly predict better than the naive model. Yet, the distributions of influx and outflux errors contain more irregularities than the distribution of occupancy rate error (which is very smooth and compact). This is also visible by the magnitude of the outliers. Also, the distributions are more skewed towards a higher MASE (above 1), which makes it evident that a significant number of in- and outflux predictions are worse than those of the naive model.

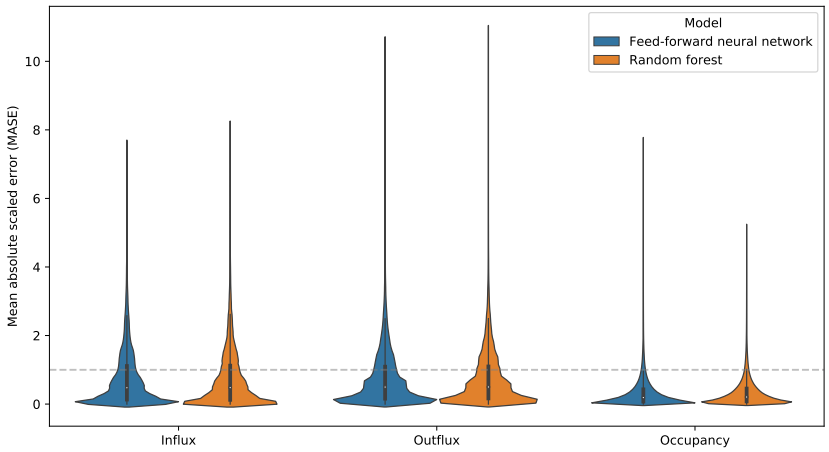


Figure 4.9: Violin plot of error distributions for both model types

In contrast, almost none of the occupancy rate predictions are worse than those of the naive model. These conclusions correspond to the observations which were previously made using Figure 4.8.

An interesting observation is the relatively high magnitude of outliers in the outflux distribution in comparison to the influx distribution. Similar to the conclusions drawn before, a plausible explanation for this is that the influx models have an advantage since the traffic flow inputs are directly related to its output predictions.

4.3.2 Efficiency of predictions

The models were also compared by their prediction time, which gives an indication of the efficiency of each model type. After 100 measurements, the mean prediction time of the feed-forward neural network (both the in/outflux and occupancy rate models) was determined to be **1.57 seconds**. Using an identical approach, the mean prediction time of the random forest was determined to be slightly lower, i.e. **1.32 seconds**. These results demonstrate that the random forest is slightly more time-efficient than the feed-forward neural network. Yet, the differences are not very large (only a fraction of a second) and it is therefore unlikely that the difference would be noticeable within the real-time predictive system. Presumably, the small differences in prediction time are caused by the relatively high number of paths which must be traversed through all 90 nodes in the neural network, as compared to the 50 trees of the random forest.

4.3.3 Final model selection

The previously discussed results demonstrate that the feed-forward neural networks outperforms the random forests in terms of MSE, MAE and MASE scores on the test set. However, the inter-model

differences are relatively small. In terms of predictive efficiency, the random forest performs slightly better. Again, however, the difference is very small (i.e. only a quarter of a second).

Overall, the results therefore do not yet provide a unanimous answer to which model configuration is best. A comparison of the arguments for using both configurations could therefore produce a clearer outcome. With the findings of the literature study (see Section 2.2) in mind, the benefits of both configurations can be defined as:

FFNN

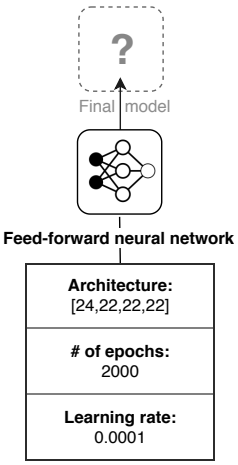
- Higher quality of predictions (lower errors)
- More flexibility regarding model optimization
- Ability of dynamic retraining when new data is available

RF

- Higher predictive efficiency (lower prediction times)
- Model optimization is less time-consuming

Rationally speaking, the additional benefits of the FFNN seem to outweigh those of the RF. Especially the option of dynamic retraining is a convenient functionality for a real-time system where new data is received on a frequent basis. Additionally, the FFNN has a multitude of other hyperparameters which can be tuned to optimize the quality and efficiency of predictions. The random forest is simpler to tune, but cannot offer the same flexibility. Overall, the feed-forward neural network can therefore be considered as the most suitable model type for predicting the influx, outflux and occupancy rate.

Consequently, the `final_ffnn_occup.h5` and `final_ffnn_inout.h5` models were adopted in the real-time predictive application. A schematic overview of the definitive configuration, as well as a summary of the final testing scores is visible in Figure 4.10.



Final testing scores

	Influx	Outflux	Occupancy rate
MSE	34.02645	35.12385	6.37295
MAE	3.93467	3.85855	1.65064
MASE	0.74247	0.79100	0.38478

Figure 4.10: Schematic overview of the definitive configuration

4.4 REAL-TIME SYSTEM PERFORMANCE

After developing the comprehensive system according to the methodology of Section 3.7, it operated continuously for one week in order to dynamically gather error magnitudes. The resulting errors (i.e. for all predictive horizons) were then used to compute the MAE, MSE and MASE values for the overall system. Figure 4.11 shows the comparison between the performance of the real-time system and the previously found testing results of the FFNN on the test set (denoted as *reference*). Note that the reference metrics are lower than the ones previously listed in Figure 4.10. This is caused by the fact that the predictive horizon is limited to only 60 minutes (i.e. since the *buffer* is used up by the real-time predictions), which logically lowers the error magnitudes.

