universität wien

MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

"Prediction of the Lower Heating Value of Waste Incinerated at a Waste Incineration Plant using Machine Learning Techniques"

verfasst von / submitted by

Robert Brunnsteiner, BSc

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien, 2022 / Vienna, 2022

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	UA 066 910
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Masterstudium Computational Science UG2002
Betreut von / Supervisor:	Univ.-Prof. Dipl.-Ing. Dr.-Ing. Moritz Grosse-Wentrup

Abstract

Municipal solid waste incineration plants are a cornerstone of waste management in developed countries. They are able to destroy organic pollutants on the one hand and concentrate inorganic pollutants on the other hand, reducing the required landfill volume by about 90 per cent. Hence, they are the main means of treatment for residual waste which cannot be recycled in a meaningful way.

In contrast to most other incineration plants, they have to deal with a wide variety of different fuels (i.e., the input wastes) that influence the incineration process. A major problem is that the composition of wastes cannot be accurately determined prior to their incineration, as visual inspection can only detect major abnormalities and the results of chemical analyses are usually only available after one or more days.

Hence, in the past couple of years, many machine learning approaches have been developed to predict properties of the waste, mainly its *lower heating value (LHV)*. However, those only predicted daily averages or the LHV of a specific fraction of the waste. Yet for the plant operator, it would be of utmost interest to predict the LHV of the waste that is actually being burnt inside the furnace in the near future, so that it is possible to react to potential spikes in time.

To this end, starting by using known predictors of the LHV—calendar data like the day of week or the calendar week and weather data like temperature or precipitation—I will go from the established prediction of daily averages to near future time series predictions using linear and non-linear regression models like ridge regression, support vector machines and random forests. Those can be shown to make solid predictions of up to 30 minutes into the future, however, from there on, they get worse very fast. They can be thought of only *continuing the trend* of the past LHV-curve.

An obvious shortcoming of this approach is, that none of those predictors carry information about the composition of the actual waste that is being burnt. To resolve that problem, I added images of the waste applied to the input chute of the furnace as predictors. This enhances the model considerably, as it provides information of the actual waste inside the furnace for up to 90 minutes into the future. Although the high dimensionality of the feature space in this model—an eighteen step time series of 480×320 images in three colour channels in addition to the numerical features—requires different approaches (i.e. special designs of artificial neural networks), the proposed model yields an extremely good accuracy for up to 80 minutes.

Contents

List of Figures

List of Listings

Abbreviations

MSW municipal solid waste

MSWI municipal solid waste incineration

WtE waste-to-energy

OTNOC other than normal operating condition

LHV lower heating value

HHV higher heating value

ANN artificial neural network

CNN convolutional neural network

RNN recurrent neural network

LSTM long short-term memory

ReLU rectified linear unit

SGD stochastic gradient descent

SVM support vector machine

RBF radial basis function

RF random forest

PCA principle component analysis

MAE mean absolute error

MSE mean squared error

FT Fourier transform

DFT discrete Fourier transform

DCT discrete cosine transform

FFT fast Fourier transform

i.i.d. independent and identically distributed

CV cross-validation

Notation

a or α	a scalar
$\boldsymbol{a}$ or $\boldsymbol{\alpha}$	a (column) vector
A	a matrix
$\mathbf{0}$	a (column) vector of all zeros
$\mathbf{1}$	a (column) vector of all ones
$\mathcal{X}$	a set
i	the imaginary unit
b	byte
bit	bit

Acknowledgements

First of all, I want to thank my supervisor Prof. Dr.-Ing. Moritz Grosse-Wentrup. In providing me the opportunity to work on a topic that I am very passionate about but is far from his main research interests, he was the one to make this thesis possible.

The second important person is Ing. Michael Görlich, team lead of the process engineering group at MVA Dürnrohr. As he knows every part of the plant by heart, he provided me with invaluable expert knowledge on what features might work for my models. Of course, I also have to thank the EVN Wärmekraftwerke GmbH for supporting my ambitions to work on this topic and allowing me to use its data. There, my special thanks go to its managing director Dipl.-Ing. Gernot Alfons.

Finally, this thesis—as well as the whole programme I was perusing the last two and a half years next to my full-time employment—would not have been possible without my family: I want to thank my parents, Sonja and Winfried, for their perpetual endorsement and—beyond everything else—my wife for her immense support (and the proof-reading) as well as my daughter for always putting a smile to my face: thank you Tini, thank you Leni!

1. Introduction

1.1 Scope of this work

Waste incineration originally arose from the need to treat waste in a way that reduces volume and handles sanitary hazards. Waste incineration plants are still the prime solution for the destruction of organic pollutants and pathogenic microorganisms as well as the concentration of toxic heavy metal compounds [1]. Moreover, modern municipal solid waste incineration (MSWI) plants are designed as waste-to-energy (WtE) plants: they recover the energy contained within the waste in the form of district heating, process heat or electricity, respectively [2].

Therefore, the energy content of the waste—measured as lower heating value (LHV)—is an important operational parameter for a WtE plant. Unfortunately, it is hard to measure directly. As municipal solid waste (MSW) is heterogeneous, representative sampling is complicated and the results may only be available after the waste has already been burnt. In order to solve this problem, several machine learning methods have been proposed to predict the LHV of the waste.

One approach for the prediction of the LHV of a specific MSW sample is to use its physical composition as predictors. In a study by Ozveren, waste was classified by its content of food, paper, plastic, textile and wood and its LHV was predicted using artificial neural networks (ANNs) [3]. In a similar way, it has been shown that the LHV[1] can be predicted from the elemental composition of waste using ANNs and support vector machines (SVMs) [4], [5].

A different concept for predicting the LHV is to use operational parameters of the plant as predictors, such as feeding rate of MSW, flue gas temperature at different stages of the process and steam parameters of the boiler. This has been done by You, Ma, Tang, *et al.*, where they used ANNs, SVMs and random forests (RFs) for heating value prediction. They also grouped the LHVs into categories, thus not doing regression analysis but classification [6]. Yet another approach is to use environmental parameters like temperature, precipitation and wind speed to predict the LHV of waste. The reasoning behind this is the fact that the moisture content of wastes is strongly related to its LHV, as water has to be evaporated during combustion. This can be done with continuous data rather than discrete samples and has been done by Birgen, Magnanelli, Carlsson, *et al.* using Gaussian process regression (GPR) [7].

[1] In some studies, the higher heating value (HHV) is used instead of the LHV. They differ only in whether the combustion product water is in liquid (HHV) or gaseous (LHV) phase and thus, in principle, contain the same information.

However, all those approaches suffer from at least one of two shortcomings when it comes to the prediction of the LHV for optimising the WtE plant operation: first, some of the predictors like elemental composition cannot be obtained in time before the waste is already burnt and second, all of the aforementioned studies either used unrelated samples or daily averages as data, which makes them unsuitable for online prediction.

In this thesis, I will show that it is possible to make near-future forecasts of the waste's LHV in time to be relevant for WtE plant operation using only sensor data that is available online. For this purpose, it is necessary to use time series data that has a much larger sample size than the one used in previous studies - tens to hundreds of thousands instead of the maximum 1024 used by Birgen, Magnanelli, Carlsson, *et al.*

Additionally, I will demonstrate that it is possible to use images of the waste that is about to enter the furnace to predict the future observed LHV, therefore increasing the feature dimensionality to the millions. This allows the plant operator to get extremely accurate predictions of the LHV for an extended period of time, corresponding to the time frame in which the waste captured on the images is inside the furnace.

1.2 Overview of the MVA Dürnrohr

There are waste incineration plants all over the world, yet all of them are different in minor and sometimes even major ways. All the experiments are performed with data from the MSWI plant in Dürnrohr, Lower Austria, operated by EVN Wärmekraftwerke GmbH, which is called *MVA Dürnrohr*. As there can be limitations in the applicability of this work's results to other plants, I will start by giving a brief description of the plant, that is also pictured in figure 1.1.

Waste is delivered to the MVA Dürnrohr mostly by train, but also—to a lesser extent—on the road. There is a small pre-bunker, mainly for logistic purposes, but in the end, all the waste delivered to the MVA Dürnrohr ends up in the main bunker, which can hold up to 25 000 cubic metres of waste. Every year, about 500 000 tons are treated in the plant. Unlike many urban MSWI plants, which are tightly integrated in its city waste management system, the MVA Dürnrohr receives wastes from various sources: the main fraction is the municipal solid waste stemming from communes all across Lower Austria, but there are also other fractions, like residues from mechanical waste treatment or municipal solid waste like industrial wastes. To a much lesser extent, mono-fractions like insulating boards and tyres are also treated. Therefore, the input wastes tend to vary by a much larger extent than in urban MSWI plants.

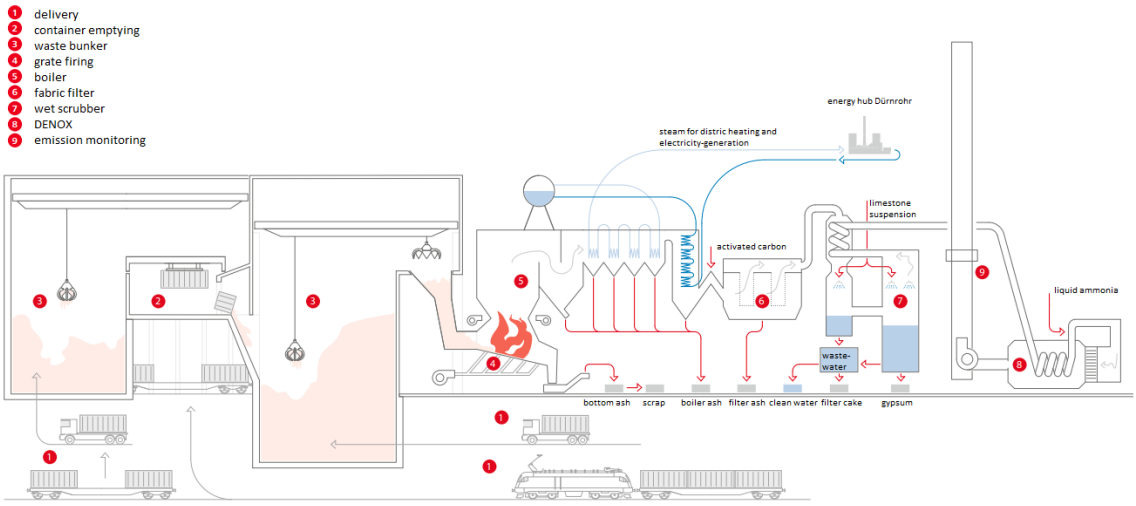


Figure 1.1: Overview of the MVA Dürnrohr.

From the main bunker, waste is applied to one of the three incineration lines via the input chute. From there, it takes about 30 minutes to enter the furnace, where it is incinerated while moving forward on the grate. This process takes up to one hour, depending on the LHV of the waste. Two of the three incineration lines are identical and have a thermal output of 60 MW, while the third, newer, line has a thermal output of 90 MW. The process technology of all three lines is nearly identical: in the boiler part, water is vaporised to steam at 390°C and 50 bar in three stages (economiser, boiler, superheater). That steam is then used for district heating, as process steam for industry and for electrical power generation. In the flue gas cleaning part of the plant, pollutants are removed so that the strict emission limits for waste incineration plants can be met. In the first stage, a fabric filter removes solid particles as well as pollutants bound to them. The second stage consists of two wet scrubbers, where acidic compounds and further pollutants are removed. In the last stage, nitrogen oxides are catalytically reduced to water and nitrogen. Finally, the residues of the plant are separately stored and disposed, according to their differing amounts of toxicity.

2. Theoretical Background

2.1 Ridge Regression

Linear regression tries to find an optimal line $\boldsymbol{w}^T\boldsymbol{x} + b$ in a sense that the squared error between the samples and the line is minimised, i.e.

$$\min_{\boldsymbol{w}} \frac{1}{m}\sum_{i=1}^{m}(\boldsymbol{w}^T\boldsymbol{x_i} - y_i)^2 \tag{2.1}$$

where the bias b is integrated in the $d+1$ dimensional weight vector $\boldsymbol{w}$. This *convex* optimization problem can be solved analytically by setting the gradient to zero and can be written as the linear system

$$\begin{gathered} A\boldsymbol{w} = \boldsymbol{b} \quad with \\ A = \sum_{i=1}^{m}\boldsymbol{x_i}\boldsymbol{x_i}^T \; and \; \boldsymbol{b} = \sum_{i=1}^{m} y_i\boldsymbol{x_i}. \end{gathered} \tag{2.2}$$

which can be solved for $\boldsymbol{w}$ [8, p. 138]. However, as the features are often correlated, this matrix A can be close to singular. Penalising the optimisation problem with the squared weights of the coefficients, i.e.

$$\min_{\boldsymbol{w}} \frac{1}{m}\sum_{i=1}^{m}(\boldsymbol{w}^T\boldsymbol{x_i} - y_i)^2 + \alpha\sum_{j=1}^{n} w_n^2 \tag{2.3}$$

leads to a less ill-conditioned coefficient matrix. This is called ridge regression and will often help reduce overfitting [9].

2.2 Support Vector Machines

SVMs have originally been designed to find a hyperplane that separates a linearly separable set of training points in an "optimal" way. This is formalised by maximising the margin, i.e. the distance of the hyperplane to the closest data points, the *support vectors* [10] (see figure 2.1).

Every hyperplane $(\boldsymbol{w}, b)$ that separates a training set $\mathcal{S} = (\boldsymbol{x_1}, y_1), ..., (\boldsymbol{x_n}, y_n)$ with the d-dimensional feature vector $\boldsymbol{x_i}$ and the label y_i being either +1 or -1 can be written as $y_i(\boldsymbol{w^T}\boldsymbol{x_i} + b) > 0$ [8, p. 167]. The maximum margin hyperplane can

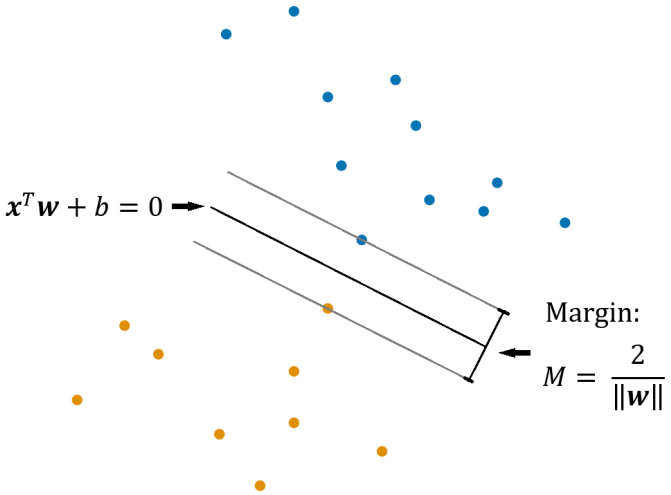


Figure 2.1: Concept of a *hard-SVM* classifier. Based on [11, p. 418].

be found by solving the optimisation problem

$$\begin{aligned} &\min_{\boldsymbol{w},b} \|\boldsymbol{w}\|_2 \\ s.t. \quad &y_i(\boldsymbol{x_i}^T\boldsymbol{w} + b) \geqslant 1 \quad \forall i, \end{aligned} \tag{2.4}$$

which has a quadratic objective function and a linear inequality constraint—it is a *quadratic program* [11, p. 418].

However, very often the training set is not linearly separable. One possibility to relax the strict margin assumption of this *hard-SVM* is (besides the kernel trick I will introduce in 2.2.1) to allow the misclassification of some data points. This is done by introducing some slack variables $\boldsymbol{\xi}$—the optimisation problem

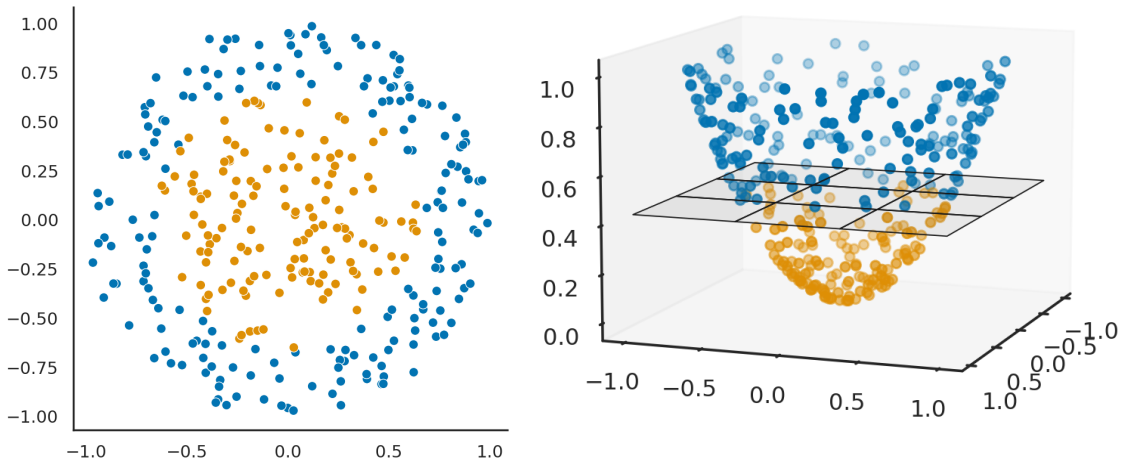
$$\begin{aligned} &\min_{\boldsymbol{w},b} \|\boldsymbol{w}\|_2 \\ s.t. \quad &y_i(\boldsymbol{x_i}^T\boldsymbol{w} + b) \geqslant 1 - \boldsymbol{\xi_i} \quad \forall i \; and \\ &\boldsymbol{\xi_i} \geqslant 0, \; \sum \boldsymbol{\xi_i} \leqslant const. \end{aligned} \tag{2.5}$$

is known as the *soft-SVM* [12], [11, p. 419].

