# Capability evaluation of deep learning for time-series prediction in medical packaging production

Diploma Thesis

for obtaining the academic degree

**Master of Science**

as part of the study

**Biomedical Engineering**

carried out by

**Valentin Frossard BSc**

student number: 1228981

*Institut für Verfahrenstechnik, Umwelttechnik und Technische Biowissenschaften*

at TU Wien (Vienna University of Technology)

Supervision

*Univ.Prof. Dipl.-Ing. Dr.techn. Anton Friedl*

*Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Johann Lohninger*

Vienna, November 2020

Valentin Frossard

## Abstract

With the increasing production volatility in the power grids due to renewable energy sources, an exact and individualized energy demand forecast of industrial customers is becoming more important to ensure the supply-demand balance at any time. In this thesis three commonly used deep learning algorithms, Artificial Neural Networks (ANNs), Convolution Neural Networks (CNNs) and Long-Short Term Memory Networks (LSTMs) are tested for their ability to predict the load demand of three industrial data sets one day in advance with an accuracy of 15 minutes. The results were compared to each other and to the method currently used by the industrial partners. It was found that deep learning algorithms can improve the prediction accuracy by 20% - 45% compared to the currently used method. Among the Deep Learning algorithms, LSTM achieved the best results, but only by 2% - 5% compared to the second-ranked algorithm, with a much higher training effort. Therefore, LSTM models generally are to be prefered over ANN or CNN algorithms. However, in cases requiring frequent retraining, using a second-ranked ANN or CNN algorithm may be advisable, as they are faster in training.

## Kurzfassung

Der steigende Anteil erneuerbar Energieträger führt zu Schwankungen in der Stromproduktion und reduziert damit aufgrund erhöhter Planungsunsicherheit die Netzstabilität. Um das Stromangebot und die Stromnachfrage im Gleichgewicht und somit die Netzstabilität zu gewährleisten, gewinnt eine genaue Bedarfsvorhersage zunehmend an Bedeutung. In dieser Arbeit werden Artificial Neural Networks (ANNs), Convolution Neural Networls (CNNs) und Long-Short Term Memory Networks (LSTMs), drei häufig verwendete Deep Learning Algorithmen, in ihrer Fähigkeit verglichen, den Strombedarf von drei Industriedatensets einen Tag vorab mit einer Genauigkeit von 15 Minuten vorherzusagen. Die Ergebnisse der Algorithmen wurden zueinander sowie mit der, von den Industriepartnern, aktuell verwendeten Methode verglichen. Die Deep Learning Algorithmen konnten, verglichen mit der aktuell verwendeten Methode, die Vorhersagegenauigkeit um 20% - 45% verbessern. Dabei erzielten die LSTM Modelle die besten Ergebnisse, allerdings nur um 2% - 5%, verglichen mit dem jeweils zweitbesten Algorithmus. Damit sind LSTM Modelle gegenüber ANN oder CNN Modellen grundsätzlich zu bevorzugen. Da allerdings LSTM Modelle wesentlich aufwendiger zu trainieren sind, kann es bei Problemstellungen, in denen häufiges neu-trainieren notwendig ist, vorteilhaft sein, ein zweitplatziertes ANN oder CNN Modell zu wählen.

Ich nehme zur Kenntnis, dass ich zur Drucklegung meiner Arbeit unter der Bezeichnung

**Diplomarbeit**

nur mit Bewilligung der Prüfungskomission berechtigt bin.

*Eidesstattliche Erkärung*

Ich erkläre an Eides statt, dass die vorliegende Arbeit nach den anerkannten Grundsätzen für wissenschaftliche Abhandlungen von mir selbständig erstellt wurde. Alle verwendeten Hilfsmittel, insbesondere die zugrunde gelegte Literatur, sind in dieser Arbeit genannt und aufgelistet. Die aus den Quellen wörtlich entnommenen Stellen, sind als solche kenntlich gemacht.

Das Thema dieser Arbeit wurde von mir bisher weder im In- noch Ausland einer Beurteilerin/einem Beurteiler zur Begutachtung in irgendeiner Form als Prüfungsarbeit vorgelegt. Diese Arbeit stimmt mit der von den Begutachterinnen/Begutachtern beurteilen Arbeit überein.

Wien, November 2020

Valentin Frossard

# Table of Abbreviations

| | |
|---|---|
| AR | Autoregressive Models |
| ARIMA | Autoregressive Integrated Moving Average |
| ARMA | Autoregressive Moving Average |
| BPTT | Backpropagation-Through-Time |
| CAE | Cumulative Absolute Error |
| CEC | Constant Error Carousel |
| CNN | Convolution Neural Network |
| ESM | Seasonal Exponential Smoothing Method |
| LSTM | Long Short-Term Memory |
| MA | Moving-Average Models |
| MAE | Mean Absolute Error |
| MLP | Multilayer Perceptron |
| MSE | Mean Square Error |
| NN | Neural Network |
| RMSE | Root Mean Square Error |
| RNN | Recurrent Neural Network |
| TF | Transfer Function |

## Preamble

This work was carried out as part of the master's programme in Biomedical Engineering. Due to data availability, deep learning algorithms for time series prediction were compared using power consumption data. Nevertheless, the gained knowledge is relevant for the biomedical engineering field and can be applied to biomedical engineering problems. For example, the algorithms can assist to make the health care system more efficient or to ease everyday life for patients with chronic diseases.