	Real-time system	*Reference*
MSE	8.57667	3.70145
MAE	1.87249	1.27988
MASE	0.36789	0.29835

Figure 4.11: Overview of real-time system performance

The results show that the system predicts significantly worse in real-time than the standalone FFNN did on the historical test set. Notably, the MSE has doubled and the MASE has increased by almost 0.1. A likely cause for this difference would be the significant delay of incoming occupancy rate measurements (i.e. 11 minutes). In the worst case, the system does therefore not possess any information about the last 11 minutes, and will therefore have to use a large part of the buffer. This increases the uncertainty of predictions, and hence the magnitude of errors. When testing the standalone FFNN model, this problem did not occur since the definite occupancy rate of the last minute was always present as a direct input for the model. This issue highlights the importance of a reliable and frequent (preferably up-to-the-minute) input feed. It should be noted, however, that the current real-time system still delivers predictions with a very high quality: the MASE is 0.36789, which amounts to a performance gain of 172% with regard to the naive model. Even though the quality of predictions are thus slightly sacrificed by the real-time implementation, the performance is still excellent altogether.

Besides an overview of the overall performance errors, a breakdown of the MASE per predictive horizon facilitates a more detailed comparison between the predictive quality of the real-time system, the standalone model and the naive model. The results are visible in Figure 4.12.

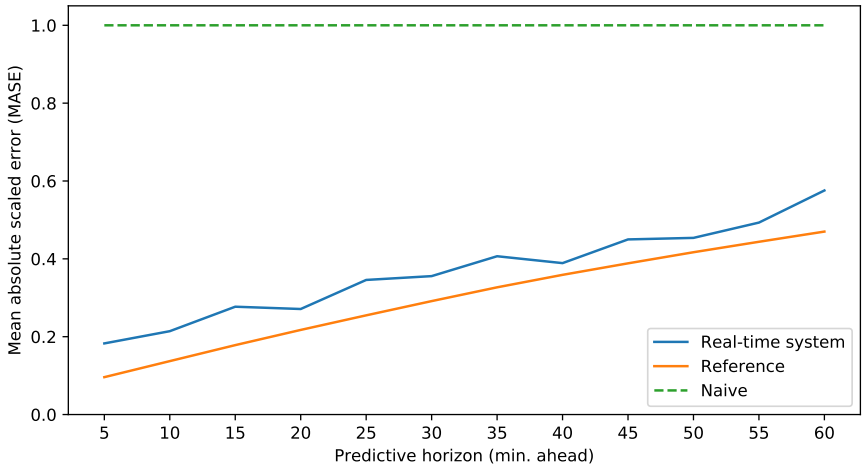


Figure 4.12: Comparison of error magnitude for every predictive horizon

The results show that the MASE of the real-time system is consistently (approx. 0.1) higher than the MASE of the standalone model on the test set. This supports the concept that the quality of all predictions is sacrificed by the delay of incoming occupancy rate measurements. Another interesting observation is that the MASE of the real-time system shows a 'rougher' pattern than the reference. This could be caused by the fact that a relatively low sample size was used (i.e. only 7 days) when compared to several months of historical data for the reference test. Once the system is kept running for a longer period, it is therefore expected that the line will smooth out. Overall, however, the MASE of the real-time system still seems to follow the same upward trend as the reference MASE, which is logical considering that the uncertainty grows when the system predicts further away from $t = 0$. Altogether, it can be concluded that the real-time system performs slightly worse across all horizons (i.e. up to 60 minutes ahead), but that the overall quality of predictions is still outstanding.

4.5 TRANSFERABILITY OF THE SYSTEM

4.5.1 Input variable dependency

Using the *feature elimination* strategy as described in Section 3.8.1, the significance of every input variable was obtained. This provides insights into the reliance of the models upon the availability and accessibility of certain data sources. The results are visualized using a horizontal bar chart, as visible in Figure 4.13.

The high influence of the **previous occupancy rate** variables immediately stands out from the results: when the complete lookback window was eliminated from the model, the MSE increased by no