Without going into more detail, it is important to state that for the computation of SVMs, usually the dual of the problem in the form of

$$\begin{aligned} &\min_{\boldsymbol{\alpha}} \frac{1}{2}\boldsymbol{\alpha}^T(y_i y_j \boldsymbol{x_i}^T\boldsymbol{x_j})\boldsymbol{\alpha} - \mathbf{1}^T\boldsymbol{\alpha} \\ s.t. \quad &\boldsymbol{y}^T\boldsymbol{\alpha} = 0 \end{aligned} \tag{2.6}$$

with all α_i between 0 and some positive constant C is used. C is a tuning parameter and can be viewed as the strength of the regularisation, i.e. as $C \to \infty$ the *hard-SVM* case is approached. From the $\boldsymbol{\alpha}$ we optimise for, the desired $\boldsymbol{w}$ can be obtained



(a) (b)

Figure 2.2: Concept of the kernel trick. (a) The original data has two feature dimensions. It is easy to see that there is no separating hyperplane (i.e., a line) for the two classes, so the data is not linearly separable. (b) Once the data is mapped into a higher dimensional feature space—three dimensions in this case—a separating hyperplane (i.e., a plane) can be found, hence the data has become linearly separable.

via the primal–dual relationship [13]. This is also a *quadratic program*, but in this formulation the feature vectors $\boldsymbol{x}$ only enter the equation in the form of an inner product with each other. This property is very useful, as we will see in the following section.

2.2.1 Kernel Methods

As described above, SVMs are solid tools when applied to linearly separable datasets. However, most real world datasets are not. If we still want to use a linear classifier like an SVM, we can try to map our dataset into a higher dimensional *feature space*, in which our dataset becomes linearly separable. This idea is shown in figure 2.2 and is the main concept behind applying kernels in machine learning.

We still consider the dataset $\mathcal{S} = (\boldsymbol{x_1}, y_1), ..., (\boldsymbol{x_n}, y_n)$. A kernel k can now be defined as

$$\begin{aligned} &k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}, \\ &k(\boldsymbol{x_1}, \boldsymbol{x_2}) = \phi(\boldsymbol{x_1})^T \phi(\boldsymbol{x_2}). \end{aligned} \tag{2.7}$$

where ϕ is the *feature map*. It maps from $\mathcal{X}$ to an inner product Hilbert space $\mathcal{H}$ [14]. It is important to realise that the transformation to the (possibly high dimensional) feature space is not done explicitly but rather implicitly by computing the kernel.

In this work, I will often use the radial basis function (RBF) kernel

$$k(\boldsymbol{x_1}, \boldsymbol{x_2}) = exp\left(-\frac{\|\boldsymbol{x_1} - \boldsymbol{x_2}\|_2^2}{2\sigma^2}\right), \tag{2.8}$$

where σ is a tuning parameter [15]. While this approach is extremely useful for smaller datasets, constructing the kernel matrix is of runtime $\mathcal{O}(m^2)$, where m is the number of training samples and thus gets prohibitively slow for very large datasets [16, p. 149].

2.2.2 SVMs for Outlier Detection

SVMs can also be used in an *unsupervised* learning setting. We can compute a subset $\mathcal{S}$ of our dataset that has a specified probability to be drawn from the underlying probability distribution $\mathcal{P}$ [17]. This is done by solving the problem

$$\begin{aligned} & \min_{\boldsymbol{\alpha}} \frac{1}{2}\boldsymbol{\alpha}^T(y_i y_j \boldsymbol{x_i}^T \boldsymbol{x_j})\boldsymbol{\alpha} \\ s.t. \quad & 0 \leqslant \alpha_i \leqslant 1, \quad i = 1, ..., l \\ & \mathbf{1}^T\boldsymbol{\alpha} \geqslant \nu l, \end{aligned} \tag{2.9}$$

where ν is a parameter from a different formulation of 2.6. It has been proven to be an upper bound on the fraction of training errors and a lower bound of the fraction of support vectors [18].

2.2.3 SVMs for Regression

The expansion of the SVM idea from classification to regression is conceptually quite straightforward but a little daunting on the mathematical side. As we change from $y \in \{-1, +1\}$ to $y \in \mathbb{R}$, we have to allow some arbitrarily small but non-zero deviation ϵ from the true value for all y. In the most basic case we want to solve the optimisation problem

$$\begin{aligned} & \min_{\boldsymbol{w}, b} \|\boldsymbol{w}\|_2 \\ s.t. \quad & y_i(\boldsymbol{w}^T\boldsymbol{x_i} - b \leqslant \epsilon) \; and \\ & \boldsymbol{w}^T\boldsymbol{x_i} + b - y_i \leqslant \epsilon). \end{aligned} \tag{2.10}$$

This can of cause only be done if there is a solution to 2.10, i.e. all datapoints lie within ϵ [19]. To relax that assumption we have to introduce a slack variable ξ as we did in 2.5. Furthermore, in order to be able to use kernels, the problem has to be formulated in its dual form. I will omit the math, but it can be found in [19] or [13].

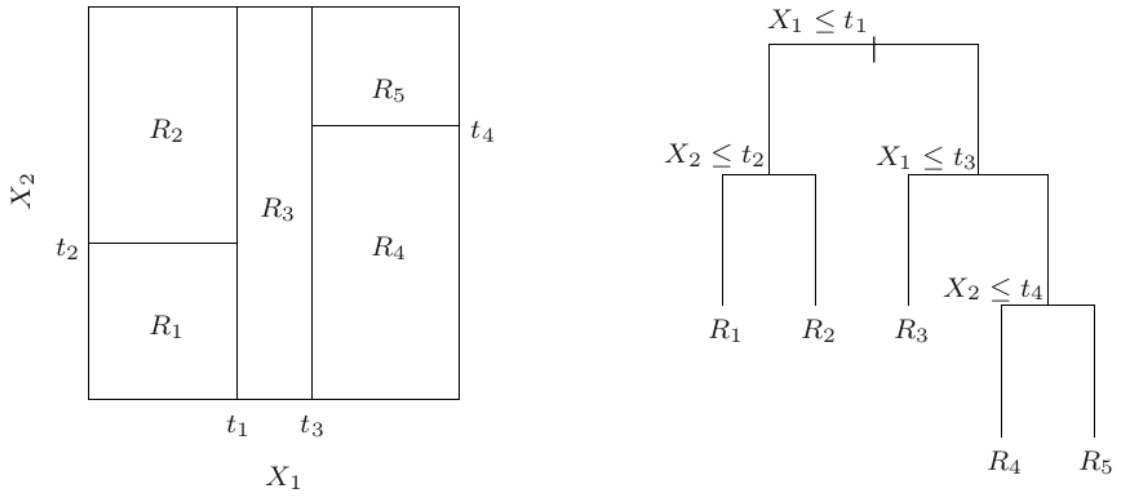


Figure 2.3: A decision tree with $d = 2$. Each split is done on one feature with a specific threshold. Each leaf of the tree corresponds to a *box* in feature space. Modified from [11, p. 306].

2.3 Decision Trees

Decision trees are conceptually easy yet powerful predictors. They predict the label of a sample by traversing a tree from its root to one of its leaves. Each split is done on one of the feature dimensions x_i at a certain threshold θ (see figure 2.3). Growing the tree until all leaves are pure, i.e. there is only one training sample in each leaf, yields a very powerful predictor [8, pp. 212–213]: in the case of classification, it can be easily seen that the number of leaves is equal to the VC-dimension [20]. As this will often lead to substantial overfitting, a stopping criterion may be applied and the tree grown to only a limited size [8, p. 213].

2.3.1 Decision Tree Regression

When performing regression analysis with decision trees, we want to predict the continuous label $y \in \mathbb{R}$ by dividing the feature space $\boldsymbol{x} \in \mathbb{R}^d$ in such a way, that it can be constructed and visualised as a binary tree. This has the positive side effect, that it is—in contrast to most other regression models—quite easy to interpret. In each of the obtained *boxes*, the prediction is a constant value c_m, where m is the number of leaves. When growing the tree, the best splits and thresholds have to be found. One common criterion for that is the mean squared error (MSE) [11, p. 307]. To make this optimisation problem computationally efficient, various algorithms have been developed. For this work, I will use the scikit-learn implementation [21] of the CART algorithm [22].

2.3.2 Random Forests

Decision trees often suffer from *instability*, i.e. they are very sensitive to small changes in the training samples and may create vastly different trees in such cases. A common approach to solve this problem and obtain better results is to train ensembles of trees on different subsets [11, p. 312]. One such method is called *random forests* [23]. There, each decision tree is grown using a random vector $\boldsymbol{\theta}$, which determines which subset of the samples (with replacement) and features is drawn for each tree. Each vector $\boldsymbol{\theta}$ is independent from the others, but drawn from some distribution (e.g. a uniform distribution) [8, p. 217], [23].

2.4 Artificial Neural Networks

ANNs as we think of them today originated in the creation of the Perceptron by Rosenblatt in the late 1950s, which was inspired by human brain neurons and was trainable with a simple algorithm [24]. This is why ANNs are often called multi layer perceptrons—or MLPs for short. As we will see later, they often feature non-linearities and yield highly non-convex optimisation problems. Those can be solved by using variations of gradient descent (see section 2.5). A way to efficiently compute the gradient of the loss function—in regression, this is usually the MSE—with respect to the weights is called *backpropagation* [25]. The main idea is, that during training, the input vector is propagated forward through the network, yielding the cost. From there, the gradient is calculated backward, layer to layer, applying the calculus chain rule [16, pp. 197–205]. It has been shown, that—although we do not have the mathematical guarantees from convex optimisation—those algorithms will yield solid results in practice [16, p. 148].

2.4.1 Feedforward Neural Networks

The most straightforward variant of an ANN is a feedforward neural network. It is comprised of an input layer consisting of one neuron—or *unit*—per input variable, an output layer consisting of a number of units equal to the number of desired output variables and any number of *hidden layers* in between them. The number of layers is called the network's *depth* and the number of units per layer is called the networks *width*. To introduce non-linearities into the model, we perform an *activation function* g on the affine transformation, i.e.

$$\boldsymbol{h} = g(W^T \boldsymbol{x} + \boldsymbol{b}), \tag{2.11}$$

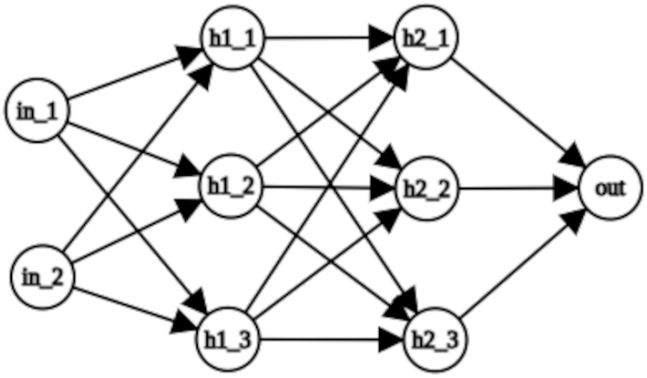


Figure 2.4: Each node in this directed graph is a unit and each edge is a weight. As each node of one layer is connected to all the nodes of the neighbouring layers, it is called a densely connected—or simply *dense*—layer.

where $g(z)$ is often a rectified linear unit (ReLU), i.e. $max\{0, z\}$, but can also be the sigmoid function $\sigma(z)$, the hyperbolic tangent $tanh(z)$ or others [16, pp. 163–191].

Feedforward ANNs—especially densely connected ones as shown in figure 2.4—are often used as part of a more elaborate model, specifically just before the output layer, to bring other parts of the network together.

2.4.2 Convolutional Neural Networks

Convolutional neural networks (CNNs) use *a priori* knowledge of the data to construct a neural network that has multiple advantages over a typical densely connected feedforward network. It can be used for structured data that is *translational invariant* and is well suited for image recognition [26]. It uses the mathematical operation of convolution (denoted by an asterisk) to combine the *input image* (I) with the *kernel* (K) to create a *feature map* (S), i.e.

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) \tag{2.12}$$

where the kernel is much smaller than the input image[1]. This can be thought of as the kernel scanning over the image and leads to translational invariance and also a massive reduction in the free parameters, when compared to densely connected layers [16, pp. 322–324]. This network can also be trained by variants of stochastic gradient descent (SGD) using backpropagation, where the same kernel parameters appear on multiple locations of the computation matrix—this is why the reduction of parameters is often also referred to as *parameter sharing*. It is also a kind of regularisation: while huge densely connected neural networks will have very low

[1]Implementations of the convolution operation for CNNs often use a related operation, the *cross-correlation* instead of the convolution, which exchanges the minus signs in equation 2.12 with a plus sign.

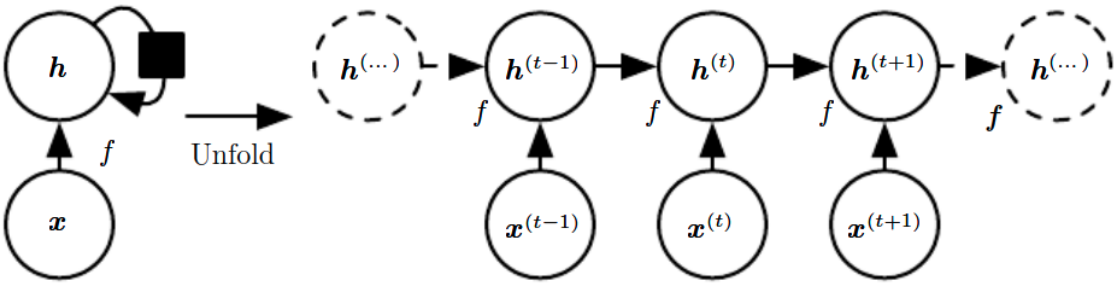


Figure 2.5: On the left, an RNN is visualised with a cyclic graph, where the black square indicates a delay of one time step. On the right, the graph is unfolded. Both graphs do not take outputs into account. Taken from [16, p. 366].

training errors by simply remembering the training data, they do not generalise well. In contrast, CNNs might have training errors that are a little higher but validation errors very close to them [26].

2.4.3 Recurrent Neural Networks

Recurrent neural networks (RNNs) [27] are neural networks that are designed to handle special kinds of structured data, namely sequence data in the form of $\boldsymbol{w}^{(1)}, ..., \boldsymbol{w}^{(\tau)}$. In a similar vein as it is the case with CNNs (see section 2.4.2), RNNs can make use of *parameter sharing* [16, p. 367] and also be efficiently trained by *backpropagation* [27]. It allows the network to generalise independent of the position of the input within the sequence.

This is done by allowing the *network graph* to form cycles, instead of the directed, acyclic graph of feedforward neural networks. This means, that—within a given hidden layer—the activations come from both the current external input and the hidden layer state of the previous time step [28, pp. 22–23], i.e.

$$\boldsymbol{h}^{(t)} = f(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta}), \tag{2.13}$$

where $\boldsymbol{\theta}$ is the RNN's parameterisation. This can be visualised as a cylcic graph, but also—for a specific number of time steps—as an unfolded graph and is shown in figure 2.5 [16, p. 366].

An immense problem for the optimisation of deep networks that apply the same weights at each layer—which is exactly what RNNs do—is that the gradient tends to *vanish* or *explode*. This comes to no surprise: applying the same weight matrix W for t times is equivalent to applying W^t once. From the eigendecomposition $W = V\ diag(\lambda)\ V^{-1}$, we can clearly see that the gradients are scaled by $diag(\lambda)^t$, that is, they explode for eigenvalues larger than one in absolute value and vanish for eigenvalues smaller than one in absolute value [16, p. 288]

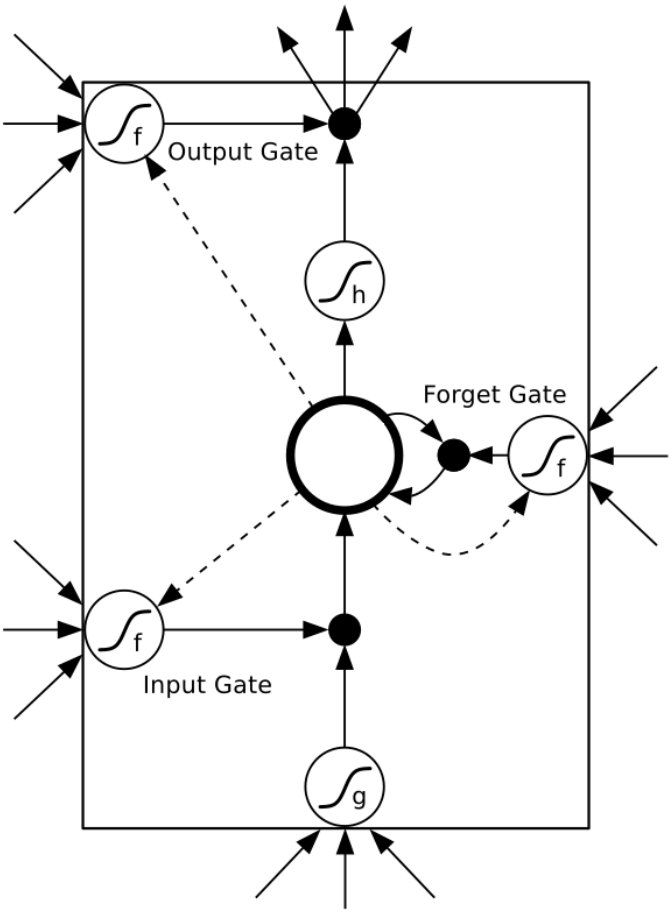


Figure 2.6: An LSTM cell. The incoming activations are passed through various gates: the *input gate*, the *forget gate* and the *output gate*. The gates' activation function is usually the sigmoid function. From there, they are collected via multiplication (pictured as black circle). The weighted "pass-through"-connections are pictured as dashed lines. Modified from [28].