Recently, a variety of problem settings in health care have been approached using deep learning for time-series prediction. Masum et al. *(Masum et al. 2019)* applied LSTM, Bi-LSTM and CNN deep learning algorithms to forecast blood pressure 30 minutes in advance, aiming to detect critical conditions early. Similarly, Zhu et al. *(Zhu et al. 2020)* utilizes recurring neural networks to predict the upcoming glucose levels of Type I Diabetes patients 30 minutes ahead, to improve insulin application planning.

Shih and Rajendran *(Shih et al. 2019)* tested the performance of classical statistical methods (ARMA, ARIMA, ESM) and machine learning algorithms (ANN, multiple regression) predicting the blood supply in Taiwan to minimize outdating and shortage of stored blood.

Lastly, Kaushik et al. *(Kaushik et al. 2020)* predicted the average expenditures on pain medication using ARIMA, MLP and LSTM for supply demand planning.

**Acknowledgements**

Firstly, I would like to express my gratitude to my advisors, Univ.Prof Dipl.-Ing., Dr. Anton Friedl, Ao.Univ.Prof Dipl.-Ing. and Dr.techn. Johann Lohninger for their supervision of my work at Vienna University Technology. I am grateful for their trust and support they have placed in me and this project. I would also like to thank the Campfire Solutions' team for providing the necessary data. I want to especially extend my thanks to Dr. Benjamin Mörzinger. His supervision of the experiments and numerous constructive discussions with me have led to the effective progress of the work.

# Contents

# 1 Introduction

## 1.1 Introduction Electricity Markets

Since electricity cannot be stored easily, the amount of electric energy introduced into the electric grid must be equal the amount of electric energy consumed. To achieve equilibrium, producers and consumers are organized in balance groups and each group is liable to balance energy consumption and production within. This can be achieved by either adjusting demand and supply within the group or by purchasing and selling energy on the electricity spot markets.
Balance groups are geographically bundled and overseen by a control area manager ("Bilanzgruppenführer"). In case of a demand-supply-mismatch due to inaccurate demand prediction, power plant failures or availability changes of renewables, the control area manager restores equilibrium by utilizing primary ($<$ 30 sec), secondary ($<$ 15 min) or tertiary spinning reserve, consuming balancing energy. The balancing energy is provided by specialized, quick-reaction electric energy sources. Costs for the balance energy can be a multitude of the normal electricity rates and are charged to the balance group responsible for the mismatch. *(APG 2020)*

## 1.2 Forecasting Motivation

The global electric energy demand is increasing exponentially and expected to continue growing. As electric energy cannot be stored, the electric energy production and demand need to be in equilibrium. If the equilibrium is distorted, the network frequency raises with a surplus of energy or falls with energy shortage. Heavy distortions can cause equipment damage or even blackouts and must be avoided at all costs. Precise demand prediction enables power suppliers to plan for the future demand and match the production accordingly.

Inaccuracies in demand prediction can result in two major problems. First, an over-estimation will cause power suppliers to schedule higher capacities, unnecessary spinning reserve and will ultimately lead to a waste of energy resources and distribution inefficiencies *(Hong 2009)*. Second, an under-estimation of load prohibits providing sufficient load capacities, spinning reserve and causes high costs in the peaking unit, discouraging any economic and industrial development *(Hong 2009)*.

The European electricity consumption splits into three major segments: "households", "commercial & public service" and "industry", shown in *Figure 1.2*. Each segment represents approximately one-third of the consumption, with "industry" being the largest *(Agency 2020)*. With the rise of building automation and the wide adaption of smart meters, demand prediction for homes, office buildings and entire districts have been covered extensively by the scientific community *(Wei et al. 2018; Chen et al. 2017; Valgaev et al. 2017)*. However, the industrial sector has received limited attention. The reasons can only be assumed, but are most likely to be found in the lack of publicly available data. By partnering with *Campfire Solutions*, a start-up at the *TUW i²ncubator*, the author was able to obtain three extensive industrial electric power consumption datasets, that will be introduced in Section 4.2