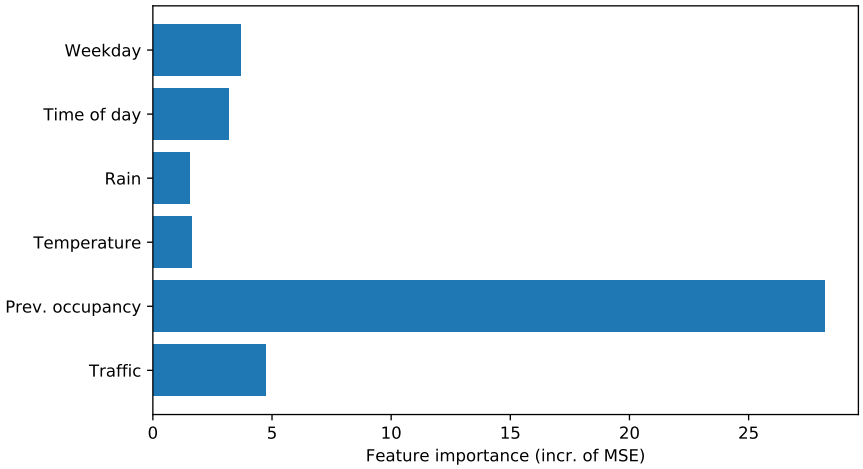


Figure 4.13: Overview of feature importance within the model

less than 28. This is a 340% increase from the initial MSE of 6.37. The results therefore suggest that the driving force behind the model is its knowledge about the past hour. Hence, the real-time component is essential for the predictive quality of the system, even though the current availability of real-time data feeds about the occupancy rate is limited. For a considerable share of the parking areas this would imply that the full potential of the predictive system cannot be unlocked. Unless the availability of real-time parking data will increase in the upcoming years, this can be regarded as a limitation of the transferability.

The **traffic flow** variables also seem to have a significant importance for the model: the MSE increases by almost 5 after eliminating these variables. This, however, is less threatening for the transferability of the system since the traffic flow data is published by the NDW for the majority of highways in The Netherlands. The availability of the data, both historically and in real-time, is therefore satisfactory which makes it feasible to implement these variables when modelling other parking areas. The same is applicable for the **rain** and **temperature** variables, both of which can be obtained for every weather station from multiple data sources (e.g. KNMI and Weerlive).

Together, the **weekday** and **time** variables also provide a meaningful input to the model. Both are independent of any external real-time data source since all other variables are time-based and therefore characterized by a mutual timestamp (i.e. the data are time series). Hence, they can be derived easily from the existing data sources or from the system clock. The availability of the weekday and time variables is therefore guaranteed, irrespective of the parking area. As a result, they will neither limit nor restrain the transferability of the system.

4.5.2 Impact of limited training data

In order to determine how much training data can be compromised to maintain an acceptable level of performance, the testing procedure of Section 3.8.2 was applied on the final FFNN configuration. The resulting line graph, which is visible in Figure 4.14, shows the MASE of both in/outflux and occupancy rate models against the training set size. The $MASE = 1$ line is added to determine the point at which the models start to perform worse than the naive model.

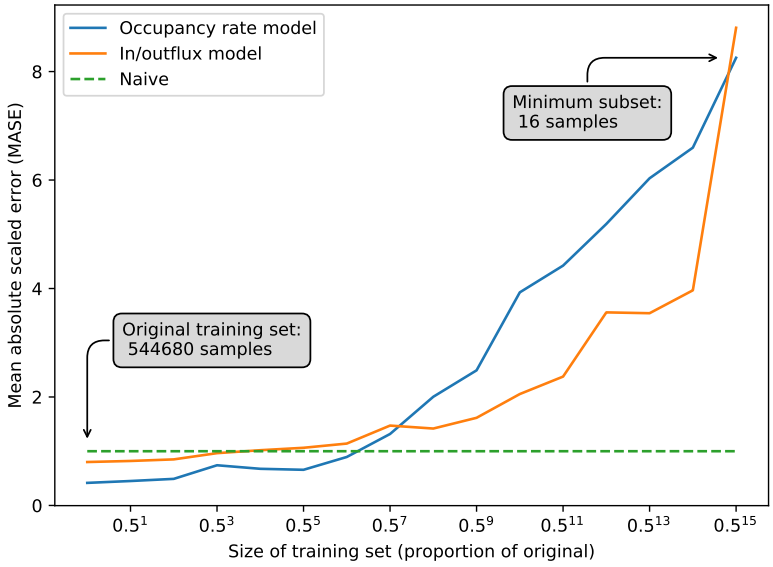


Figure 4.14: Plot of MASE against amount of training data

The results demonstrate that the MASE increases significantly when the amount of training data is reduced. In the worst case (when the training set has been halved 15 times), the MASE is around 8, which suggests that the models perform eight times worse than the naive model. This is unsurprising, given the fact that only $0.5^{15} \cdot 544680 = 16$ samples (i.e. a quarter of an hour) are left in this case. Overall, the results confirm the traditional proposition that more training data implies a better performing machine learning model.

Up until the sixth halving of the training set, the occupancy rate model performs better than the naive model. This corresponds to $0.5^{6} \cdot 544680 = 8510$ training samples. It can therefore be concluded that approximately 8,510 minutes (i.e. almost 6 days) of data are minimally needed to develop an FFNN which performs better than the naive model. Given that the weekly pattern was identified as an important seasonality of the parking data (as explained in Section 3.6.1), this is a logical outcome.

The in-/outflux model clearly needs more training data to perform at a satisfactory level: after already three halvings the model is outperformed by the naive model. This corresponds to $0.5^{3} \cdot 544680 = 68085$

training samples. Hence, it can be concluded that 68,085 minutes (approximately one and a half month) of data are minimally needed to develop an adequately performing FFNN for the in- and outflux. The fact that this number is relatively large might be related to the high variance of the in- and outflux variables. As a result, more training data will be needed for the model to generalize at an acceptable level.