A solution to this problem is to allow the gradient to pass through time uninhibited along certain paths. This idea leads to the formulation of the long short-term memory (LSTM) model [29], that I will use throughout this work. An LSTM usually has multiple LSTM cells (see figure 2.6). Via the multiplication with the gate value, which can be thought of as opening or closing the gate, as it uses a sigmoid activation function, it is possible to pass information through the RNN over many time steps without facing the *vanishing gradient* problem. Additionally, it is also possible to forget information from a past time step by a different configuration of the gates [28, pp. 37–43].

2.5 Stochastic Gradient Descent

Learning algorithms need to find the optimum of its *objective function*. This can sometimes be done analytically, like for ridge regression (see section 2.1), but often,

iterative methods are used. Standard (non-stochastic) gradient descent calculates the *gradient* of the objective function (sometimes—especially in deep learning—also called *loss function*) and then modifies the weight vector $\boldsymbol{w}$ by moving in the opposite direction of the gradient, starting at $\boldsymbol{w}^{(1)}$ either $\mathbf{0}$ or some small non-zero value vector, i.e.

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta \nabla f(\boldsymbol{w}^{(t)}), \tag{2.14}$$

where the gradient $\nabla f(\boldsymbol{w})$ is $\left(\frac{\partial f(\boldsymbol{w})}{\partial w_1}, ..., \frac{\partial f(\boldsymbol{w})}{\partial w_d}\right)^T$ for a function $f : \mathbb{R}^d \to \mathbb{R}$ and η is the step length or *learning rate* [8, p. 151].

For larger training sets, however, it is often infeasible to calculate the gradient on the whole dataset, and therefore, a stochastic variant [30] is applied.

The main idea behind SGD is that the gradient of the whole training set is an expectation that can be estimated reasonably well by a small subset, ranging from one to a few hundred samples. For large training sets, this decreases the runtime of the algorithm tremendously, because the batch size stays fixed: it is only $\mathcal{O}(1)$ instead of $\mathcal{O}(m)$, where m is the size of the training set.

For all variants of gradient descent, it is important to note that there is no guarantee that it will find a local optimum in reasonable time or—for non-convex optimisation problems like in deep learning—a minimum that is close to the global minimum. In practice, however, SGD will often find very low values of the cost function very fast, and is therefore of great use [16, pp. 148–149].

A short note on the terminology: sometimes, only the variant with one sample is called stochastic gradient descent, whereas the variant for larger subsets—called *minibatches*—is called minibatch gradient descent. Furthermore, non-stochastic gradient descent is sometimes called batch gradient descent. To avoid confusion, I will call every variant that uses a subset of the training samples stochastic gradient descent (or SGD), while I will call the variant that calculates the gradient on the whole dataset simply gradient descent.

2.6 Low-Pass Filter and Discrete Fourier Transform

All signals can be thought of as being a linear combination of signals of a pure frequency. To obtain those frequencies, the signal must be transformed from the time domain to the frequency domain by the Fourier transform (FT). As signals obtained from a measurement are always discrete in time, the discrete Fourier transform

(DFT)[2] can be applied for this transformation, i.e.

$$X(n) = \frac{1}{N} \sum_{j=0}^{N-1} x(j) e^{\frac{-2\pi i n j}{N}}. \tag{2.15}$$

From there, all frequency components above a certain frequency can be set to zero—a low-pass filter is applied. After applying the inverse DFT

$$x(j) = \sum_{n=0}^{N-1} X(n) e^{\frac{2\pi i n j}{N}}, \tag{2.16}$$

the filtered signal is obtained [31].

Of course, this procedure only works if the whole signal is available at the time of processing. In an online setting it would be necessary to design filters in the time domain, but this is beyond the scope of this work.

Furthermore, in practice the DFT is not computed by applying the equations from above, but using an algorithm called fast Fourier transform (FFT). It improves the runtime of the naive DFT from $\mathcal{O}(n^2)$ to $\mathcal{O}(n \log n)$, making it viable to process much longer time series [32].

[2] When working with computers, both the time domain and the frequency domain representation of the signal has to be discrete.

3. Methods and Results

3.1 Computational Environment

All experiments have been performed on a machine with a twelve core Intel core i7-12700K CPU featuring eight hyper-threaded cores that clock at 4700 MHz and four single-threaded cores clocking at 3600 MHz. Hence, if there are tasks that can be parallelised, this machine is especially efficient if this can be done using multiples of twelve or twenty. It has access to 32 Gb of DDR4 memory clocking at 3600 MHz and features a 25 Mb top level cache.

The GPU of this machine is a Nvidia GeForce RTX 3070 with 5888 cores clocking at 1730 MHz with 8 Gb of GDDR6 video memory with a clock speed of 7000 MHz. It can be utilized as a general purpose GPU using Nvidia CUDA.

Most computations—including all classical methods implemented in scikit-learn [21]—are run on the CPU, while neural network training using keras [33]/tensorflow [34] is performed by the GPU.

3.2 Time Series Data Preprocessing

From the MSWI plant in Dürnrohr (MVA Dürnrohr), time series data on several parameters that are known to be predictors of the waste's LHV [7] are obtained: the day of the week, the calendar week and the outside temperature, wind strength and humidity. Additionally, the air pressure is obtained as well as the volume of waste in the two bunkers of the MVA Dürnrohr: the pre-bunker and the main bunker.

In contrast to MSWI plants in urban areas, the input streams to the MVA Dürnrohr are not as steady and uniform. Hence, there is no fixed time span between the delivery of the waste and its incineration. Consequently, the bunker values—and especially their change rate—can hint to the age of the waste being burnt.

Furthermore, as time series data with a sampling frequency of $\frac{1}{5\ min}$ is obtained instead of the daily data used by Birgen, Magnanelli, Carlsson, *et al.*, also the hour is used as a predictor. Additionally, as the MVA Dürnrohr has three incineration lines which are all fed from the same main bunker, the LHV of all the three lines is obtained to be used as a possible label.

3.2.1 Outlier Detection

Time series data was obtained from 1st January 2018 until 31st March 2022, totalling 446 689 measurements for all features. In this time span, a lot of other than nor-

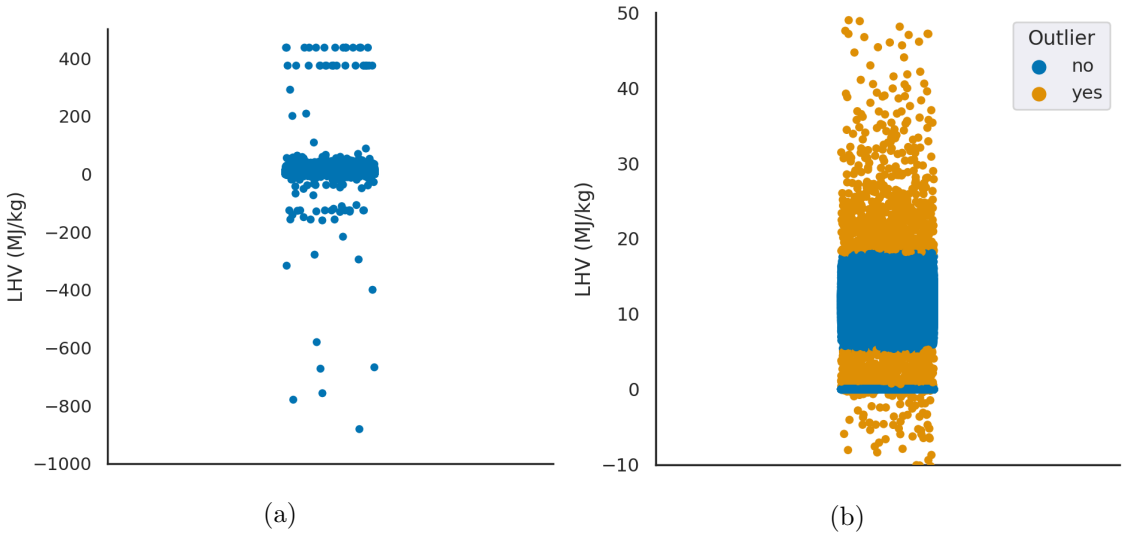


Figure 3.1: LHV (in MJ/kg) for incineration line 2. (a) The distribution of all 446 689 points. Reasonable values should not be below 0 and above 30. However, there are data points above 400 and below -900, although well over 99,9% of all points are within the main band. (b) 1 402 outliers are detected by this approach. All other values are located in a central band around the 10 MJ/kg region. However, there is also a band at exactly 0.0 MJ/kg, that is not classified as an outlier. Those values are either from plant standstills or sensor malfunctions and have to be handled separately.

mal operating conditions (OTNOCs) occurred: there are two planned maintenance shutdowns each year for every line and occasional breakdowns due to technical problems. The latter include waste related malfunctions, such as melting metals that can incapacitate the movement of the grate or the malfunction of plant components due to deterioration. Also, sensors sometimes yield wrong readings—either due to malfunctions or due to exceeding their measurement range. As the large dataset makes cleaning it by hand not feasible, an automated way to deal with them has to be applied.

Data processing was done in python using both numpy [35] and pandas [36] for data handling. Outlier detection was performed by one-class SVM using a RBF kernel as outlined in section 2.2.2 (see figure 3.1). As SVMs scale very unfavourably with a large sample size (at best quadratic in the number of samples), a one-class SVM based on SGD is used together with the Nystroem RBF kernel approximation. The Nystoem approximation uses a γ value of 0.2 and 500 components to create the mapping and the SGD one-class SVM uses a ν of $5 \cdot 10^{-3}$. For most of the feature- and label-vectors, this leads to less than 0.1% of the samples being classified as outliers. Computation has been done using the implementation of scikit-learn [21].

To compare this dataset from the MVA Dürnrohr and my LHV prediction approach to the one of Birgen, Magnanelli, Carlsson, *et al.*, which used daily averages, I calculated those averages from the preprocessed time series data, only keeping *valid* averages if all samples of a given day have non-outlier, non-zero values in all

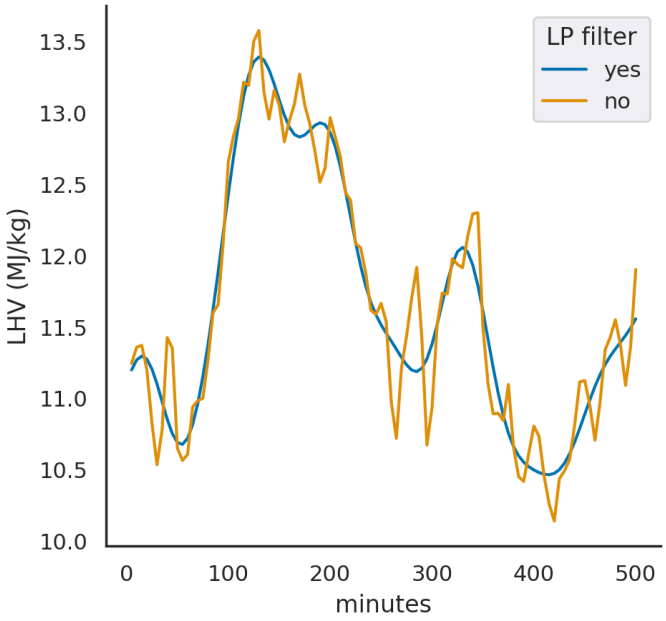


Figure 3.2: The LHV data has been sampled with a frequency of $\frac{1}{5\ min}$. By applying a low-pass filter with cut-off frequency $\frac{1}{30\ min}$, the mostly non-desired high frequency spikes can be removed. This figure shows an arbitrary 500 minute section of the data.

features and labels. From the 1 552 days of the data, 904 have been kept.

For the short-time LHV prediction that will be described in section 3.4, some more preprocessing is needed: outliers in the features were replaced with the mean of the respective feature. All the samples containing an outlier in the label are discarded, as replacing them with the mean would lead to an overestimation of the prediction quality. The same is done for an LHV of exactly 0.0 MJ/kg, as those correspond to the standstill of the incineration line or a malfunction of the sensor.

3.2.2 Application of a Low-Pass Filter

In this work, I will apply a low-pass filter (see section 2.6) for two different purposes.

The first one is to apply a low-pass filter with a cut-off frequency of $\frac{1}{30\ min}$ to the label (see figure 3.2). The reason for this is twofold: first, with the chosen features, the prediction of short-term fluctuations in the LHV is not feasible, and second, it is also not necessary—the prediction of the low frequency components of the LHV-curve gives enough information to the operator.

The other application of a low-pass filter is downsampling. In section 3.5, I will use data with a sampling frequency of $\frac{1}{1\ min}$ but create samples with time steps of 5 minutes. To avoid aliasing, all frequencies above the Nyquist-frequency have to be removed prior to the downsampling [37].

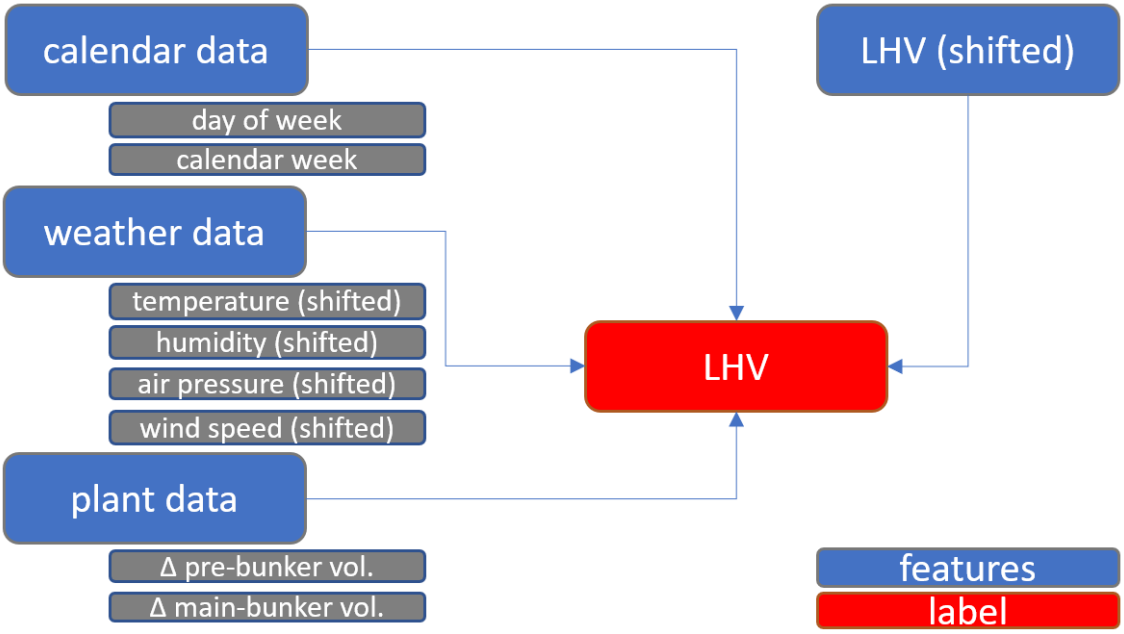


Figure 3.3: Design of the training set for the prediction of daily averages for the LHV.

3.3 LHV Prediction with Daily Averages

For the prediction of daily averages, an average of the LHVs of the three incineration lines weighted by nominal thermal output has been created as label. From the pre- and main bunker volumes, the change to the day before is calculated and used as a feature. Finally, all features except the weekday and calendar week are shifted by one day, so that the value of the day before is used to predict the current LHV (see figure 3.3).

After that, a possible correlation of the features with the label is inspected visually, by plotting them against each other. Such a correlation can be seen, however, it is also clear that it is not extremely strong. Therefore, predicting the LHV from them will be a challenging task (see figures 3.4 and 3.5. Additional correlations are presented in appendix C, figures C.1, C.2, C.3 and C.4).

Before starting with the predictions, some preprocessing steps specific for this task have to be done. First, missing entries created by the calculation of differences and the shifting operations are replaced with the mean of the feature. The same is done for values that are exactly 0.0, as they are most likely outliers that cannot be detected by the outlier detection described in 3.2. Second, all feature vectors are scaled to unit variance with a mean of zero.