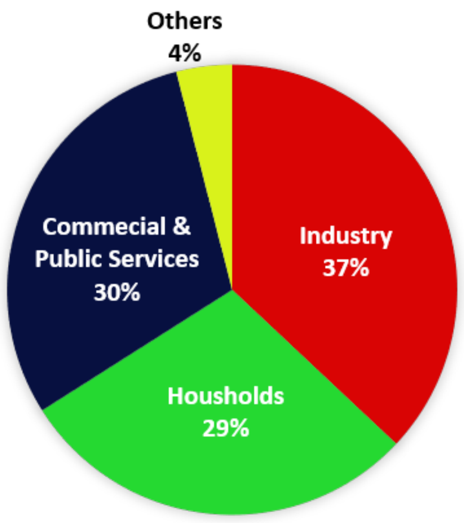

Figure 1: European Electricity Consumption by Consumer Type *(Agency 2020)*

## 1.3 Time Series Prediction

Electricity demand data is time series data, with a energy consumption measurement for every quarter-hour. Predicting future energy demand is a time-series prediction problem. Time series predictions usually aim to forecast a value at some future point in time $t + t'$ using available observations of a time series up to time $t$. Time series prediction problems can take various forms and are grouped according to the following categories. *(Han et al. 2019)*

### 1.3.1 Time Intervals of Observations

The majority of existing studies presume equidistant intervals of time for the data at hand. This condition simplifies data handling and the problem can be formulated as *(Han et al. 2019)*

$$\hat{x}_{t+h} = f(x_t, x_{t-1}, ..., x_{t-N}) \tag{1}$$

with $\hat{x}_{t+h}$ being the predicted result, $x_t, x_{t-1}, ..., x_{t-N}$ referring to the sequence the prediction is based upon and N being the number of inputs. $h$ can be 1 *(C.-H. Lee et al. 2014, Miranian et al. 2013)* for one-step ahead predictions or, for multi-step predictions, any positive integer *(Taieb et al. 2012, Parlos et al. 2000, Han et al. 2019)*.

In some cases equidistant time intervals cannot be assumed *(Ramasso et al. 2012, Aladag et al. 2012)* and the problem of time series prediction has to be extended by $l_1, l_2, ..., l_N$, representing the various time spaces *(Han et al. 2019)*

$$\hat{x}_{t+h} = f(x_{t-l_1}, x_{t-l_2}, ..., x_{t-l_N}, l_1, l_2, ..., l_N) \tag{2}$$

### 1.3.2 Prediction Strategy

To predict several time steps ahead, either a recursive or a direct prediction strategy can be deployed. The recursive prediction strategy is an iterative process *(Young*

*2012, Mirikitani et al. 2010, H. Liu et al. 2012)*, using subsequent predicted values to predict the next and can be expressed as follows *(Han et al. 2019)*

$$\begin{aligned}\hat{x}_{t+1} &= f(x_t, x_{t-1}, ..., x_{t-N})\\ \hat{x}_{t+2} &= f(\hat{x}_{t+1}, x_t, ..., x_{t-N+1})\\ &\cdots\\ \hat{x}_{t+M} &= f(\hat{x}_{t+M-1}, \hat{x}_{t+M-2}, ..., x_{t-N+M})\end{aligned} \tag{3}$$

with $M$ being the number of values to be predicted, $\hat{x}_t$ a predicted value, $x_t$ a measured value and $N$ the number of values used for one prediction. The disadvantage of the recursive prediction is the accumulation of prediction error and thereby the gradual deterioration of prediction accuracy.

For this reason some researchers *(Zhao et al. 2015, Han et al. 2016)* focus on predicting multiple data points at once. This direct prediction strategy can be expressed as *(Han et al. 2019)*

$$\hat{\mathbf{X}} = f(x_t, x_{t-1}, ..., x_{t-N}) \tag{4}$$

with $\hat{\mathbf{X}} = [\hat{x}_{t+1}, \hat{x}_{t+2}, ..., \hat{x}_{t+M}]^T$ being the predicted vector.

### 1.3.3 Univariate and Multivariate Modelling

A model with solely one input variable is a univariate model *(Pankratz 2009)* and with more than one input variable it is a multivariate model *(Khashei et al. 2011)*. Univariate models are defined according to *Equation 1* and multivariate models are defined as *(Han et al. 2019)*

$$\hat{x}_{t+1} = f(x_t^1, ..., x_{t-N}^1, x_t^2, ..., x_{t-N}^2, ..., x_t^L, ..., x_{t-N}^L) \tag{5}$$

with L representing the number of input variables.

## 1.4 Research Question and Objectives

The main research question is, which standard deep learning algorithm is best suited for day-ahead electric load forecasting in 15 minute accuracy for industrial production data?

The following objectives should be accomplished during this master thesis:

1. Obtain three industrial production datasets covering a minimum of 1.5 years
2. Select three standard deep learning algorithms to test
3. Construct a real-life baseline approach
4. Build a machine learning pipeline to pre-process data, train models and automatically perform hyper-parameter tuning
5. Select and suggest the model exhibiting the smallest error

# 2 State of the art

Electricity demand prediction models are classified by type, as introduced in *Section 2.1*, or the predicted time horizon. Models can be built to create short-term, medium-term and long-term forecasts *(Suganthi et al. 2012)*. Short-term forecasts targets time spans of hours to weeks, mid-term forecasting covers a duration of weeks to a year and long-term forecasts cover anything beyond *(Suganthi et al. 2012)*.