Overall, the results demonstrate that a significant amount of training data can be compromised while maintaining a satisfactory level of performance. Only 1.5 months of data is necessary to develop a set of models which predicts the influx, outflux and occupancy rates better than the naive model. This is a promising finding regarding the ease of implementing other parking areas into the system, especially in cases where the availability of data is substandard. Therefore, the results are promising for the transferability and potential scalability of the system. Still, however, it should be mentioned that optimal results are obtained with higher volumes of data: the goal should therefore always be to collect as much data as possible with the given time and resources.

5 CONCLUSIONS AND RECOMMENDATIONS

5.1 CONCLUSIONS

The aim of this thesis was to find the optimal methodology for predicting the influx, outflux and occupancy rate of parking areas on a horizon of up to 60 minutes ahead. From an in-depth study of existing literature, a combination of time, weather, traffic flows and occupancy rates of the past hour was determined to be an optimal range of independent variables for a predictive model. In addition, it was concluded that *random forest* and *feed-forward neural network* are the foremost machine learning techniques for predicting parking area states on the short-term. Both techniques were implemented in a real-world case in the city of Arnhem, The Netherlands, resulting in two fully optimized candidate models which can predict the future states of a large garage in the city center. The available historic and real-time data feeds served as a constraint for the methodology. An extensive inter-model comparative testing process has demonstrated that a **feed-forward neural network** with a hidden layer composition of $[24, 22, 22, 22]$ and a learning rate α of 0.0001 outperforms its opponent (the optimized random forest) in terms of predictive quality and potential flexibility, although the differences are small and both are outperforming naive models. Notably, when predicting the occupancy rate, the resulting feed-forward neural network achieves a performance gain of 235% over the naive model (considering only the horizons up to 60 minutes, hence disregarding the buffer).

When predicting influx and outflux, the performance gain is less obvious but still significant, i.e. $33\frac{1}{3}$% and 25% respectively. The test results therefore prove that predicting the in- and outflux is a far more difficult task which requires more training data than occupancy rate. This likely relates to the differences in data variance and lacking real-time data feed for in- and outflux. Yet, it should be mentioned that the performance of predicting in- and outflux turns out to be less sensitive to the predictive horizon.

When deployed in a real-time system, the model has also proved to outperform the naive model by a large margin of 172%. Also, the MAE shows that the system is able to predict the occupancy rate with an average error of only 1.87. This once again demonstrates that the system generates exceptionally accurate predictions, and that it

is able to provide valuable information for pro-active traffic management as well as mobility service providers.

With regard to the transferability of the system, some limitations were identified, yet the overall results seem promising. The high importance of the occupancy rate lookback window suggests that the availability of real-time parking data is crucial for the system to perform optimally. Before a new parking area can be added to the system, it must first be ensured that historical and real-time data feeds regarding the specific parking area are fully available and operational. Since the majority of parking operators do not yet disclose this data, this can be regarded as a limitation to the transferability. However, there is an upward trend visible in availability of parking data (both historically and in real-time), and therefore it is expected that the potential for expansion will grow over time. Additionally, the results have demonstrated that a relatively small amount of training data is needed to maintain satisfactory performance. Regarding the occupancy rate, only six days of data are needed to achieve better performance than the naive model. For the in-/outflux, significantly more training data (i.e. one and a half month) is required, but still this is significantly less than the original training set of approx. one year. This is highly promising regarding the ease of implementing other parking areas into the system, especially in cases where the availability of data is substandard.

5.2 FUTURE RESEARCH

Further research will focus on extending the system deployment towards other parking areas, improving the in- and outflux predictions, performing anomaly detection, and integrating the model with traffic state prediction models in order to empower complete predictions of urban traffic states. As mentioned before, it is expected that more and more parking areas can be incorporated into the system over time. This emphasizes the relevance of researching structural and computational complexities that are associated with scaling up the system. Also, considering the growing availability of real-time urban traffic flows (e.g. via connected traffic light controllers) it is expected that the quality of in- and outflux predictions can be improved. In addition, there are opportunities for extending the models with a measure for reliability based on further analyzing the error patterns. To illustrate, anomaly detection and data imputation could enhance the accuracy and reliability of the system. A dynamic retraining process can also be implemented to make the system robust to long-term trends and seasonalities. Finally, the application of the system to support pro-active traffic management will be further researched to assess the actual value of predictions for traffic management purposes.

BIBLIOGRAPHY

[1] CBS. Aantal personenauto's neemt verder toe. [Online]. Available: https://www.cbs.nl/nl-nl/maatschappij/verkeer-en-vervoer/transport-en-mobiliteit/infra-vervoermiddelen/vervoermiddelen/categorie-vervoermiddelen/personenauto-s

[2] S. Biswas, S. Chandra, and I. Ghosh, "Effects of on-street parking in urban context: A critical review," *Transportation in Developing Economies*, vol. 3, no. 1, pp. 1–14, Apr 2017.

[3] D. Shoup, "Cruising for parking," *Transport Policy*, vol. 36, no. 6, pp. 479–486, Feb 2006.

[4] G. Cookson and B. Pishue, *The Impact of Parking Pain in the US, UK and Germany*. INRIX Research, Jul 2017.

[5] Z. Sándor and C. Csiszar, "Role of integrated parking information system in traffic management," *Periodica Polytechnica Civil Engineering*, vol. 59, Apr 2015.