For the regression analysis, the data was kept in order to account for the fact that it is not necessarily independent and identically distributed (i.i.d.), as the underlying distribution might change over time. Training was done on the first 70% of the data, while validation was done on the last 20%, discarding the data in between. All the

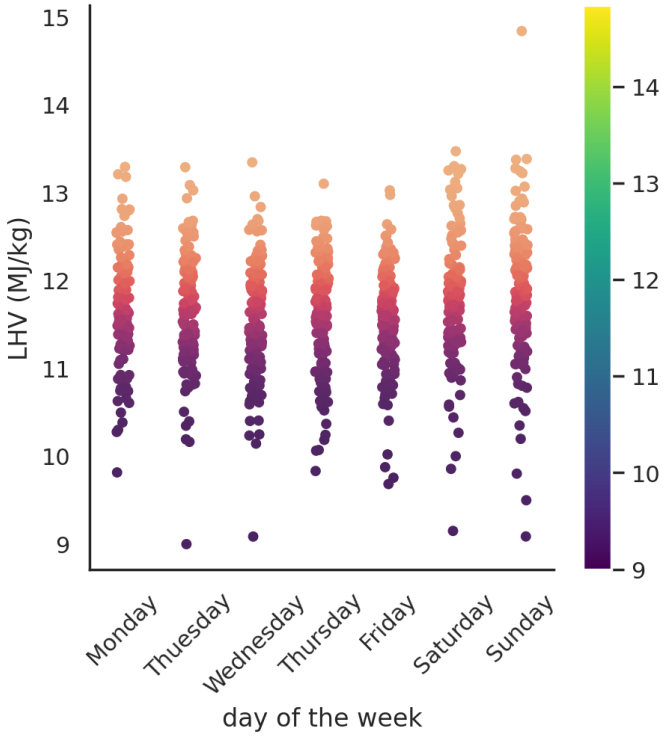


Figure 3.4: A correlation of the LHV over the different weekdays can be seen, with lower LHVs in the middle of the week.

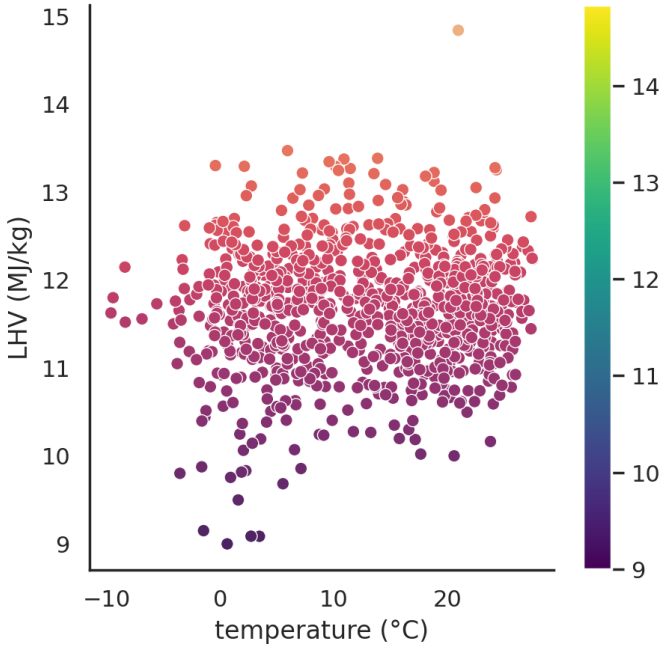
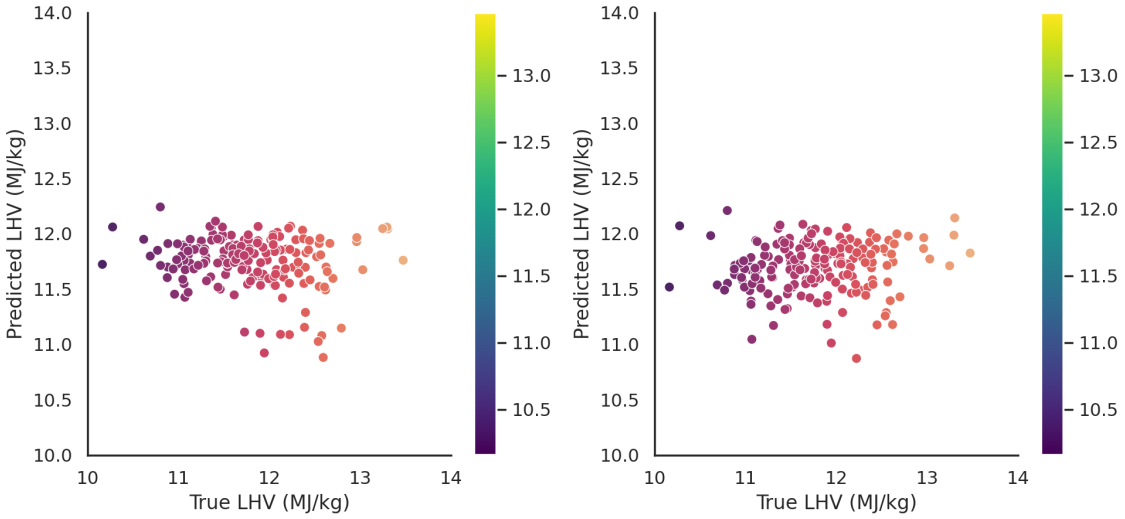


Figure 3.5: Lower LHVs can be found at relatively low outside temperatures more often than at higher ones, however, this is only a small correlation.



(a) Random Forest

(b) SVM

Figure 3.6: (a) shows the performance of the RF regressor, while (b) shows the performance of the SVM regressor, each for a single run.

above operations have been done with Python's scikit-learn package [21]. Prediction has been done by using both an RF regressor and an SVM regressor, as outlined in sections 2.3.2 and 2.2.3.

The RF regressor used 100 decision trees and the *squared error* as the criterion to find the best split. The SVM regressor used a C of 0.5, an ϵ of 0.15 and an RBF kernel. The implementations are from scikit-learn [21], but the SVM regression is actually computed by the LIBSVM library [13]. Averaging over 10 runs, I was able to obtain a mean absolute error (MAE) of $0.54 MJ/kg$ for the RF regressor and $0.50 MJ/kg$ for the SVM regressor, respectively. This compares favourably to the $0.59 MJ/kg$ that was obtained by Birgen, Magnanelli, Carlsson, *et al.* in 2021. Nevertheless, this prediction accuracy is not too impressive, which can be seen when visualising the prediction (see figure 3.6).

3.4 LHV Short-Term Prediction

From a plant operator's point of view, the behaviour of the installation in the near future is of utmost importance. Therefore, they are often mainly interested in the composition of the waste in the near future—and often especially in the LHV. The main idea now is to use the features described in the previous sections, however, in contrast to the last section, here I will not only use the value of the previous time point, as I did in section 3.3, but use a series of past measurements. Specifically, I chose to use eighteen time points, which corresponds to 90 minutes of time. That is about the time the waste needs from being administered to the chute leading

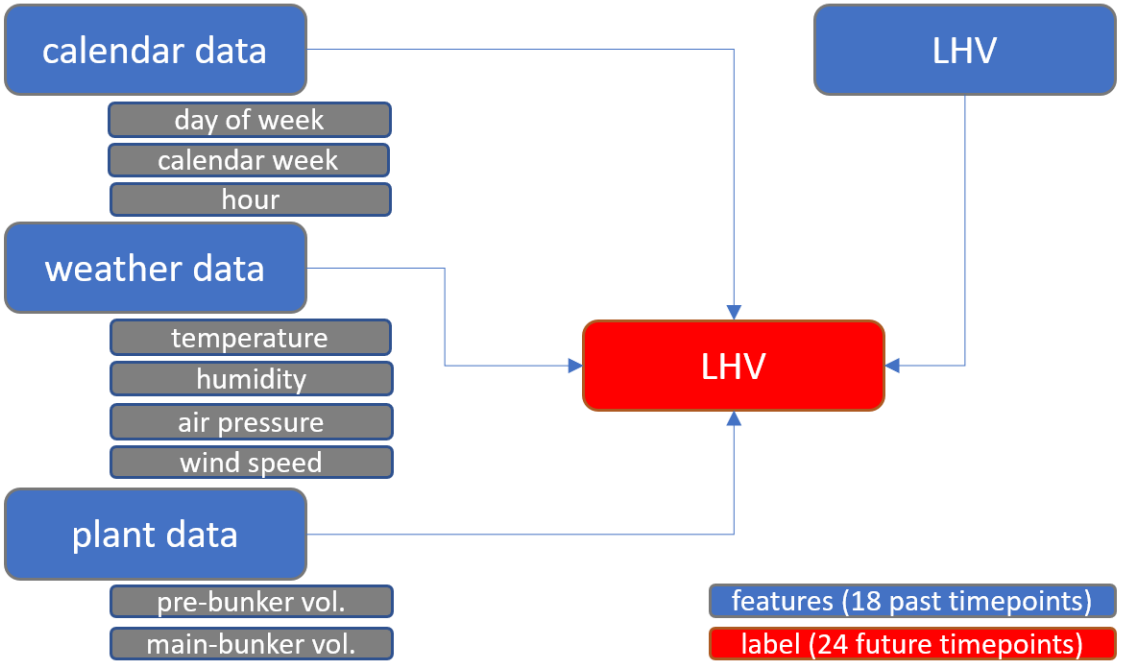


Figure 3.7: Design of the training set for the short-term LHV prediction without the use of images.

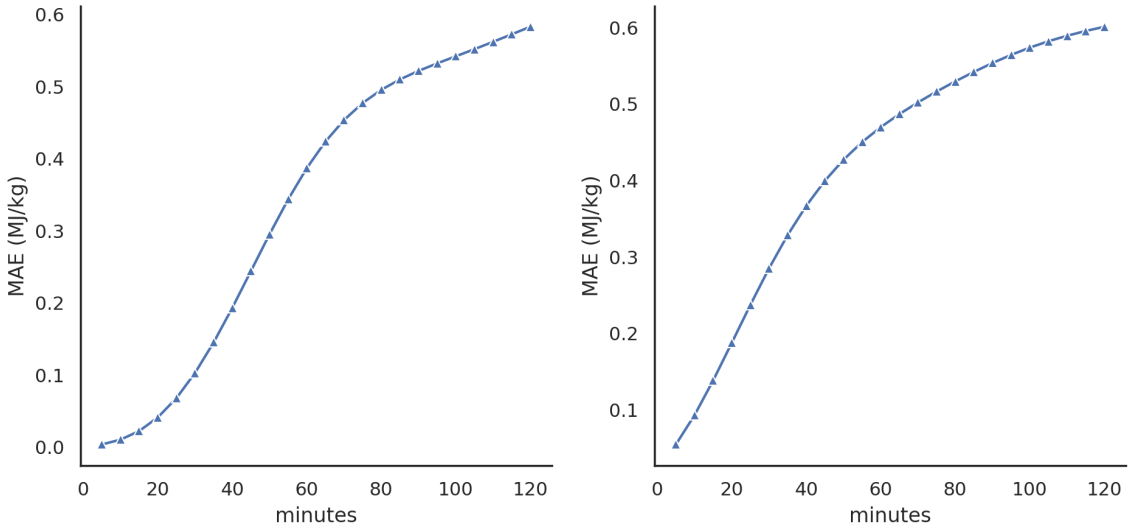
to the furnace until being fully incinerated. The target value is the LHV of the waste burnt in incineration line 2 of the MVA Dürnrohr in the next 24 time points, corresponding to 120 minutes (see figure 3.7).

For training and validating models, it is important to note that the data is a time series, so it is not i.i.d.. This means, that using standard cross-validation (CV) [38] or any splitting of shuffled data would lead to an overestimation of the model. Therefore, I use a variant of CV that preserves the order of the samples—I will call that *backtesting*. For testing, the size of the validation set is always 8 640 (one month) preceded by a gap of 8 640 samples and a variable training set size using a scikit-learn implementation [21] with ten splits.

3.4.1 LHV Prediction using Classical Regression Methods

For most regression models we can see a very good prediction accuracy for the next few time points, but it gets worse quite fast. However, when using ridge regression (see section 2.1) on the fully preprocessed dataset, the prediction accuracy stays quite low for a little longer, so that the MAE-curve resembles a sigmoid curve (see figure 3.8).

Although the prediction accuracy is largely dependent on how far into the future the predictions are made (see figure 3.9), I will use the average of the MAE for the predicted 24 future time points to compare the performance of different models and approaches.



(a) Ridge Regression

(b) Random Forest

Figure 3.8: (a) The overall best regression accuracy was obtained by using ridge regression with $\alpha = 1.0$ on the fully preprocessed dataset, with an average MAE of 0.34 for the training set of one year. Here, it can be seen that the error stays low for more future time points than in any other model. (b) RFs also obtained very high accuracies on this dataset, however, the error increases much faster, leading to an average MAE of 0.42.

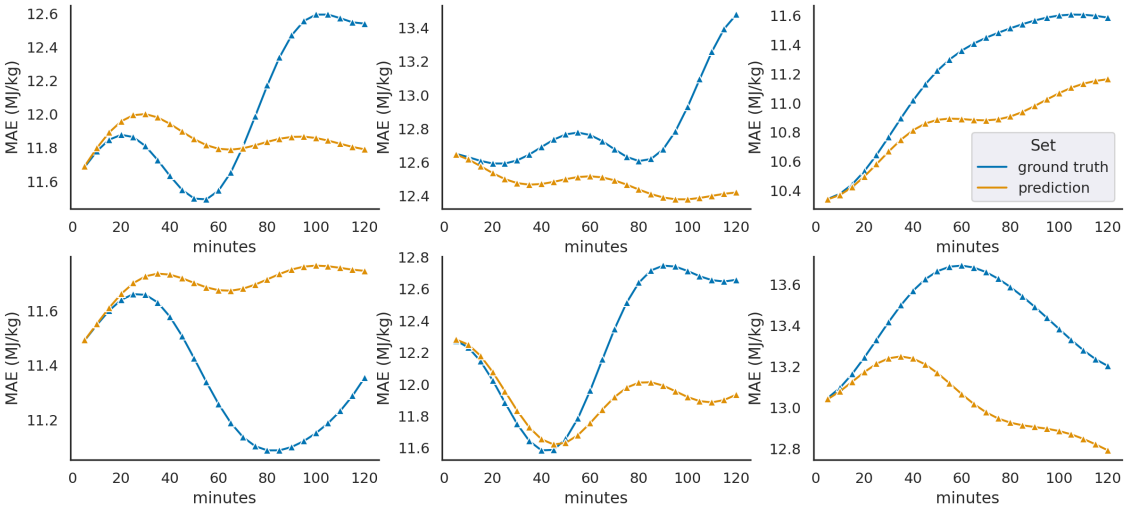


Figure 3.9: Arbitrarily chosen representative predictions of the ridge regression model with an average MAE between 0.33 and 0.35 MJ/kg. From them, it is easy to see that this model *continues the trend*, i.e. the first few time points resemble the ground truth quite well, but from there on, the accuracy decreases fast. Time points beyond about 30 minutes are in fact of no practical use for the plant operator.

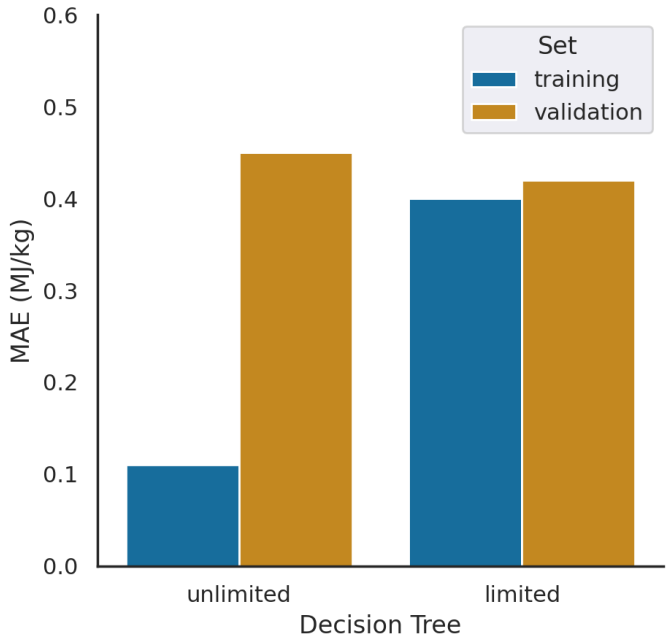


Figure 3.10: The MAE of an RF with twenty decision trees grown on the one year training set. The unlimited trees are grown on the full training set until the leaves are pure, resulting in 65 542 leaves on average. The limited trees are grown on half of the training set until the leaves contain no more than 0.1% of the training samples, resulting in only 301 leaves on average.

First of all, the reason for the good performance of the regularised, low complexity ridge regression lies in the fact that most other models tend to overfit on this dataset quite heavily. For example, the average decision tree in an RF fitted on the one year dataset has over 65 000 leaves. It has a tiny training error, however, it does not generalise well. If the trees are not allowed to grow to their full size, better validation errors—that are also much closer to the training error—can be obtained (see figure 3.10).

The superior accuracy of the ridge regression comes at a cost, however. If the data is not carefully preprocessed or we want to predict the LHV without the application of a low-pass filter, the models with higher complexity start doing better. The limited RF matches the ridge regression on the non-filtered data and outperforms it on the data that has not undergone outlier detection (see figure 3.11).

Another important factor of the model performance is of course the training set size. If it is very small, the (limited) RF outperforms ridge regression. However, with sizes of three months and more, ridge regression performs better than both the RFs and SVMs (see figure 3.12). The latter also suffers from its quadratic runtime scaling in the size of the training set. One split of training the SVM regressor for the 10-fold backtesting on the one year dataset would have taken over two hours—limited RFs only take about six seconds and ridge regression only 60 ms—and has therefore been omitted.

Figure 3.11: On the fully preprocessed data, ridge regression outperforms RF regressors. However, that is not the case for data that has not been carefully preprocessed. (RFs with twenty limited trees, ridge regression with a regularisation α of 1.0 and the one year training set.)