## 2.1 Model Types

Prediction problems can be solved by utilizing white-box, black-box or grey-box approaches. White-box models rely on a deep understanding and mathematically modelling of underlying physical or chemical processes and their boundary conditions for both, building operations and production activities *(Ferrarini et al. 2019)*. Using white-box models, a forecast can be created by solving the modelled set of equations for every point in time. White-box models provide good interpretability, however the model creation requires in-depth process knowledge, is labour intensive and solely applicable to the process the model was created for. Ferrarini et al. *(Ferrarini et al. 2019)* applied white-box modelling to create an energy model of a multi-storey residential building and Vaccaro et al. *(Vaccaro et al. 2012)* used a white-box model as a baseline in wind intensity predictions.

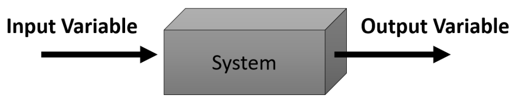

Figure 2: Black-Box concept inspired by *(Bunge 1963)*

Contrary, a black-box model is a purely data-driven approach. The system is viewed as unknown entity and its properties are derived ("learned") from the relationship between input and output variables *(Bunge 1963)*, as shown in *Figure 2*. Black-box models do not require system specific expert knowledge and algorithms can be easily applied to multiple problem settings of the same kind. On the down-side, black-box models lack transparency and interpretability of derived predictions, not being able to explain why a specific decision has been made *(Holzinger et al. 2017)*. Deep learning algorithms are black-box models, since their high level of recursiveness limits interpretability *(Rudin 2019)*.

A hybrid form between white-box and black-box models can be classified as grey-box models. Grey-box models have been applied for energy demand prediction in office and residential buildings *(Berthou et al. 2014, Li et al. 2016, Afram et al. 2018)* and to predict output power of both, photovoltaic *(Paulescu et al. 2017)* and wind power plants *(Ferrarini et al. 2019)*.

## 2.2 Short-term load forecasting using statistical methods

In the field of short-term load forecasting, any type of timeseries forecasting technology being en-vogue at that time, has been applied, aiming to improve prediction performance. Historically Box and Jenkins models have shown to be well-suited in forecasting short term loads *(Hagan et al. 1987)*, which have then been surpassed by ARIMA models *(Amjady 2001)*

Historically, many different statistical methods have been deployed in the field of short-term load forecasting. Moghram and Rahman *(Moghram et al. 1989)* compared the performance of multiple linear regression, stochastic time series, general exponential smoothing, state space method and a knowledge-based approach, which can be considered traditional statistical forecasting methods.

### 2.2.1 Multiple Linear Regression

In the multiple linear regression model, the load is calculated by inferring from explanatory variables and the model is expressed according to Moghram and Rahman *(Moghram et al. 1989)* as

$$y(t) = a_0 + a_1 x_1(t) + ... + a_n x_n(t) + a(t) \tag{6}$$

| | |
|---|---|
| $y(t)$ | electrical load |
| $x_n(t)$ | explanatory variable correlated with $y(t)$ |
| $a(t)$ | random variable with zero mean and constant variance |
| $a_n$ | regression coefficients |

These explanatory variables are identified by performing a correlation analysis between the independent variables and the load. The regression coefficients, used to calculate the prediction, are determined using the least square estimation technique. To determine the significance of the regression coefficients, statistical tests, for example the F-Test, are performed. *(Moghram et al. 1989)*

### 2.2.2 Stochastic Time Series

Stochastic time series modelling infers the prediction as output of a linear filter with random series input, zero mean and unknown fixed variance. Depending on the filter used, Autoregressive Models (AR), Moving-Average Models (MA), Autoregressive Moving-Average Models (ARMA), Autoregressive Integrated Moving Average Models (ARIMA), Seasonal Models, as an extension to AR, MA, ARMA or ARIMA, or Transfer Function (TF) Models can be built *(Moghram et al. 1989)*.

#### 2.2.2.1 Autoregressive Models (AR)

In the autoregressive model, the current values of $y(t)$ is inferred linearly from its previous values $y(t-1), ..., y(t-p)$ and a random noise $a(t)$. The order $p$ of the model depends on the temporal oldest value used to calculate the predicted value. The AR model can be written as *(Moghram et al. 1989)*

$$y(t) = \Phi_1 y(t-1) + \Phi_2 y(t-2) + ... + \Phi_p y(t-p) + a(t) \tag{7}$$

| | |
|---|---|
| $y(t)$ | electrical load |
| $y(t-n)$ | historic values of y(t) |
| $a(t)$ | random variable with zero mean and constant variance |
| $\Phi_n$ | weight determining the importance of historic value, based on previous random noises |