[6] CIVITAS, *Intelligent Transport Systems and traffic management in urban areas*, 2015.

[7] T. van de Zande, *Parkeerverwijssystemen*. Radboud Universiteit, 2013.

[8] L. Wismans, L. Suijs, L. Brederode, H. Palm, and P. van Beek, "State estimation, short term prediction and virtual patrolling providing a consistent and common picture for traffic management and service providers," in *25th ITS World Congress*, Sep 2018.

[9] E. Vlahogianni, K. Kepaptsoglou, V. Tsetsos, and M. Karlaftis, "A real-time parking prediction system for smart cities," *Journal of Intelligent Transportation Systems*, vol. 20, no. 2, pp. 192–204, Apr 2015.

[10] T. Giuffrè, S. M. Siniscalchi, and G. Tesoriere, "A novel architecture of parking management for smart cities," *Procedia - Social and Behavioral Sciences*, vol. 53, pp. 16–28, Oct 2012.

[11] L. Snellen, *Een instrument voor het voorspellen van de bezettingsgraad van parkeergarages*. University of Twente, 2019.

[12] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, ser. Springer Series in Statistics. New York, NY, USA: Springer New York Inc., 2001.

[13] J. D. Kelleher, B. M. Namee, and A. D'Arcy, *Fundamentals of Machine Learning for Predictive Data Analytics: Algorithms, Worked Examples, and Case Studies*. The MIT Press, 2015.

[14] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *Journal of Machine Learning Research*, vol. 3, pp. 1157–1182, Mar 2003.

[15] B. Chen, F. Pinelli, M. Sinn, A. Botea, and F. Calabrese, "Uncertainty in urban mobility: Predicting waiting times for shared bicycles and parking lots," in *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*, Oct 2013, pp. 53–58.

[16] C. Badii, P. Nesi, and I. Paoli, "Predicting available parking slots on critical and regular services by exploiting a range of open data," *IEEE Access*, vol. 6, pp. 44 059–44 071, Aug 2018.

[17] R. Hampshire, T. Fabusuyi, V. Hill, and K. Sasanuma, "Decision analytics for parking availability in downtown pittsburgh," *Interfaces*, vol. 44, no. 3, pp. 286–299, Jun 2014.

[18] X. Chen, *Parking occupancy prediction and pattern analysis*. Stanford University, 2014.

[19] Y. Zheng, S. Rajasegarar, and C. Leckie, "Parking availability prediction for sensor-enabled car parks in smart cities," in *2015 IEEE Tenth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, Apr 2015, pp. 1–6.

[20] A. Camero, J. Toutouh, D. H. Stolfi, and E. Alba, "Evolutionary deep learning for car park occupancy prediction in smart cities," in *Learning and Intelligent Optimization*, R. Battiti, M. Brunato, I. Kotsireas, and P. M. Pardalos, Eds. Springer International Publishing, 2019, pp. 386–401.

[21] J. Lijbers, *Predicting parking lot occupancy using Prediction Instrument Development for Complex Domains*. University of Twente, 2016.

[22] F. V. Monteiro and P. Ioannou, "On-street parking prediction using real-time data," in *21st International Conference on Intelligent Transportation Systems (ITSC)*, Apr 2018, pp. 2478–2483.

[23] M. Reinstadler, M. Braunhofer, M. Elahi, and F. Ricci, "Predicting parking lots occupancy in Bolzano," Dec 2013.

[24] C. Pflügler, T. Köhn, M. Schreieck, M. Wiesche, and H. Krcmar, "Predicting the availability of parking spaces with publicly available data," in *GI-Jahrestagung*, 2016.

[25] E. Jansen, "Wat de huidige open parkeerdata de markt oplevert." Parkeer24, Mar 2015. [Online]. Available: https://www.parkeer24.nl/nieuws/160315/wat-de-huidige-open-parkeerdata-de-markt-oplevert

[26] D. H. Stolfi, E. Alba, and X. Yao, "Predicting Car Park Occupancy Rates in Smart Cities," in *Smart Cities: Second International Conference, Smart-CT 2017, Málaga, Spain, June 14-16, 2017, Proceedings*, E. Alba, F. Chicano, and G. Luque, Eds. Cham: Springer International Publishing, 2017, pp. 107–117.

[27] J. Connor and L. Atlas, "Recurrent neural networks and time series prediction," in *IJCNN-91-Seattle International Joint Conference on Neural Networks*, vol. 1, July 1991, pp. 301–306.

[28] J. Li, J. Li, and H. Zhang, "Deep learning based parking prediction on cloud platform," in *4th International Conference on Big Data Computing and Communications (BIGCOM)*, Aug 2018, pp. 132–137.

[29] R. Caruana and A. Niculescu-Mizil, "An empirical comparison of supervised learning algorithms using different performance metrics," 2019.

[30] T. O. Kvalseth, "Cautionary note about r^2," *The American Statistician*, vol. 39, no. 4, pp. 279–285, 1985.

[31] C. J. Willmott, "Some comments on the evaluation of model performance," *Bulletin of the American Meteorological Society*, vol. 63, no. 11, pp. 1309–1313, 1982.

[32] R. Pelánek, "A brief overview of metrics for evaluation of student models," in *EDM*, 2014.