As ridge regression has a very good performance in the task at hand, there is one more thing to note: the learnt model is quite easy to interpret, as the coefficients provide information about the importance of the features. Unsurprisingly, the label itself from past time points—especially the very last one—carries most of the information for predicting future time points. However, all the other features have coefficients significantly larger than zero, too. They are comparable with each other in their values, including the bunker volume not used in the study of Birgen, Magnanelli, Carlsson, *et al.*

All models were created using the respective scikit-learn implementation [21].

3.4.2 LHV Prediction using RNNs

Neural networks are extremely powerful regressors, but, as we have seen in the previous section, this is not really needed—or even desired—in this task. I have shown that regressors that are too powerful will overfit and substantial regularisation is required. To put this reasoning to a test, I designed a neural network with one LSTM layer (see section 2.4.3) followed by two densely connected layers and a 24 neuron output layer. To counteract the anticipated overfitting, two regularisation measures were taken: first, all layers but the output layer have an L2 penality applied to them and second, the LSTM layer features a dropout of 0.4. Detailed information on the design of the nework can be found in appendix B.1.

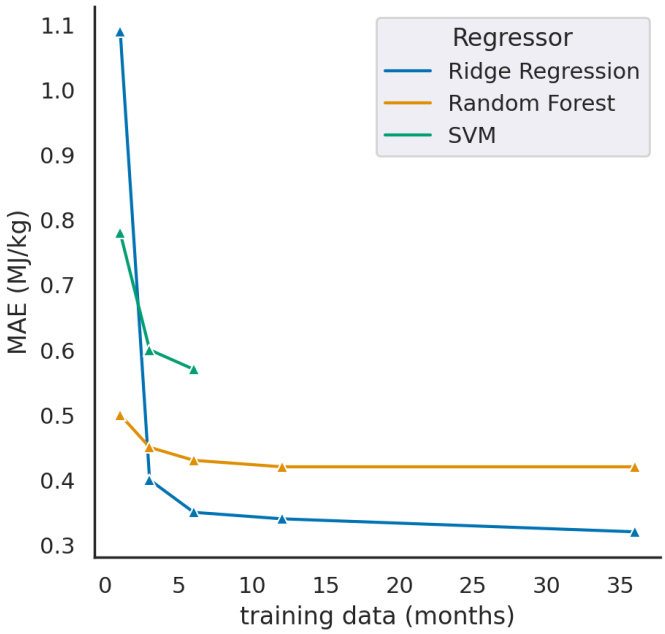


Figure 3.12: Ridge regression needs enough training data to perform well, but has the best accuracy for larger training sets. SVMs are generally worse than both RFs and ridge regression across all possible regularisation factors C.

The performance of this network was moderate, with a MAE of 0.51 MJ/kg. Worse still, not even the first few prediction points were as low as they have been in all of the other methods (see figure 3.13). Even if further improvements of the network architecture might improve the average MAE, ANNs do not seem to be the optimal regressors for the task at hand.

The neural network was implemented using keras [33]/tensorflow [34].

3.5 Including Image Data for the LHV Short-Term Predictions

In the previous sections, I have shown that there is information in calendar, temperature and plant data to make short-term predictions of the LHV of the waste that is about to be burnt. Those predictions are very good for the very near future, but get worse quite fast.

Conceptually this is easy to understand: the LHV of the near past holds information on the waste that is currently in the furnace and the other features hold information about the average waste that is about to enter. However, there is no information on the exact waste that is going to enter the furnace. This shortcoming can be solved by adding images of the waste (see figure 3.14) that is applied to the input chute of the furnace to the features of the model. As the waste takes up to 30 minutes from being applied to the chute until entering the furnace and another

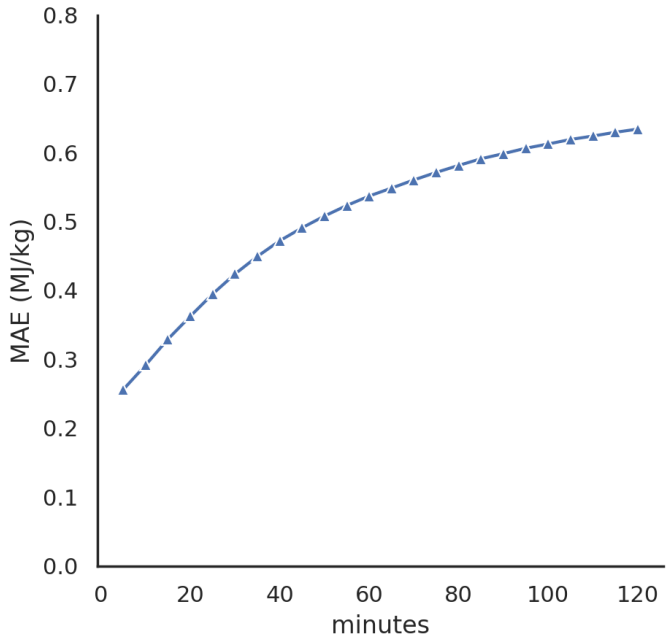


Figure 3.13: MAE of the RNN from section 3.4.2: it lacks the very low errors at near future time points that were obtained using all other regressors. The average MAE is 0.51 MJ/kg.

60 minutes until being fully incinerated, this should allow the model to make better prediction for the more distant future—even more so, as the actual waste that is applied to the chute gives a good indication on the bunker composition and therefore a good estimation on what waste might enter the chute in the future.

I will therefore add one image (320 rows, 480 columns, 3 channels, 8 bit) to each of the past eighteen time points of the model described in section 3.4 (see figure 3.15). This increases the dimensionality of the feature set from 180 to over eight million and the size of each sample—when stored at 32 bit—from about 1.4 kb to over 30 Mb. With a sampling rate of $\frac{1}{5min}$ this amounts to about 9 Gb of training data per day, which means that we cannot fit it into memory for all but the very smallest training sets.

3.5.1 LHV Prediction using Classical Regression Methods

A straight forward way to handle the large training set size is to use RFs. From an assembled basis data table, which contains all numerical features (at 64 bit precision) and the image (at 8 bit precision) with time steps of one minute, samples are created in batches of 500 *on the fly*. These contain all the data with 32 bit precision, as this is needed for the training algorithm, and have a size of about 16 Gb. More details on the creation of the training batches can be found in appendix A.

As they use the same five minute spacing between time steps as in section 3.4, the numerical values in the basis table have to be already preprocessed by a low-pass

Figure 3.14: Some arbitrary samples of what the image data used for LHV prediction looks like.

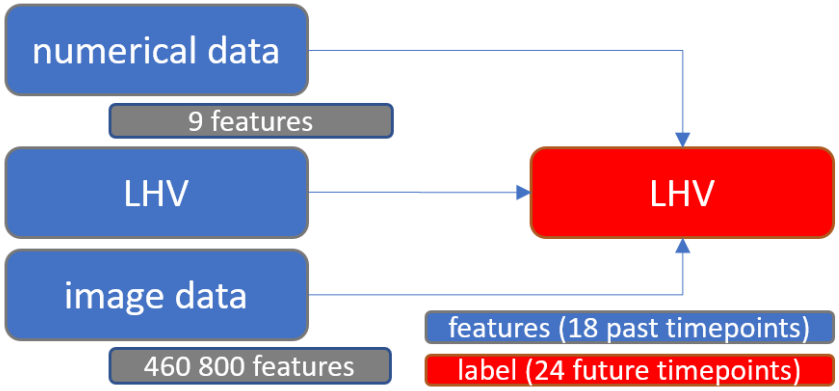


Figure 3.15: Design of the training set for the short-term LHV prediction, including the use of images.

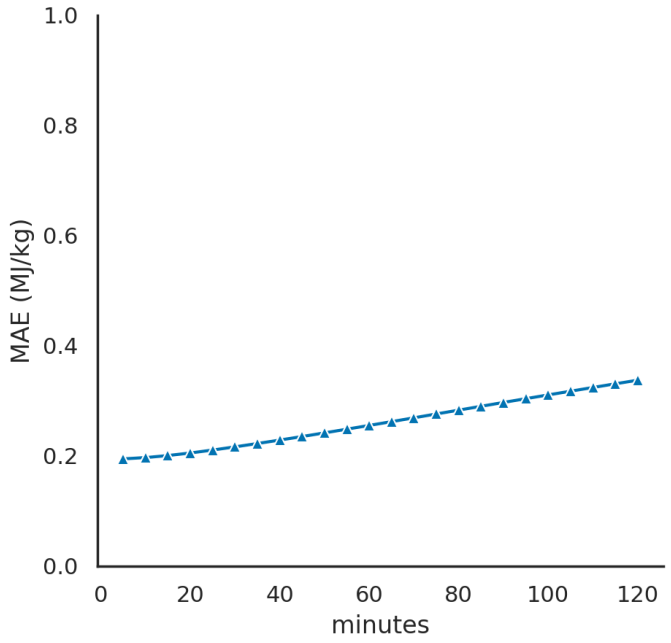


Figure 3.16: Constant increase of the MAE over time for the validation set. The Random Forest contains 240 decision trees, twenty of which are created for each batch. The trees are grown on half the training set until the leaves contain no more than 0.1% of the training samples, to reduce overfitting. The average MAE is 0.26 MJ/kg.

filter, so that no aliasing occurs from the downsampling (see section 2.6). For each of those batches, an ensemble of twenty limited decision trees (as used in section 3.4.1) is trained. Afterwards, the next 500 samples are loaded into memory and another twenty decision trees are *added* to the ensemble. Training is done using the limited decision trees described in section 3.4.1 in their respective scikit-learn implementation [21].

Evaluation of the accuracy was done by 4-fold backtesting using a training set size of 12 batches (about three weeks) and yielded an average validation MAE of 0.26 MJ/kg, using one batch as the validation set, which is separated by three batches from the training set. Runtime for training is now significant, with about 150 minutes per split of backtesting. The evolution of the MAE looks different than the one seen in section 3.4: there is no steep rise in the beginning, but rather a steady increase (see figure 3.16). This means, that the numerical data is not dominating the prediction—however, it has still a big influence, as the image data should be able to make constantly good predictions for at least half an hour.

There are two big downsides of using RFs. First of all, it cannot make use of the structure of the input—i.e., the fact that the added features are images—and second, the outlined training approach does not allow the training algorithm to see all possible combinations of the samples. At least for the former of those problems, I will show a solution in the upcoming section.

In principle, it is possible to use the approach of building ensembles with other predictors than decision trees: this general approach is called *bagging* [39], however, no other predictor was found to perform nearly as well as those.

Linear models, like ridge regression performed significantly worse, especially for the training error, as they were not expressive enough to fit the data. More expressive models, like kernalised SVMs (see section 2.2) need a lot of additional memory, and so it would be necessary to reduce the batch size significantly. Those models also significantly underperform the RFs.

Yet another approach is to train SVMs using SGD. However, there is a similar problem as in the bagging approach: linear models (without the use of a kernel) perform poorly, while kernalised SVMs suffer from the small batch size. This is because kernels for models trained by SGD need to be approximated by the Nystroem method (as was done for the outlier detection in 3.2.1), but its performance is limited for small training set sizes.

However, there is a model that can be trained by SGD and that is also able to model non-linearities: artificial neural networks, which will be covered in the upcoming section.

3.5.2 LHV Prediction using Neural Networks with Convolutional and Recurrent Layers

In section 3.4.2, I described the use of an ANN that uses an LSTM layer to make predictions on the future evolution of the LHV. However, this network did not outperform the classical methods.

Now there is a different situation when incorporating images as features. They are highly structured and can be processed efficiently by CNNs (see section 2.4.2). For each training sample, each of the images for the eighteen time points is passed through three consecutive convolutional layers, with pooling in between them. The final *feature map* is then flattened and—together with the numerical data—fed to the LSTM layer (see figure 3.17. A more detailed explanation of the architecture of that neural network can be found in appendix B.2). Additional regularisation could be applied to the network by adding dropout layers between the convolutional layers or adding an L2 penalty to the LSTM and/or dense layers (like it was done in section 3.5.2), though this was not found to be necessary. As outlined in section 2.4.2, the CNN in itself acts as a strong regularisation for image data.

Implementation and training of the outlined model has been done using keras [33]/tensorflow [34]. Batches are loaded to memory on the fly as outlined in section 3.5.1 and the neural network is trained on them, using a batch size of eight for the Adam optimiser [40], which performs a variant of SGD. The large feature maps and

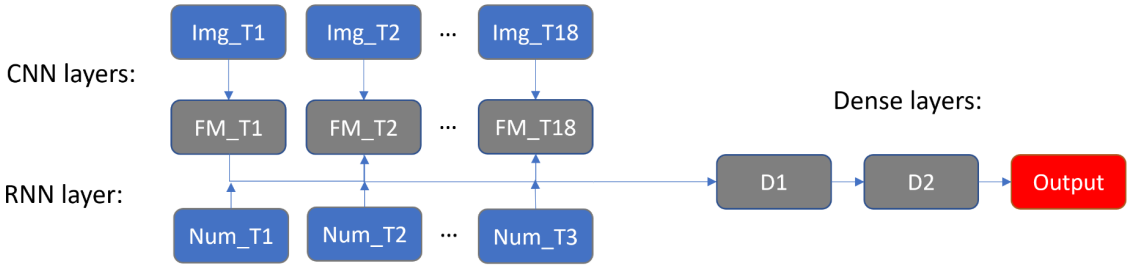


Figure 3.17: The basic architecture of the deep neural network for LHV prediction using image data: the eighteen input images are processed by three consecutive convolutional layers with pooling in between them, yielding the eighteen *feature maps* FM_T1 to FM_T18. They are then concatenated with the numerical features and processed by an RNN layer (specifically an LSTM layer, see section 2.4.3 and especially figure 2.6). Finally, there are two hidden dense layers (D1 and D2) and a dense output layer.

samples make larger batch sizes for the optimiser impossible to train on a machine with 8 Gb of GPU memory.

As neural networks usually benefit greatly from increased training data size, after processing all batches (i.e., for the next pass over the dataset) batches are created with an offset of one minute. With an interval of five minutes between the time points, this allows for an increase of the training set by a factor of five (details on the creation of those offset batches can be found in appendix A, listings 2, 3 and 4).

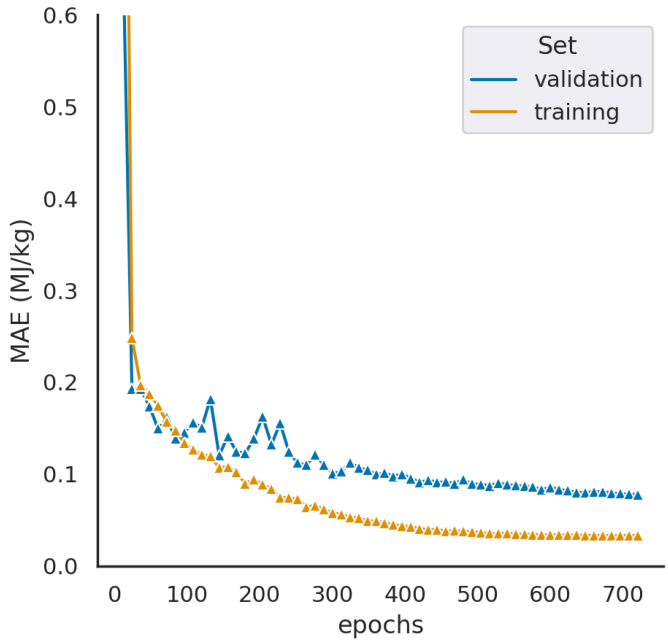


Figure 3.18: The evolution of the training and validation loss over 720 epochs. In the beginning, there are significant spikes in the validation loss, however, they become smaller and smaller as epochs pass. In the end, the validation loss settles in at a relatively constant loss rate, signifying that a (local) optimum has been approached. For this behaviour, a decaying learning rate is necessary.

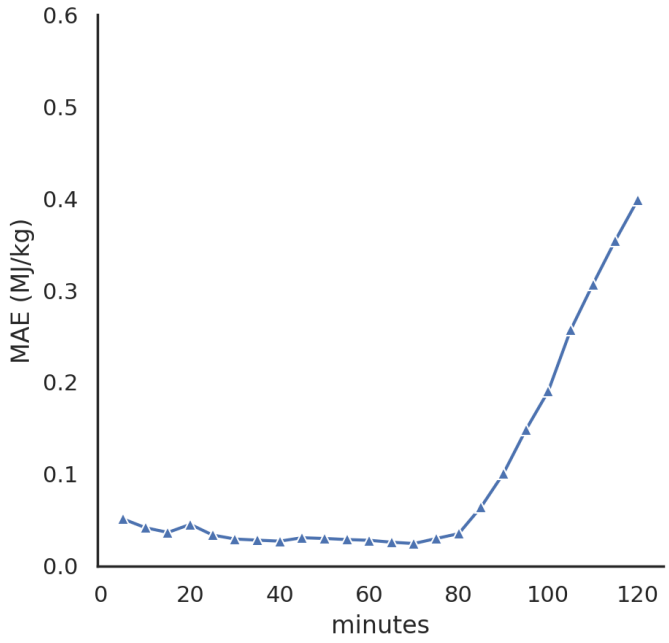


Figure 3.19: MAE of the neural network with convolutional and recurrent layers trained on the dataset containing image data. Up until 80 minutes, validation error stays at a constant low level. This is in sharp contrast to the predictions without images (see figure 3.8): as the images used for training the model fully describe the content of the furnace up until that point, the prediction does not get worse for that period of time. Only thereafter, the accuracy decreases in a steady way. The average MAE is 0.10 MJ/kg.