#### 2.2.2.2 Moving-Average Models (MA)

In the moving-average model, the current value $y(t)$ of the time series, is inferred linearly from current and previous values of a white noise series $a(t), ..., a(t-q)$. The noise series is calculated from the forecast errors between prediction and true value, once the true value becomes available. The order $q$ of the model depends on the oldest considered value $a(t-q)$ *(Moghram et al. 1989)*. According to Moghram and Rahman *(Moghram et al. 1989)*, the model can be written as

$$y(t) = a(t) - \Theta_1 a(t-1) - \Theta_2 a(t-2) - ... - \Theta_q a(t-q) \tag{8}$$

| | |
|---|---|
| $y(t)$ | electrical load |
| $a(t)$ | white noise |
| $a(t-n)$ | previous values of white noise |
| $\Theta_n$ | weighting factor |

#### 2.2.2.3 Autoregressive Moving-Average Models (ARMA)

In the autoregressive moving-average model, the current value of the time series $y(t)$, is inferred linearly from values of previous periods $y(t-1), ..., y(t-p)$ as well as current and previous values of a white noise series $a(t), ..., a(t-q)$. The order of the model depends on the oldest previous series value and oldest previous white noise value used *(Moghram et al. 1989)*. According to Moghram and Rahman *(Moghram et al. 1989)*, the model can be written as

$$y(t) = \Theta_1 y(t-1) + ... + \Theta_p y(t-p) + a(t) + \Phi_1 a(t-1) + ... + \Phi_q a(t-q) \tag{9}$$

| | |
|---|---|
| $y(t)$ | electrical load |
| $y(t-n)$ | historic values of y(t) |
| $a(t)$ | white noise |
| $a(t-n)$ | previous values of white noise |
| $\Theta_n$ | weighting factor for AR part |
| $\Phi_n$ | weighting factor MA part |

#### 2.2.2.4 Autoregressive Integrated Moving-Average Models (ARIMA)

The previously described models presume stationary processes, bearing that the mean and covariance among observations do not change with time. Having a non-stationary process, the observations first have to be transformed by differencing $d$ times until the process becomes stationary.This is achieved by introducing the $\nabla$ operator, with $\nabla^d$ representing the $d^{th}$ differentiation and $\nabla^d y(t)$ the $d^{th}$ derivative of ARIMA model $y(t)$. To obtain the original time series $y(t)$, $\nabla^d y(t)$ hast to be integrated $d$ times. *(Moghram et al. 1989)*

According to *(Moghram et al. 1989)*, the differential model of the ARIMA can be written as

$$\Phi(B)\nabla^d y(t) = \Theta(B)a(t) \tag{10}$$

| | |
|---|---|
| $y(t)$ | electrical load |
| $a(t)$ | white noise |
| $\Theta$ | weighting factor for AR part |
| $\Phi$ | weighting factor MA part |
| $B$ | backshift operator |
| $\nabla$ | differential operator |

with the backshift operator $B$ connecting $y(t-1) = By(t)$.

#### 2.2.2.5 Seasonal Models

Due to daily, weekly, yearly or other effects, many time series exhibit a periodic behaviour. The AR, AM, ARMA and ARIMA models can be utilized, but require to be extended by seasonal terms. According to Moghram and Rahman *(Moghram et al. 1989)* the extended ARIMA model *(Equation 11)* shows the ARIMA model of *Section 2.2.2.4* extended by the seasonal terms $\phi(B^S)$, $\nabla^d {\nabla^D}_S$ and $\theta(B^S)$, capable of modelling one seasonality.

$$\Phi(B)\phi(B^S)\nabla^d {\nabla^D}_S y(t) = \Theta(B)\theta(B^S)a(t) \tag{11}$$

| | |
|---|---|
| $y(t)$ | electrical load |
| $a(t)$ | white noise |
| $\Theta$ | weighting factor for AR part |
| $\theta(B^S)$ | weighting factor AR part seasonal |
| $\Phi$ | weighting factor MA part |
| $\phi(B^S)$ | weighting factor MA part seasonal |
| $B$ | backshift operator |
| $\nabla$ | differential operator |
| ${\nabla^D}_S$ | seasonal differential operator |

Of course, the model introduced above can be further extended to account for two or more seasonalities, simply by adding the respective terms for further periodicity as introduced by Moghram and Rahman *(Moghram et al. 1989)*

$$\Phi(B)\phi(B^S)\phi'(B^{S'})\nabla^d\nabla^D{}_S\nabla^{D'}{}_{S'}y(t) = \Theta(B)\theta(B^S)\theta'(B^{S'})a(t) \tag{12}$$

| | |
|---|---|
| $y(t)$ | electrical load |
| $a(t)$ | white noise |
| $\Theta$ | weighting factor for AR part |
| $\theta(B^S)$ | weighting factor AR part seasonal |
| $\theta'(B^{S'})$ | second weighting factor AR part seasonal |
| $\Phi$ | weighting factor MA part |
| $\phi(B^S)$ | weighting factor MA part seasonal |
| $\phi'(B^{S'})$ | second weighting factor MA part seasonal |
| $B$ | backshift operator |
| $\nabla$ | differential operator |
| $\nabla^D{}_S$ | seasonal differential operator |
| $\nabla^{D'}{}_{S'}$ | second seasonal differential operator |

### 2.2.2.6 Transfer Function Models

The models introduced in *Section 2.2.2.1* to *Section 2.2.2.5* express univariate time-series $y(t)$, based on their history and white noise. To account for the effect of other variables, a transfer function model is combined with a noise model $n(t)$ discussed in *Section 2.2.2.1* to *Section 2.2.2.5* , as shown in *Figure 3 (Dale 1981)*.