[33] R. Hyndman, "Another look at forecast accuracy metrics for intermittent demand," *Foresight: The International Journal of Applied Forecasting*, no. 4, pp. 43–46, 2006.

[34] MEDIUM.com. Decision Tree Algorithm With Hands On Example. [Online]. Available: https://medium.com/datadriveninvestor/decision-tree-algorithm-with-hands-on-example-e6c2afb40d38

[35] University of Tennessee. Feedforward Neural Network. [Online]. Available: http://web.utk.edu/~wfeng1/spark/fnn.html

[36] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[37] M. Haghighat, M. Abdel-Mottaleb, and W. Alhalabi, "Discriminant correlation analysis: Real-time feature level fusion for multimodal biometric recognition," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 9, pp. 1984–1996, Sep. 2016.

[38] OpenParking, *Data availability of facilities*, https://openparking.nl/.

[39] Gemeente Arnhem Open Data. Transactiedata - Parkeergarages. [Online]. Available: https://parkeerdata.nl/opendata/arnhem/

[40] NDW Dexter. Data exploration and exporter. [Online]. Available: https://dexter.ndwcloud.nu/

[41] KNMI. Uurgegevens van het weer in Nederland. [Online]. Available: https://projects.knmi.nl/klimatologie/uurgegevens/selectie.cgi

[42] Gemeente Arnhem Open Data. Dynamische parkeerdata. [Online]. Available: http://opendata.arnhem.nl/datasets/Arnhem::dynamische-parkeerdata

[43] NDW. Open data. [Online]. Available: http://opendata.ndw.nu/

[44] Weerlive. KNMI Weer API. [Online]. Available: http://weerlive.nl/delen.php

[45] S. Karsoliya, "Approximating number of hidden layer neurons in multiple hidden layer bpnn architecture," in *International Journal of Engineering Trends and Technology*, vol. 3, no. 6, Jan 2012.

[46] W. Koehrsen, "Hyperparameter tuning the random forest in python," *Towards Data Science*, Jan 2018.

[47] M. Gilliland, *Which Naive Model to Use?* SAS Institute, 2011, https://blogs.sas.com/content/forecasting/2011/04/26/which-naive-model-to-use/.

[48] C. Knaflic, *Storytelling with Data: A Data Visualization Guide for Business Professionals*. Wiley, 2015.

[49] S. Few, *Information Dashboard Design: Displaying Data for At-a-glance Monitoring*. Analytics Press, 2013.

A SAMPLE OF FINAL DATASET

Note that this sample only displays two entries of the `totalData.csv` file. The actual dataset contains 756,635 entries, taking up 123 MB of memory space.

Feature	Timestamp	
	2017-11-23 15:46:00	*2017-11-23 15:47:00*
occup	77.13207547169812	77.0327706057597
occup_prev	78.94736842105263	78.64945382323734
occup_prev11	79.74180734856007	79.84111221449852
occup_prev22	81.0327706057597	80.53624627606753
occup_prev33	83.0188679245283	82.52234359483614
occup_prev44	85.10427010923534	85.00496524329692
occup_prev55	88.0327706057223	88.53624627606432
in	5	2
out	3	3
weekday	3	3
hour	-0.8338858220671681	-0.8362861558477592
min	-0.5519369853120584	-0.5482932295199142
temp	107.0	107.0
rain	1.0	1.0
0-10_RWS01_MONICA_00D00C12BC0A10200005	348.0	360.0
10-20_RWS01_MONICA_00D00C12BC0A10200005	348.0	354.0
20-30_RWS01_MONICA_00D00C12BC0A10200005	438.0	432.0
0-10_RWS01_MONICA_00D00C15003210200009	690.0	672.0
10-20_RWS01_MONICA_00D00C15003210200009	570.0	642.0
20-30_RWS01_MONICA_00D00C15003210200009	726.0	696.0
0-10_RWS01_MONICA_00D03218D42800200007	168.0	168.0
10-20_RWS01_MONICA_00D03218D42800200007	228.0	216.0
20-30_RWS01_MONICA_00D03218D42800200007	252.0	234.0
0-10_RWS01_MONICA_00D03218D80010200005	564.0	588.0
10-20_RWS01_MONICA_00D03218D80010200005	498.0	450.0
20-30_RWS01_MONICA_00D03218D80010200005	600.0	654.0
0-10_RWS01_MONICA_01D145026032D007000B	924.0	966.0
10-20_RWS01_MONICA_01D145026032D007000B	990.0	996.0
20-30_RWS01_MONICA_01D145026032D007000B	972.0	972.0
0-10_RWS01_MONICA_01D14502B000D007000B	972.0	990.0
10-20_RWS01_MONICA_01D14502B000D007000B	1080.0	1062.0
20-30_RWS01_MONICA_01D14502B000D007000B	1200.0	1164.0
0-10_RWS01_MONICA_01D14502F400D007000B	948.0	942.0
10-20_RWS01_MONICA_01D14502F400D007000B	930.0	936.0
20-30_RWS01_MONICA_01D14502F400D007000B	1050.0	1092.0
0-10_RWS01_MONICA_01D145032C00D007000B	1284.0	1278.0
10-20_RWS01_MONICA_01D145032C00D007000B	1350.0	1338.0
20-30_RWS01_MONICA_01D145032C00D007000B	1440.0	1476.0
0-10_RWS01_MONICA_01D145038000D0050009	1320.0	1248.0
10-20_RWS01_MONICA_01D145038000D0050009	1296.0	1326.0
20-30_RWS01_MONICA_01D145038000D0050009	1350.0	1380.0
0-10_RWS01_MONICA_01D14503D400D0050009	1266.0	1308.0
10-20_RWS01_MONICA_01D14503D400D0050009	1200.0	1164.0
20-30_RWS01_MONICA_01D14503D400D0050009	1290.0	1272.0