One important parameter for training neural networks is the learning rate of the optimiser. Setting it to a value that is too high will often lead to divergence, while setting it too low will trap the optimisation in some high-loss state. I used the Adam optimiser [40] with an initial learning rate of $3 \cdot 10^{-3}$ and a decay of 1% after each epoch, starting after passing over each training batch once and stopping at $3 \cdot 10^{-5}$. This is necessary, as otherwise there is the risk of jumping out of an optimum when training on a very unusual batch (see figure 3.18. More details on the training can be found in appendix C: the effects of the decaying learning rate are shown in figure C.5 and the effect of the number of epochs the current training batch is trained for is shown in figure C.6.)

Evaluation of the accuracy has been done by 4-fold backtesting using twelve batches with all five possible offsets for twelve total passes over the whole dataset, yielding a total of 720 epochs. Runtime was about ten hours per split of backtesting. The model has an average MAE of 0.10 MJ/kg, but—more importantly—the MAE of time points up to about 80 minutes is at a constant low level (see figure 3.19). This behaviour can be expected, as this is the time span for which the waste on the images completely determines what kind of waste is inside the furnace. After that, the accuracy decreases in a steady way, as from there on, the images only partly determine the content of the furnace. Examples of this can be seen in figure 3.20.

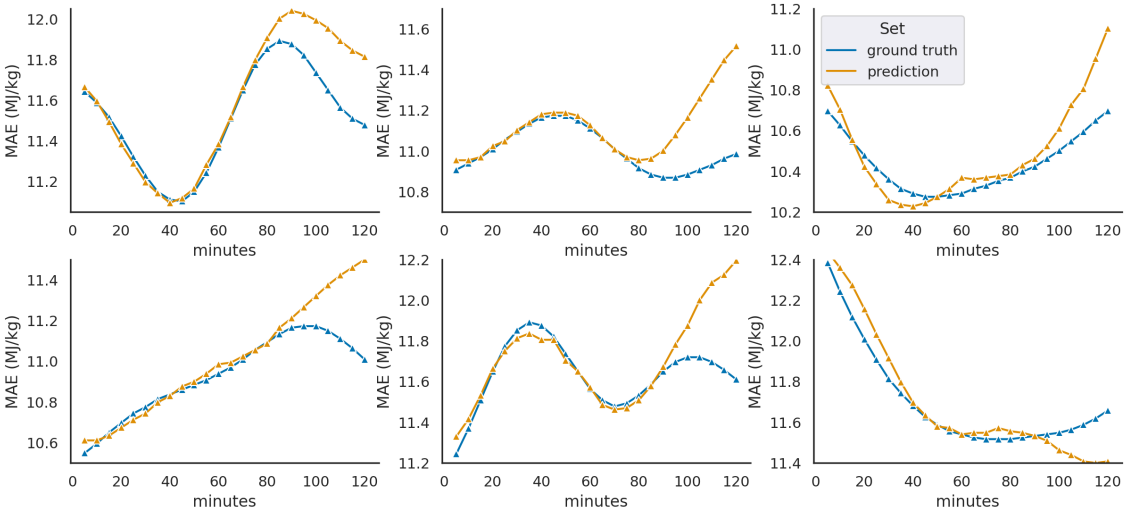


Figure 3.20: Arbitrarily chosen representative predictions of the ANN model with an average MAE between 0.09 and 0.11 MJ/kg. When comparing them with the predictions of the model without images (see figure 3.9), there is an immense difference: the model does not simply continue the trend, but rather accurately predicts the ground truth LHV up until the point where the training images fully describe the content of the furnace. For about 80 minutes the obtained predictions are extremely useful for the plant operator, while they still remain useable for the whole two hours.

4. Limitations and Future Work

Before discussing the results of the previous chapter, it is necessary to consider their limitations in-depth and point to possible future paths to mitigate them.

The main fact that has to be kept in mind for all the results obtained in this thesis—from the simple models for the daily averages to the sophisticated short-term predictions using image data—is that the data generating distribution $\mathcal{D}$ is not stationary, i.e., it changes over time. Consider the following possibility: a new ventilation system is introduced into the waste bunker so that the waste dries faster. The correlation between air humidity and the LHV (see figure C.2 in the appendix) might now decrease or even vanish and—in consequence—the model's performance will decrease. While this reason for the change in $\mathcal{D}$ might get noticed, others will not: if a municipality stops collecting plastic bottles separately, they will end up in the residual waste bags. Those will still look exactly the same on the images, yet have a higher LHV.

Such changes in $\mathcal{D}$ might, however, be found when analysing the MAE of the model over time. If there is a steep increase, the underlying $\mathcal{D}$ might have substantially changed and the model needs to be retrained.

Even so, what is probably going to occur more often is not a punctual *shift* in $\mathcal{D}$, but a moderate, ongoing *drift*. Therefore, one important path for future research is to find methods that cater for that ever-changing data generating distribution. One such possibility might be to use window-based approaches—within each window, $\mathcal{D}$ can be considered approximately stationary. This has already been done for time series forecasting with ANNs [41] and might be a good starting point.

While the non-stationary data generating distribution is fundamental to the problem tackled by this thesis, the choice of the feature time series is not. As outlined in section 3.4, eighteen evenly spaced past time points with an interval of five minutes were used for each feature based on *expert knowledge*. While increasing the number of time points will put more strain on the memory constraints of the model, future work is needed to optimise the spacing of the time points: they might be packed looser or tighter, therefore spanning a larger or shorter amount of time. Moreover, it is possible that they should not be evenly spaced and it is better to skew them towards a particular point in time.

Speaking of the strain that more time points would put on the model, handling the large dimensionality of the feature space is another way for improvement, and it can be achieved from two sides: first, reducing the dimensionality—especially of the images—would allow for the use of larger past time series or larger training batches. There are many possibilities for that: reducing the size of the images, reducing the

colour channels, using compression methods like autoencoders or cosine-transform based ones or even using generic dimensionality reduction methods like principle component analysis (PCA). Second, the capabilities of the computational environment (see section 3.1) can be easily improved: CPU memory can be quadrupled to 128 Gb and GPU memory tripled to 24 Gb, without even going beyond what is currently available at the consumer market.

This would allow the ANN to be substantially improved. The convolutional part of the network from section 3.5.2 (outlined in detail in appendix B.2) has a deceasing number of filters from layer one to three only due to memory constraints. Lifting that restriction—or even adding more convolutional layers—could certainly enable the network to capture more information from the images, while adding more units to the LSTM layer could—together with more past time points—improve the ability of the network to generalise for changing waste application sequences. Finally, making the network even deeper by adding more dense layers could eventually bring it to the point where it overfits and additional regularisation is needed. Such a model often performs better than one that does not need regularisation to start with.

Moreover, the model from section 3.5.2 is trained by only using minibatch sizes of eight for SGD. That is on the low side and might lead to the optimiser getting stuck in a local optimum far from the global one. By relieving the memory constraints encountered in this work and consequently using minibatch sizes of 32 or 64, the model might well be significantly improved.

One last point that hinders the performance of all models suffering from the training set not fitting into memory is, that the training algorithm—be it RFs or ANNs—does not see all combinations of samples during optimisation. Constructing the training batches that are loaded into memory in a way that they consist of samples from different (and varying) parts of the whole training set would alleviate that problem. However, this is much easier if memory is not as tight, so that larger (or more) dataframes can be loaded into memory for constructing the training batches (see appendix A for more details).

5. Discussion

In this thesis, I have shown that it is possible to make accurate short-term predictions of the LHV of waste that is about to be burnt in an MSWI furnace using only data that is already available and recorded continuously in the plant, as it is used for various operational means. No further sensors have to be installed, no new analyses have to be performed and the operational process does not need to be altered—facts that are very convenient for the plant operator.

All that data can of course also be accessed by the operating personnel, where they can use it to make estimates on the evolution of the LHV. However, that data is not easy to interpret. While it might seem straightforward to get information from the numerical values (calendar data, weather data, the bunker volumes and the current LHV), a lot of information lies in the past evolution of those values, which is hard to interpret by even the most experienced operating personnel. A similar argument can be made for the image data. The crane driver, who applies the waste to the input chute, will constantly monitor the camera from which the image data is drawn and will try to produce a smooth waste input that leads—among others—to smooth LHV values. However, deducting the waste composition from the camera images is often hard, especially as those judgments have to be made in a short amount of time.

In the course of this work, I have described two different models, which can use that data and make short-term predictions. For the very short future, ridge regression using only the non-image data can make very good predictions, with MAEs below 0.1 MJ/kg. This model (described in section 3.4.1) puts very much emphasis on continuing the LHV-trend. This can be seen when analysing the coefficients of the ridge regression or the splitting attributes of the decision trees in the RFs. On the downside, that model's prediction accuracy decreases drastically at time points more than 30 minutes into the future. Still, it is easy and fast to train, even for very large training sets. If prediction of the LHV is only required for a short future time span, this model is very useful. Moreover, the underlying data generating distribution of this model seems to be quite stable and the model generalises well, therefore it can be used without the need for much retraining.

The other model (described in section 3.5.2) uses deep neural networks in combination with the image data to make predictions of more distant future time points. The MAE of that model does not increase in the same way the first model does. In contrast, its MAE stays constantly low (under 0.05 MJ/kg) for about 80 minutes and increases steadily from there. This model does not only continue the trend, but also uses the information in the images that is directly connected to the waste's

LHV. As the waste on the images takes anywhere from a couple of minutes to half an hour to enter the furnace and another hour to be fully incinerated, it is easy to see why this is the case.

However, there are currently severe limitations for this model, which I have described in chapter 4. Moreover, I do not claim that my design of the neural network is optimal by any means. I have given some outlines on how to improve them in chapter 4, too, but future research might come up with different network designs altogether. In the end, the results of this work should be seen as a starting point rather then a destination.

Despite all of those caveats, the main point I showed in this thesis is, that by using only data that is already available and recorded at a typical MSWI plant, it is possible to predict the LHV of the waste that is about to be burnt. This finding might even be more generally applicable: it seems likely that other operational parameters dependent on the composition of the input waste can be predicted using similar models. Those parameters might include emission values such as sulphur dioxide, dioxins and many other environmentally relevant pollutants. Still, further research is needed to verify this assumption.

Appendices

A. Image Dataset Handling

As described in section 3.5, because of the large size of the image dataset, the generation of the training set batches is done in a two-step process.

In the first step, a pandas [36] dataframe containing the image data in one column is created by attaching the images to the dataframe with the numerical data. The python code can be seen in listing 1. The images are stored as 640×480 JPEGs, each day in a folder named DD_MM. The image naming convention is *VMSExport_YYYY-MM-DD hh.mm.ss.lll.jpg*, where *lll* is the number of milliseconds. This is the native export format of the MVA Dürnrohr's camera export system.

The raw images (see figure A.1) are cut to only contain the input chute and no surroundings or image caption, resulting in an image dimension of 480×320. This has been chosen as small as possible to minimise the training sample size. The numerical data of the dataframe is stored as 64-bit float, the image as a one-dimensional array with 153600 8-bit ints (see figure A.2).

In the second step, during the training of the models of section 3.5, the training set batches are created *on the fly*. Each sample contains a time series of eighteen past time points of all features with a spacing of five minutes in between them and a time series of 24 future time points for the label, using the same spacing. One training batch contains 500 of those samples (about one day and eighteen hours), with the origin of the past and future time series shifted five minutes between each sample. For the deep learning applications, those batches are also created with all five possible one minute offsets to increase the training set size.

To be more economical with the available memory, the pandas dataframe is split into smaller ones, each containing the raw data for four sample batches (and all possible offsets), i.e. 10 000 rows plus the necessary rows before and after them to account for the past and future time series (in total, 10 215 rows). The naming convention for them is *img_dataset_nnn.ftr*, using the feather file format.

The creation of the numpy [35] arrays with the training batches is done by the functions in listings 2, 3 and 4, creating one array for the labels, one array for the numerical features and one array for the image features.

```
import pandas as pd
import numpy as np
from glob import glob
from PIL import Image

fails = 0
df_img = pd.DataFrame()
for i, row in df.iterrows():
   folder = str (row.date.day).rjust (2, '0') + '_' \
   + str (row.date.month).rjust (2, '0')
   hour = row.date.hour
   minute = row.date.minute
   found = False
   while not found: # found an image for the row?
      filename = 'VMSExport_' + str (row.date.year) + '-' + \
      str (row.date.month).rjust (2, '0') + '-' \
      + str (row.date.day).rjust (2, '0') + ' ' + str (hour).rjust (2, '0') \
      + '.' + str (minute).rjust (2, '0') + '.'
      filestring = './images/' + folder + '/' + filename + '*' + '.jpg'
      file = glob (filestring)
      if file: # check if we get at least one file
         found = True
         # load image into array
         img = Image.open(file[0])
         img_array = np.asarray (img)
         # image cutting (to 320x480x3)
         img_array = img_array [90:, 75:, :]
         img_array = img_array [:-70, :-85, :]
         # flatten image
         img_array = img_array.flatten()
         img.close ()
         row['image'] = img_array
         df_img = pd.concat ([df_img, row.to_frame().T], axis = 0,
                             ignore_index=True)
      else: # no file found
         if minute < 59:
            minute += 1
         else:
            hour += 1
         if hour > 23 or minute > 59: # definitive fail
            fails += 1
            print (f'Fail at row {i}.')
            found = True
print (f'Fails: {fails}.')
```

Listing 1: Creation of the pandas dataframe containing the image data, by appending an already existing dataframe containing the numerical features with a column for the image (as a one-dimensional array).

Figure A.1: Raw image from the camera export system of the MVA Dürnrohr. This picture's filename is *VMSExport_2022-06-24 06.00.08.446.jpg*. Compared fo figure 3.14, it can be seen that the parts that are not relevant for the prediction are cut away: the in-image caption and the image parts outside the input chute.

	date	hour	weekday	week	temperature	moisture	pressure	wind	main_bunker	pre_bunker	label	image
0	2022-05-27 00:00:00	0	4	21	16.125922	83.745874	1017.46889	0.319845	18514.279807	62.902841	11.929245	[87, 87, 85, 86, 86, 84, 85, 85, 83, 84, 84, 8...
1	2022-05-27 00:01:00	0	4	21	16.572734	79.759721	1017.057907	0.11883	16872.920899	34.434609	11.973659	[148, 149, 153, 149, 150, 154, 149, 150, 154, ...
2	2022-05-27 00:02:00	0	4	21	16.921641	76.555643	1016.721641	0.014044	15544.76438	10.778929	12.017925	[150, 151, 155, 152, 153, 157, 153, 154, 158, ...
3	2022-05-27 00:03:00	0	4	21	17.138305	74.429354	1016.493999	0.083979	14659.180619	5.643933	12.061893	[144, 145, 147, 144, 145, 147, 145, 146, 148, ...
4	2022-05-27 00:04:00	0	4	21	17.217571	73.433097	1016.383987	0.10099	14248.665937	13.910841	12.105417	[148, 149, 153, 148, 149, 153, 150, 151, 155, ...
...	...	...	...	...	...	...	...	...	...	...	...	...
51835	2022-07-01 23:55:00	23	4	26	14.4062	98.750516	1018.879344	2.080937	24528.235486	165.163311	11.71027	[131, 132, 134, 131, 132, 134, 132, 133, 135, ...
51836	2022-07-01 23:56:00	23	4	26	14.531728	97.64443	1018.807572	1.711115	24119.248236	157.746926	11.753149	[131, 133, 132, 132, 134, 133, 134, 136, 135, ...
51837	2022-07-01 23:57:00	23	4	26	14.791268	95.392245	1018.61291	1.311521	23235.458175	142.692008	11.796635	[131, 133, 132, 133, 135, 134, 133, 135, 134, ...
51838	2022-07-01 23:58:00	23	4	26	15.172997	92.082282	1018.300809	0.92769	21908.859705	120.399862	11.840581	[132, 134, 131, 133, 135, 132, 134, 136, 133, ...
51839	2022-07-01 23:59:00	23	4	26	15.637561	88.036401	1017.902589	0.591361	20268.400252	92.773479	11.884835	[132, 134, 133, 132, 134, 133, 132, 134, 133, ...