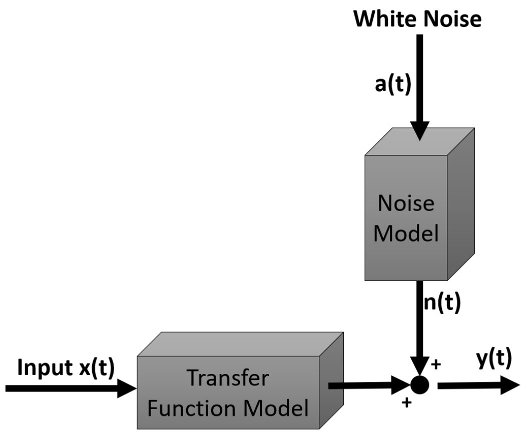

Figure 3: Transfer Function Model according to Moghram and Rahman *(Moghram et al. 1989)*

### 2.2.3 General Exponential Smoothing

Standard exponential smoothing (Holt-Winters Exponential Smoothing) was introduced by Winters *(Winters 1960)* and is able to capture one pattern of seasonality. An additive trend is assumed and the local slope $T_t$ is estimated by smoothing subsequent differences $(S_t - S_{t-1})$ on a local level $S_t$. The local seasonal index $I_t$ is determined by smoothing the ratio of observed value $X_t$ to local level $S_t$ *(Taylor 2003)*. According to Taylor *(Taylor 2003)*, the k-step-ahead forecast $\hat{X}_t(k)$ can be calculated by

$$\hat{X}_t(k) = (S_t + kT_t)I_{t-s+k} \tag{13}$$

with the **level** defined as

$$S_t = \alpha \frac{X_t}{I_{t-s}} + (1-\alpha)(S_{t-1} + T_{t-1}) \tag{14}$$

the **trend** defined as

$$T_t = \gamma(S_t - S_{t-1}) + (1-\gamma)T_{t-1} \tag{15}$$

and **seasonality** defines as

$$I_t = \delta \frac{X_t}{S_t} + (1-\delta)I_{t-s} \tag{16}$$

| | |
|---|---|
| $X_t$ | observed value |
| $\hat{X}_t(k)$ | k-step-ahead forecast |
| $\alpha, \gamma, \delta$ | adjustable smoothing parameters |
| $T_t$ | local slope |
| $S_t$ | local level |
| $I_t$ | local s-period seasonal index |

Standard exponential smoothing is limited to one seasonality. Taylor *(Taylor 2003)* extended the standard exponential smoothing to be better suited for short-term electricity demand forecasting by adding multi-seasonal capabilities.

## 2.3 Short-term forecasting using machine learning

Machine Learning algorithms can be categorized in two major architecture types: Generative Models and Discriminative Models. Generative models converge faster whereas discriminative models provide a better asymptotic error performance *(B. Liu et al. 2010)*. Therefore, generative models are preferably used with little trainings data available and a discriminative approach is used with enough data at hand *(B. Liu et al. 2010)*.

### 2.3.1 Generative Models

Generative models learn the joint probability $p(x, y)$ of inputs x and target y. Predicitons are made by calculating $p(y|x)$ for all $y$ for a given $x$ and selecting the most likely as prediction for $y$ *(Ng et al. 2002)*.

Examples for generative models are Gaussian Process, Bayesian Networks, Hidden Markov Model, Restricted Boltzman Machine, Deep Believe Network and Generative Adverseral Network *(Han et al. 2019)*.

### 2.3.2 Discriminative Models

Discriminative models learn a direct mapping of input x to target y, minimizing the error function (decision risk) without estimating $p(x, y)$, $p(x|y)$ or $p(y|x)$ *( B. Liu et al. 2010)*.

#### 2.3.2.1 Artificial Neural Network (ANN)

Neural networks are universally applicable and have been deployed to problems ranging from regression, classification to feature extraction, inference and others. The most common neural network configuration is the so-called Multilayer Perceptron (MLP) combined with Back Propagation *(Hecht-nielsen 1992, Wong 1991)*.