B OVERVIEW OF VISUALIZATIONS

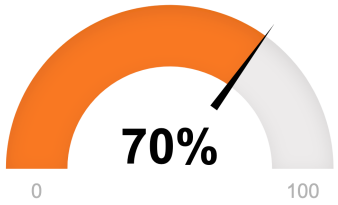


Figure B.1: Gauge denoting the real-time occupancy rate

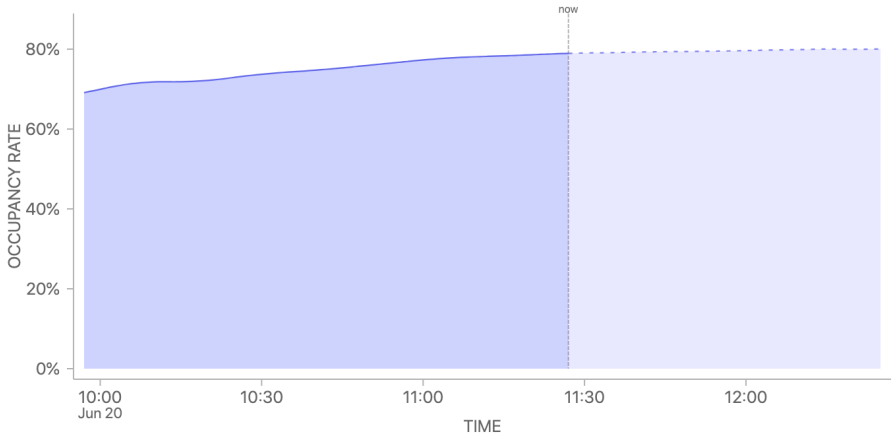


Figure B.2: Line graph showing the occupancy rate predictions

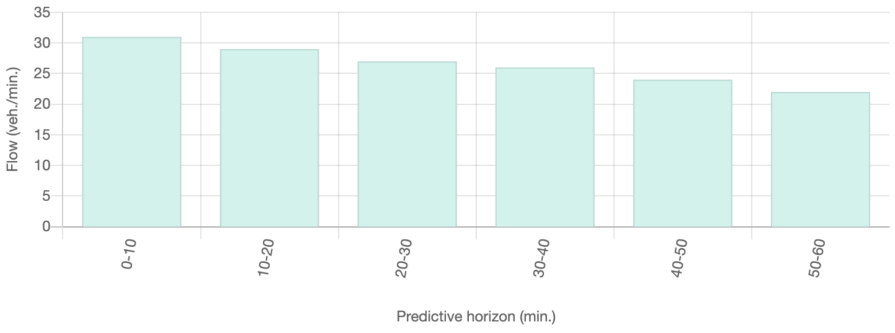
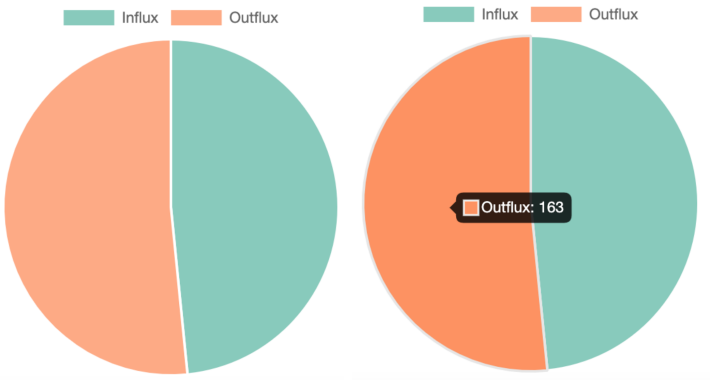


Figure B.3: Bar chart showing influx/outflux predictions



(a) Default state **(b)** Hovered state (tooltip)