Figure A.2: Pandas dataframe with the raw data of the whole training set containing the image data as a one-dimensional array.

```
import pandas as pd
import numpy as np

# from a dataframe, create samples of images from start to end with
# a given offset and a given interval. Images have a given dimension
# (rows, cols, channels) in the row of df given by img_feature. Each
# sample contains back_length past images.
def create_img_array (df, end, img_dim, start=0, interval=5, offset=0,
                      back_length=18, img_feature='image', bit='32'):
   if start+offset < back_length:
      raise ValueError (f'Start at {start} is not possible with \
                          back_length == {back_length}.')
   if bit == '16' or bit == 16:
      type = np.float16
   elif bit == '32' or bit == 32:
      type = np.float32
   elif bit == '64' or bit == 64:
      type = np.float64
   else:
      raise ValueError ('bit must be either 16, 32 or 64.')
   # for the first dimension, we need to round up
   Xi = np.empty (((end-start)//interval+((end-start)%interval>0),
                  back_length, img_dim[0], img_dim[1], img_dim[2]),
                  dtype=type)
   for i, sample in enumerate (range (start+offset, end+offset,
                                      interval)):
      for back in range (back_length):
         # the features are ordered chronologically until the
         # current one (from sample-back_length to sample).
         Xi[i, back] = (df[img_feature][sample-back_length+
                       (back*interval)+1]/255).reshape (
                       (img_dim[0], img_dim[1], img_dim[2]))
   return Xi
```

Listing 2: Function for the creation of the image feature batches.

```python
# from a dataframe create samples of numerical data from start to
# end with a given offset and a given interval. Features to be
# taken are given in the feature-list. Each sample contains
# back_length time points for each feature.
def create_num_array (df, end, features, start=0, interval=5,
                      offset=0, back_length=18, bit='32'):
   if start+offset < back_length:
      raise ValueError (f'Start at {start} is not possible with \
                        back_length == {back_length}.')
   if bit == '16' or bit == 16:
      type = np.float16
   elif bit == '32' or bit == 32:
      type = np.float32
   elif bit == '64' or bit == 64:
      type = np.float64
   else:
      raise ValueError ('bit must be either 16, 32 or 64.')
   # for the first dimension, we need to round up
   Xn = np.empty (((end-start)//interval+((end-start)%interval>0),
                  back_length, len (features)), dtype=type)
   for i, sample in enumerate (range (start+offset, end+offset,
                                      interval)):
      for back in range (back_length):
         for j, feature in enumerate (features):
         # the features are ordered chronologically until the
         # current one (from sample-back_length to sample).
         Xn[i, back, j] = df[feature][sample-back_length+
                           (back*interval)+1]
   return Xn
```

Listing 3: Function for the creation of the numerical feature batches.

```python
# from a dataframe create the label array from the column given by
# label from start to end with a given offset and a given interval.
# Each label contains forward_length future time points.
def create_label_array (df, end, start=0, interval=5, offset=0,
                        forward_length=24, label='label', bit='32'):
   if end+offset > df.shape[0]-forward_length:
      raise ValueError (f'End at {end} is not possible with \
                        forward_length == {forward_length}.')
   if bit == '16' or bit == 16:
      type = np.float16
   elif bit == '32' or bit == 32:
      type = np.float32
   elif bit == '64' or bit == 64:
      type = np.float64
   else:
      raise ValueError ('bit must be either 16, 32 or 64.')
   # for the first dimension, we need to round up
   y = np.empty (((end-start)//interval+((end-start)%interval>0),
                  forward_length), dtype=type)
   for i, sample in enumerate (range (start+offset, end+offset, interval)):
      for forward in range (forward_length):
         y[i, forward] = df[label][sample+(forward*interval)+1]
   return y
```

Listing 4: Function for the creation of the label batches.

B. ANN Design

B.1 Design of the ANN without Image Features

The detailed architecture of the ANN from section 3.4.2 looks as follows: the input is passed through an LSTM layer with a *tanh* activation function and a dropout of 0.4. The flattened output is then fed through two densely connected layers with 512 neurons and a ReLU activation function. The output layer resembles those layers, but only consists of 24 neurons for the 24 prediction values. All layers but the output layer have a L2 regularisation penalty of 0.01.

The network is defined by the code in listing 5 using keras [33]/tensorflow [34] and is visualised in figure B.1.

```
from tensorflow.keras import layers, Input, Model, regularizers

units = 128
lags = 18
dim = 10

input_data = Input (shape=(lags, dim))
x = layers.LSTM (units, return_sequences=True, dropout=0.4,
                 recurrent_dropout=0.4, kernel_regularizer=
                 regularizers.L2 (1e-2))(input_data)
x = layers.Flatten()(x)
x = layers.Dense (512, activation='relu', kernel_regularizer=
                  regularizers.L2 (1e-2))(x)
x = layers.Dense (512, activation='relu', kernel_regularizer=
                  regularizers.L2 (1e-2))(x)
output = layers.Dense (24, activation='relu')(x)

model = Model (input_data, output)
model.compile (optimizer='adam', loss='mean_squared_error')
```

Listing 5: Creation of the ANN for LHV prediction without the use of image features (see section 3.4.2).

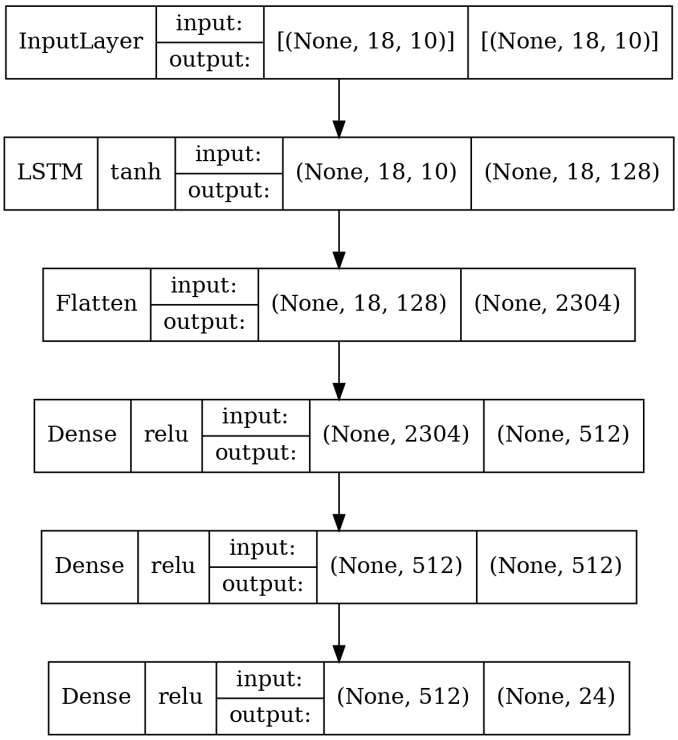


Figure B.1: Visualisation of the architecture of the neural network from section 3.4.2 that does not use images as features.

B.2 Design of the ANN with Image Features

The detailed architecture of the ANN from section 3.5.2 looks as follows: there are three consecutive convolutional layers using a 2×2 kernel and a leaky ReLU activation with an α of 0.15. In between, there are 3×3 max pooling layers (only pooling within an image and not mixing the different time points). The final feature map is then concatenated with the numerical features and processed by an LSTM layer with the same leaky ReLU activation and a sigmoid recurrent activation. Finally, there are two dense layers and a dense output layer, all with a leaky ReLU activation. No additional regularisation or dropout layers are applied.

The network is defined by the code in listing 6 using keras [33]/tensorflow [34] and is visualised in figure B.2.

```
from tensorflow.keras import Model, Input
from tensorflow.keras import layers, regularizers, optimizers

cnn_kernel = (2, 2)
cnn_filter = 8
cnn_stride = 1
pool_size = (1, 3, 3)
lstm_units = 128
input_img = Input (shape=(lags, rows, cols, channels))
input_num = Input (shape=(lags, numdim))

# CNN part
x = layers.Conv2D (cnn_filter, cnn_kernel,
                   activation=layers.LeakyReLU (0.15),
                   strides=cnn_stride)(input_img)
x = layers.MaxPooling3D (pool_size=pool_size)(x)
x = layers.Conv2D (cnn_filter//2, cnn_kernel,
                   activation=layers.LeakyReLU (0.15),
                   strides=cnn_stride)(x)
x = layers.MaxPooling3D (pool_size=pool_size)(x)
x = layers.Conv2D (cnn_filter//4, cnn_kernel,
                   activation=layers.LeakyReLU (0.15),
                   strides=cnn_stride)(x)
x = layers.MaxPooling3D (pool_size=pool_size)(x)

# LSTM part
x = layers.Reshape ((lags, -1))(x)
x = layers.Concatenate()([x, input_num])
x = layers.LSTM (units=lstm_units,
                 activation=layers.LeakyReLU (0.15),
                 recurrent_activation='sigmoid')(x)

# Dense part
x = layers.Flatten()(x)
x = layers.Dense(128, activation=layers.LeakyReLU (0.15))(x)
x = layers.Dense(128, activation=layers.LeakyReLU (0.15))(x)
output = layers.Dense(24, activation=layers.LeakyReLU (0.15))(x)

# Model creation
model = Model ([input_img, input_num], output)
model.compile (optimizer=optimizers.Adam(learning_rate=1e-3),
               loss='mean_squared_error')
```

Listing 6: Creation of the ANN for LHV prediction with the use of image features (see section 3.5.2).

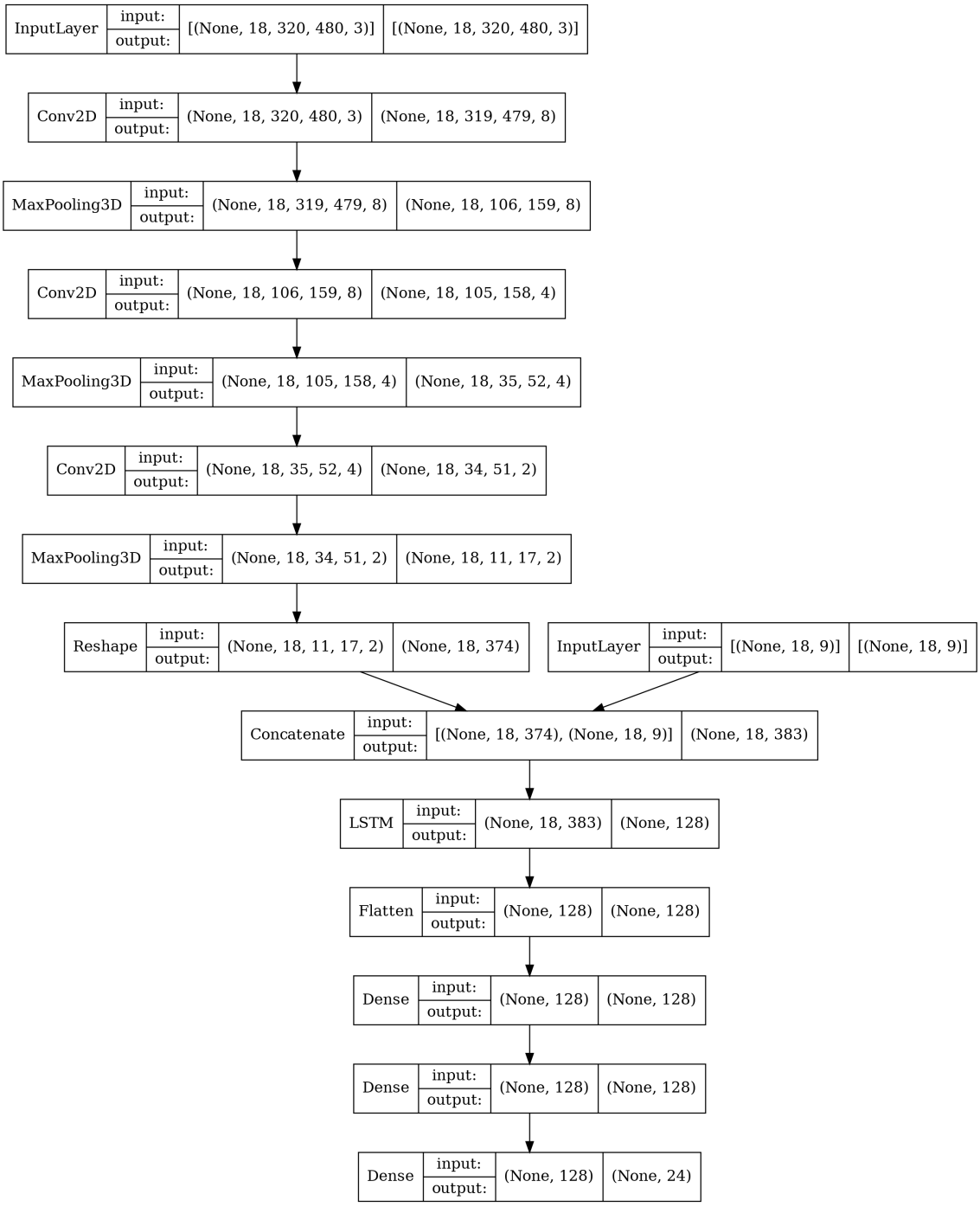


Figure B.2: Visualisation of the architecture of the neural network from section 3.5.2 that uses images as features.

C. Additional Results

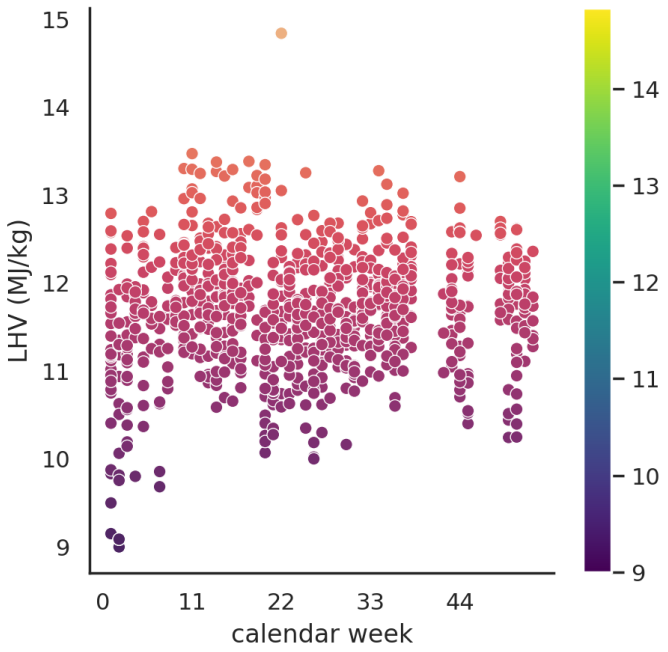


Figure C.1: A correlation of the LHV over the course of the year can be seen, with lower LHVs in the winter and summer holiday seasons.

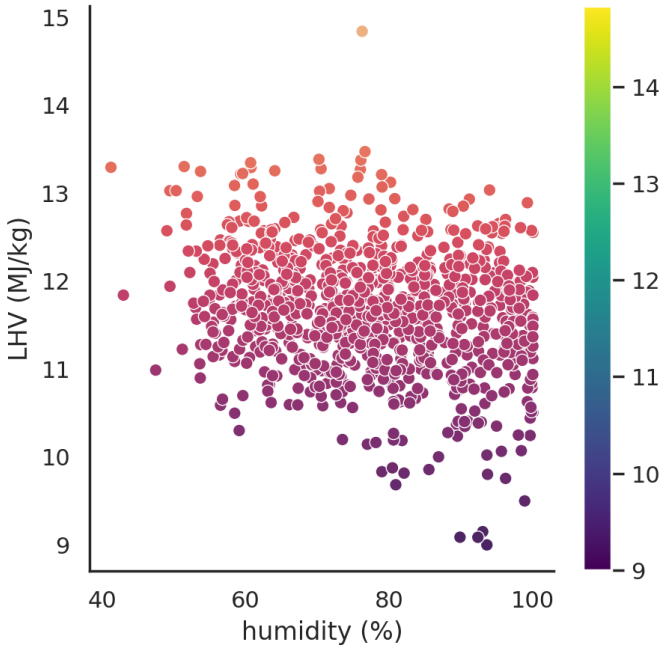


Figure C.2: A correlation of the LHV with the measured air humidity can be seen, with higher LHVs at lower relative humidity.

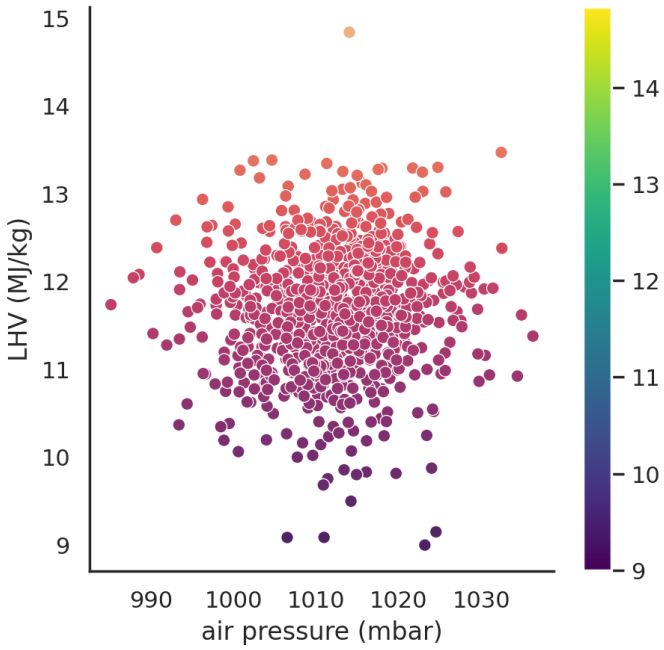


Figure C.3: The correlation between the LHV and the air pressure is relatively weak, however, there is a small positive correlation.