An artificial neural network resembles the neural structure of the human cortex. A single neuron is shown in *Figure 4* with $x$ being the input, $w$ being the weight of the edge, $b$ the neurons bias, $y$ the output and $F(x)$ the activation function. $F(x)$ calculates the neuron output based on the sum of the neuron inputs. Based on *Figure 4*, the neuron can be described as *(S.-C. Wang 2003)*

$$y = F(x * w + b) \tag{17}$$

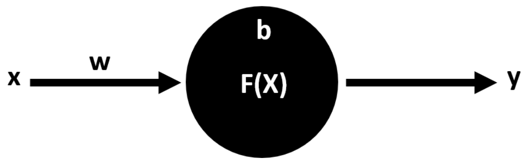

Figure 4: Neuron model according to *(S.-C. Wang 2003)*

Multiple neurons grouped in layers form the ANN, with the first layer denoted as input layer and the last layer as output layer. *Figure 5* depicts a simple ANN. The output of the model is calculated by forward propagation and the model is refined during training using back propagation. *(Hirasawa et al. 1996, Che et al. 2011)*.

ANNs have been used for forecasting tasks and have been combined with other techniques to improve prediction performance. Mellit and Pavan *(Mellit et al. 2010)* used an ANN to predict the solar irradiance of a PV plant in Italy. Heng et al. *(Heng et al. 1998)* introduced an ANN with genetic algorithm to perform short term load forecasting, which was subsequently extended by Liao et al. *(Liao et al. 2006)* to a fuzzy neural network with chaos genetic algorithms to perform short term load forecasting. Recently, Eseye et al. *(Eseye et al. 2019)* tested various feature selection

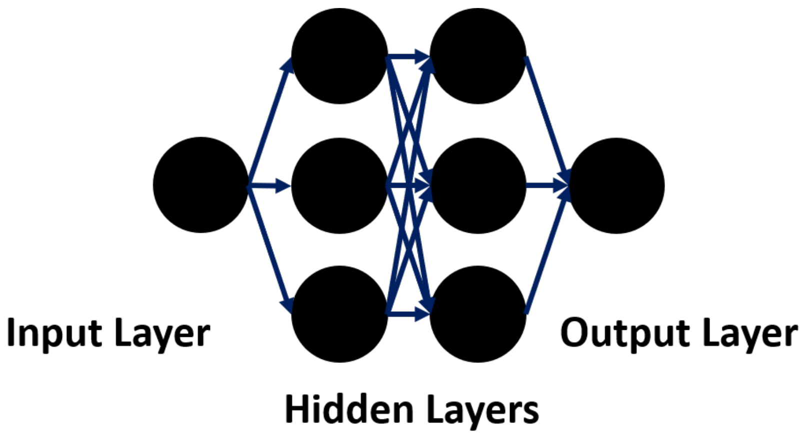

Figure 5: Artificial Neural Network with two Hidden Layers

algorithms and evaluated the forecasting performance of the selected feature subsets using a feedforward artificial neural network. "Dynamic neural network" introduced by Mordjaoui et al. *(Mordjaoui et al. 2017)* is a dynamic load forecasting approach based on artificial neural networks, able to adaptively learn patterns from the data.

Recurrent Neural Networks extend classical neural networks by connecting the neurons' output to their inputs, creating a closed loop and thus gaining the ability to memorize information involving trend and tendency. The training process is called Backpropagation-Through-Time (BPTT) *(Werbos 1990)* and has been described by Connor et al. *(Connor et al. 1994)*, Anbazhagan and Kumarappan *(Anbazhagan et al. 2012)* and Guresen et al. *(Guresen et al. 2011)*.

#### 2.3.2.2 Convolutional Neural Network (CNN)

CNNs are inspired by a unique neuron structure in a cat's receptive fields, discovered in 1959 by two neurobiologists, Hubel and Wiesel *(Hubel et al. 1959)*. LeCun and Bengio *(LeCun et al. 1995)* initially applied CNNs for speech, time series and image processing in 1995.

Similar to ANNs, a CNN consists of an input layer, multiple hidden layers and an output layer. Each layer contains activation functions such as Rectified Linear Unit (ReLu). Hidden layers come in three types: fully connected layers, convolutional layers and pooling layers. Convolutional layers learn to extract specific features of the sample (i.e. edges in an image) whereas pooling removes unnecessary information reducing the matrix size for subsequent processing and increasing processing speed. Fully connected layers have the neuron architecture of an MLP, connecting all of their input neurons and generating the output of the network *(Han et al. 2019)*. Deep believe networks, a generative model, use a similar technique, but adding probability in the pooling process *(K. Lee et al. 1992)*.

For univariate sequences, the CNN operates using a set of filters in the convolution layer and the output of the layer is obtained by computing the dot product between overlapping input and the filter weights in an autoregressive manner *(Han et al. 2019)*. Multivariate sequences can be thought of as 2D images, with time being one axis and variables the other *(Han et al. 2019)*.

In literature CNNs have been used to predict chaotic as well as real-world time series *(Han et al. 2019)*. In the field of autonomous driving, Hoermann et al. *(Hoermann et al. 2018)* applied CNNs to predict data from multiple sensors in a down town scenario with multiple road users. Ding et al. *(Ding et al. 2015)* applied a dilated CNN architecture for event driven stock market prediction and Wang et al. *(H.-z. Wang et al. 2017)* deployed a combination of wavelet transform and CNNs for wind power forecasting. Further, CNNs have been heavily used for image recognition *(Han et al. 2019)*.