Figure B.4: Pie chart showing in/outflux proportion for coming hour

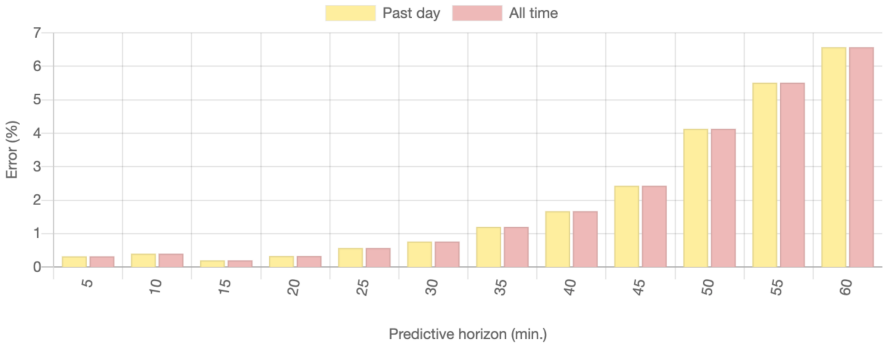


Figure B.5: Grouped bar chart showing magnitude of errors

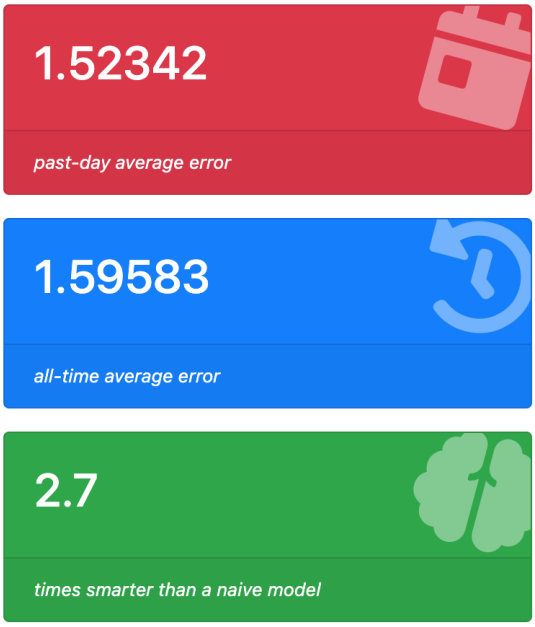


Figure B.6: Summary statistics on model performance

C OVERVIEW OF DASHBOARD

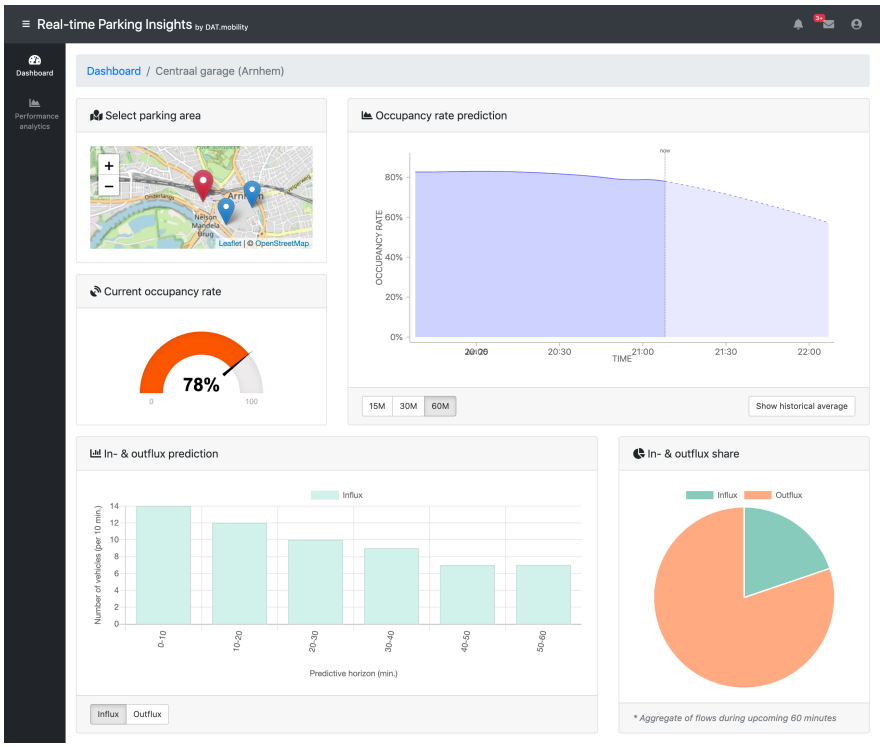


Figure C.1: Screen recording of the 'Parking monitoring' page

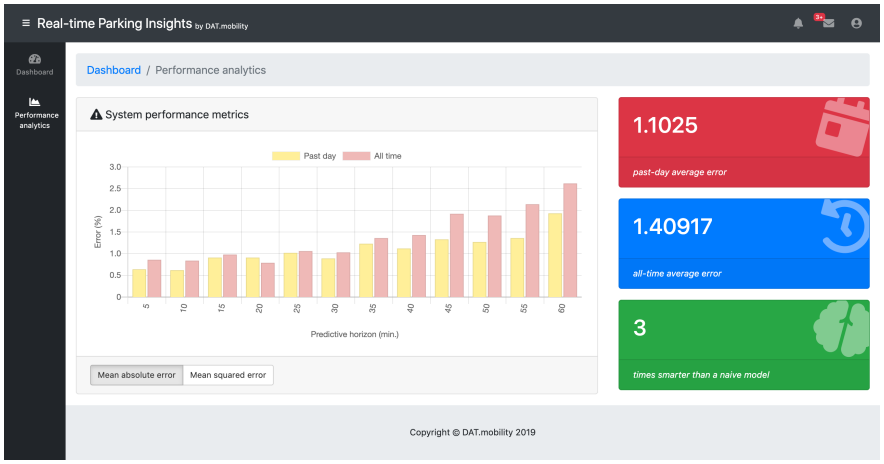


Figure C.2: Screen recording of the 'Performance analytics' page