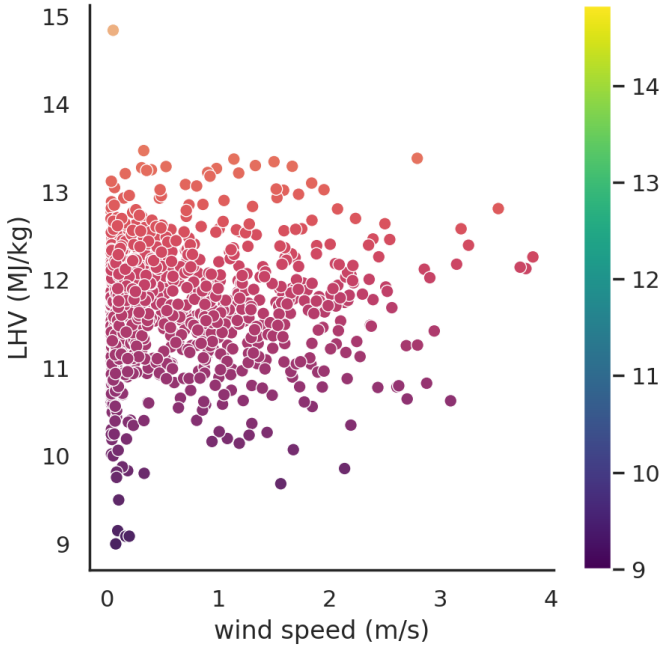


Figure C.4: The correlation between the LHV and the wind speed is relatively weak, however, there is a small positive correlation.

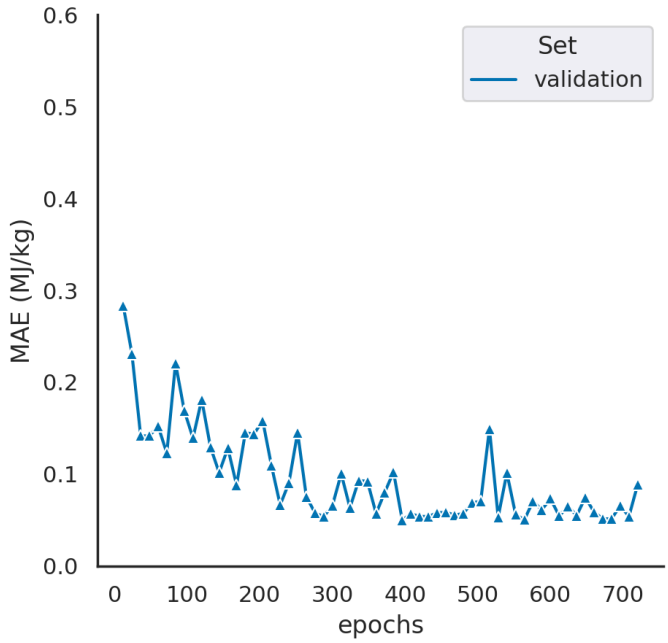


Figure C.5: Evolution of the validation loss of the neural network including image data as features from section 3.5.2 without a decaying learning rate over 720 training epochs. It can be seen that the validation loss often jumps out of an optimum, undoing the optimisation progress made thus far.

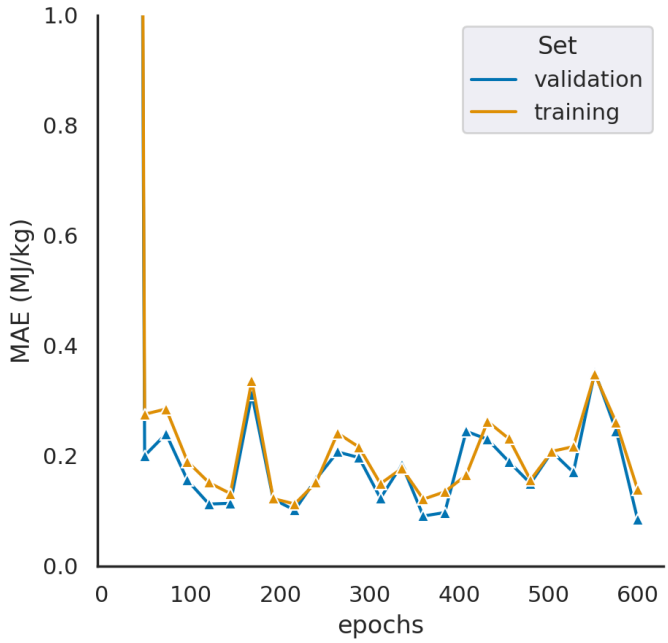


Figure C.6: Evolution of the training and validation loss of the neural network including image data as features from section 3.5.2 without a decaying learning rate and the training of the model for three epochs for each batch loaded to memory over 720 training epochs. While the training of three epochs for each batch loaded into memory may sound like an efficient use of time, it decreases the training performance significantly, never allowing the optimiser to find a stable optimum.

D. Zusammenfassung

Müllverbrennungsanlagen für Siedlungsabfälle sind ein integraler Bestandteil der Abfallwirtschaft von Industriestaaten. Sie sind einerseits in der Lage, organische Schadstoffe zu zerstören und können andererseits das benötigte Deponievolumen um etwa 90 Prozent reduzieren, indem sie anorganische Schadstoffe aufkonzentrieren. Aus diesem Grund stellen sie die Hauptbehandlungsmethode für jene Abfälle dar, die nicht sinnvoll recycelt werden können.

Im Gegensatz zu anderen Feuerungsanlagen müssen Müllverbrennungsanlagen mit verschiedensten Brennstoffen – den eingesetzten Abfällen – zurechtkommen, die den Verbrennungsprozess signifikant beeinflussen können. Dabei stellt die Tatsache, dass die Zusammensetzung des Abfalls vor dessen Verbrennung nicht exakt bestimmt werden kann, ein großes Problem dar: So sind bei der Sichtkontrolle von Abfällen nur größere Auffälligkeiten erkennbar und die Ergebnisse chemischer Analysen stehen üblicherweise erst ein oder mehrere Tage später zur Verfügung.

Daher sind in den letzten Jahren zahlreiche *Machine Learning*-Methoden entwickelt worden, die die Eigenschaften des Abfalls – allem voran dessen *Heizwert* – vorhersagen können. Diese zielen entweder auf die Vorhersage des Tagesmittelwertes oder des Heizwertes einer bestimmten Abfallfraktion ab, wohingegen es für den Anlagenbetreiber von großem Interesse ist, den Heizwert des Abfalls, der sich in der nahen Zukunft im Feuerraum befinden wird, vorherzusagen. Mit dieser Information wären operative Eingriffe möglich, um das Enstehen von Heizwertspitzen zu verhindern.

Zu diesem Zweck werde ich – ausgehend von bekannten Prädiktoren des Heizwertes wie Kalenderdaten (Wochentag, Kalenderwoche) und Wetterdaten (Temperatur, Niederschlag usw.) – die bekannten Vorhersagemodelle für Tagesmittelwerte zu Vorhersagemodellen für Zeitreihen in der nahen Zukunft erweitern, welche Methoden wie Ridge Regression, Support Vector Machines, oder Random Forests verwenden. Diese sind in der Lage, solide Vorhersagen für die nächsten 30 Minuten zu tätigen. Für Zeitpunkte, die weiter in der Zukunft liegen, werden sie allerdings sehr schnell unbrauchbar. Sie können insofern nur den Trend der vergangenen Heizwertkurve fortschreiben.

Ein offensichtlicher Mangel dieser Methode ist die Tatsache, dass keiner der Prädiktoren Informationen zur tatsächlichen Zusammensetzung des Abfalls enthält, der im Feuerraum verbrannt werden wird. Zur Lösung dieses Problems erweitere ich jene um Bilddaten, die den Abfall im Bereich der Aufgabeschurre zeigen. Diese entwickeln das Modell wesentlich weiter, da sie Informationen über den Abfall, der sich in den nächsten 90 Minuten im Feuerraum befinden wird, liefern. Wenngleich

die hohe Dimensionalität des dadurch entstehenden Merkmalraums (bestehend aus einer Zeitreihe mit 18 Zeitschritten, von denen jeder zusätzlich zu den numerischen Merkmalen ein 480×320 großes Bild in drei Farbkanälen enthält) andere Methoden – wie ein spezielles Design von künstlichen neuronalen Netzen – verlangt, erzielt das damit entwickelte Modell außerordentlich gute Genauigkeitswerte für bis zu 80 Minuten in die Zukunft.

Bibliography

[1] P. H. Brunner and H. Rechberger, "Waste to energy – key element for sustainable waste management," *Waste Management*, vol. 37, pp. 3–12, 2015, Special Thematic Issue: Waste-to-Energy Processes and Technologies, ISSN: 0956-053X. DOI: https://doi.org/10.1016/j.wasman.2014.02.003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0956053X14000543.

[2] K. Weber, P. Quicker, J. Hanewinkel, and S. Flamme, "Status of waste-to-energy in germany, part i – waste treatment facilities," *Waste Management & Research*, vol. 38, no. 1_suppl, pp. 23–44, 2020, PMID: 31928170. DOI: 10.1177/0734242X19894632. eprint: https://doi.org/10.1177/0734242X19894632. [Online]. Available: https://doi.org/10.1177/0734242X19894632.

[3] U. Ozveren, "An artificial intelligence approach to predict a lower heating value of municipal solid waste," *Energy Sources, Part A: Recovery, Utilization, and Environmental Effects*, vol. 38, no. 19, pp. 2906–2913, 2016. DOI: 10.1080/15567036.2015.1107864. eprint: https://doi.org/10.1080/15567036.2015.1107864. [Online]. Available: https://doi.org/10.1080/15567036.2015.1107864.

[4] M. Bagheri, R. Esfilar, M. S. Golchi, and C. A. Kennedy, "A comparative data mining approach for the prediction of energy recovery potential from various municipal solid waste," *Renewable & Sustainable Energy Reviews*, vol. 116, p. 109 423, 2019.

[5] O. O. Olatunji, S. A. Akinlabi, N. Madushele, P. A. Adedeji, and F. A. Ishola, "Multilayer perceptron artificial neural network for the prediction of heating value of municipal solid waste," *AIMS Energy*, 2019.

[6] H. You, Z. Ma, Y. Tang, *et al.*, "Comparison of ann (mlp), anfis, svm, and rf models for the online classification of heating value of burning municipal solid waste in circulating fluidized bed incinerators," *Waste Management*, vol. 68, pp. 186–197, 2017, ISSN: 0956-053X. DOI: https://doi.org/10.1016/j.wasman.2017.03.044. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0956053X17301903.

[7] C. Birgen, E. Magnanelli, P. Carlsson, Ø. Skreiberg, J. Mosby, and M. Becidan, "Machine learning based modelling for lower heating value prediction of municipal solid waste," *Fuel*, vol. 283, p. 118 906, Jan. 2021. DOI: 10.1016/j.fuel.2020.118906.

[8] S. Shalev-Shwartz, *Understanding machine learning : from theory to algorithms /*, eng. Cambridge : Cambridge University Press, 2014, ISBN: 1-107-29801-6.

[9] A. E. Hoerl and R. W. Kennard, "Ridge regression: Biased estimation for nonorthogonal problems," *Technometrics*, vol. 12, no. 1, pp. 55–67, 1970, ISSN: 00401706. [Online]. Available: http://www.jstor.org/stable/1267351 (visited on 06/07/2022).

[10] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, ser. COLT '92, Pittsburgh, Pennsylvania, USA: Association for Computing Machinery, 1992, pp. 144–152, ISBN: 089791497X. DOI: 10.1145/130385.130401. [Online]. Available: https://doi-org.uaccess.univie.ac.at/10.1145/130385.130401.

[11] T. Hastie, *The elements of statistical learning : data mining, inference, and prediction* (Springer series in statistics), eng, 2. ed.. New York, NY: Springer, 2009, ISBN: 9780387848570.

[12] C. Cortes, "Support-vector networks," eng, *Machine learning*, vol. 20, no. 3, p. 273, 1995, ISSN: 0885-6125.

[13] C.-C. Chang and C.-J. Lin, "Libsvm: A library for support vector machines," *ACM Trans. Intell. Syst. Technol.*, vol. 2, no. 3, May 2011, ISSN: 2157-6904. DOI: 10.1145/1961189.1961199. [Online]. Available: https://doi.org/10.1145/1961189.1961199.

[14] T. Hofmann, B. Schölkopf, and A. J. Smola, "Kernel methods in machine learning," *The Annals of Statistics*, vol. 36, no. 3, pp. 1171–1220, 2008. DOI: 10.1214/009053607000000677. [Online]. Available: https://doi.org/10.1214/009053607000000677.

[15] J. Vert, K. Tsuda, and B. Schölkopf, "A primer on kernel methods," *Kernel Methods in Computational Biology, 35-70 (2004)*, Jan. 2004.

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

[17] B. Schölkopf, J. Platt, J. Shawe-Taylor, A. Smola, and R. Williamson, "Estimating support of a high-dimensional distribution," *Neural Computation*, vol. 13, pp. 1443–1471, Jul. 2001. DOI: 10.1162/089976601750264965.

[18] B. Schölkopf, A. J. Smola, R. C. Williamson, and P. L. Bartlett, "New support vector algorithms," *Neural Comput.*, vol. 12, no. 5, pp. 1207–1245, May 2000, ISSN: 0899-7667. DOI: 10.1162/089976600300015565. [Online]. Available: https://doi.org/10.1162/089976600300015565.

[19] A. Smola and B. Schölkopf, "A tutorial on support vector regression," *Statistics and Computing*, vol. 14, pp. 199–222, 2004.

[20] V. N. Vapnik and A. Y. Chervonenkis, "On the uniform convergence of relative frequencies of events to their probabilities," *Theory of Probability & Its Applications*, vol. 16, no. 2, pp. 264–280, 1971. DOI: 10.1137/1116025. eprint: https://doi.org/10.1137/1116025. [Online]. Available: https://doi.org/10.1137/1116025.

[21] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, "Scikit-learn: Machine learning in python," *J. Mach. Learn. Res.*, vol. 12, no. null, pp. 2825–2830, Nov. 2011, ISSN: 1532-4435.

[22] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and regression trees*. Routledge, 2017.

[23] L. Breiman, "Random forests," English, *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001, ISSN: 0885-6125. DOI: 10.1023/A:1010933404324. [Online]. Available: http://dx.doi.org/10.1023/A%3A1010933404324.

[24] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Review*, pp. 65–386, 1958.

[25] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, 1986.

[26] Y. LeCun, B. Boser, J. Denker, *et al.*, "Handwritten digit recognition with a back-propagation network," in *Advances in Neural Information Processing Systems*, D. Touretzky, Ed., vol. 2, Morgan-Kaufmann, 1989. [Online]. Available: https://proceedings.neurips.cc/paper/1989/file/53c3bce66e43be4f209556518c2fcb54-Paper.pdf.

[27] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," 1986.

[28] A. Graves, *Supervised Sequence Labelling with Recurrent Neural Networks*. Jan. 2012, vol. 385, ISBN: 978-3-642-24796-5. DOI: 10.1007/978-3-642-24797-2.

[29] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, pp. 1735–80, Dec. 1997. DOI: 10.1162/neco.1997.9.8.1735.

[30] H. Robbins and S. Monro, "A Stochastic Approximation Method," *The Annals of Mathematical Statistics*, vol. 22, no. 3, pp. 400–407, 1951. DOI: 10.1214/aoms/1177729586. [Online]. Available: https://doi.org/10.1214/aoms/1177729586.

[31] J. Cooley, P. Lewis, and P. Welch, "The finite fourier transform," *IEEE Transactions on Audio and Electroacoustics*, vol. 17, no. 2, pp. 77–85, 1969. DOI: 10.1109/TAU.1969.1162036.

[32] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex fourier series," *Mathematics of Computation*, vol. 19, pp. 297–301, 1965.

[33] F. Chollet *et al.*, *Keras*, https://keras.io, 2015.

[34] Martın Abadi, Ashish Agarwal, Paul Barham, *et al.*, *TensorFlow: Large-scale machine learning on heterogeneous systems*, Software available from tensorflow.org, 2015. [Online]. Available: http://tensorflow.org/.

[35] C. R. Harris, K. J. Millman, S. J. van der Walt, *et al.*, "Array programming with NumPy,", vol. 585, no. 7825, pp. 357–362, Sep. 2020. DOI: 10.1038/s41586-020-2649-2. arXiv: 2006.10256 [cs.MS].

[36] W. McKinney, "Data structures for statistical computing in python," Jan. 2010, pp. 56–61. DOI: 10.25080/Majora-92bf1922-00a.

[37] C. Shannon, "Communication in the presence of noise," *Proceedings of the IRE*, vol. 37, no. 1, pp. 10–21, Jan. 1949. DOI: 10.1109/jrproc.1949.232969. [Online]. Available: https://doi.org/10.1109/jrproc.1949.232969.

[38] M. Stone, "Cross-validatory choice and assessment of statistical predictions," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 36, no. 2, pp. 111–133, 1974. DOI: https://doi.org/10.1111/j.2517-6161.1974.tb00994.x. eprint: https://rss.onlinelibrary.wiley.com/doi/pdf/10.1111/j.2517-6161.1974.tb00994.x. [Online]. Available: https://rss.onlinelibrary.wiley.com/doi/abs/10.1111/j.2517-6161.1974.tb00994.x.

[39] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, no. 2, pp. 123–140, 1996, ISSN: 1573-0565. DOI: 10.1007/BF00058655. [Online]. Available: https://doi.org/10.1007/BF00058655.

[40] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2014. DOI: 10.48550/ARXIV.1412.6980. [Online]. Available: https://arxiv.org/abs/1412.6980.

[41] M. Vafaeipour, O. Rahbari, M. A. Rosen, F. Fazelpour, and P. Ansarirad, "Application of sliding window technique for prediction of wind velocity time series," *International Journal of Energy and Environmental Engineering*, vol. 5, no. 2, p. 105, 2014, ISSN: 2251-6832. DOI: 10.1007/s40095-014-0105-5. [Online]. Available: https://doi.org/10.1007/s40095-014-0105-5.