#### 2.3.2.3 Long-Short Term Memory (LSTM)

RNNs, as introduced in *Section 2.3.2.1* suffer from a vanishing or exploding gradient problem. The BPTT procedure depends exponentially on the weights for each timestep, failing to learn information over more than 5-10 timesteps *(Han et al. 2019)*. The Long-Short Term Memory (LSTM) extends the RNN architecture by introducing a linear unit, the Constant Error Carousel (CEC) holding information that can be added to each timestamp. The error flow control of the CEC is performed using 'gates', the input gate controls the information added to the cell, the output gate controls the information contributed to the rest of the network and the forget gate controls the decay of the information. Therefore, LSTMs can maintain temporal information for many timesteps and it has been used in sequential data analysis, prediction and classification tasks *(Hochreiter et al. 1997, Filonov et al. 2016)* for both, univariate and multivariate *(Fu et al. 2016, Filonov et al. 2016)* problems.

## 2.4 Standardization and Normalization

Machine learning models learn a mapping from input variables to output variables. These variables might be given in different units (kWh, GWh, sec, hr, ...) and are likely to be differently scaled. Differences in scaling may increase the complexity of the problem being modelled and lead to large weight values of the model. Models with large weights are often unstable and prone to bad generalization abilities *(Brownlee 2020)*. Standardization or normalization techniques can be applied to mitigate this problem.

### 2.4.1 Normalization

Using normalization, the data is rescaled to a fixed range, usually to the range of 0 and 1, by subtracting the minimum value and dividing by the range of minimum and maximum value *(Witten et al. 2002)*. Clearly, the minimum and maximum observable values have to be known or estimable *(Brownlee 2020)*. A value is normalized as follows *(Brownlee 2020)*

$$y = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{18}$$

with $x_{min}$ being the minimum value and $x_{max}$ the maximum value.

A value outside the minimum-maximum range is going to transform to a value below 0 or above 1.

### 2.4.2 Standardization

Standardization is rescaling the observed data distribution to a mean of zero and a standard deviation of one. This is achieved by subtracting the mean from each value and dividing the result by the standard deviation *(Witten et al. 2002)*. A value is standardised as follows *(Brownlee 2020)*

$$y = \frac{x - \mu}{\sigma} \tag{19}$$

Using standardization can prove more robust than normalizing, especially in respect to outliers. Furthermore, the values are not limited to a specific range. In this work, *RobustScaler* of scikit-learn is used to standardize data. *RobustScaler* uses percentiles to determine mean and standard deviation, making it even more resilient against outliers *(s. scikit 2020)*.

# 3 Methodology

## 3.1 Data Analysis Pipeline

A standardized process is utilized to perform the machine learning experiments and *Figure 6* depicts the overall data analysis pipeline. First, the data is loaded and all necessary representation specific preprocessing (merging of files, adjustment of the representation of numbers, ...) is performed. Second, the data is manually inspected (*Section 3.1.1*). If a data specific preprocessing is necessary, it is conducted as third step of the data analysis pipeline (*Section 3.1.2*). The implementation of the first three steps is highly specific for each dataset, aiming to transform each into a coherent data structure usable by all following processing steps. After pre-processing, a baseline, serving as a performance reference for more elaborate algorithms, is established (*Section 3.2*). All processes relevant for training deep learning models are discussed in *Section 3.3*, relying on the baseline to discriminate well and poorly performing algorithms. Finally, the results are compared across all datasets available in *Section 5*.

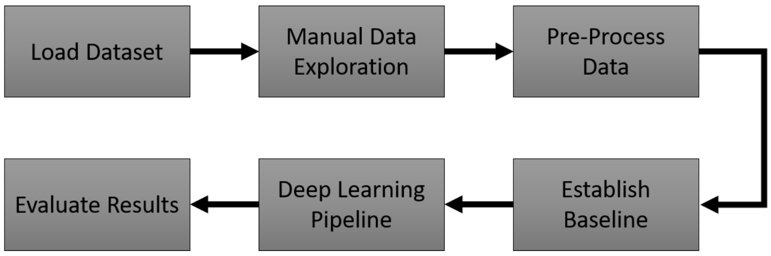

Figure 6: Applied Data Analysis Pipeline

### 3.1.1 Manual Data Exploration

The goal of the manual data exploration is to get a feeling for the processed data and to answer the following key questions:

(a) Do outliers exist?

(b) Is there a significant trend?

(c) Are recurring patterns observable?

For the use-case of electric demand prediction, load peaks (outliers) can worsen the prediction outcomes and must be removed. Outlier removal is introduced in *Section 3.1.2*.

A trend is a continuous change, for example an increase or a decrease, of electricity demand over time, beyond seasonal patterns. Since neither of the datasets introduced in *Section 4.2* experiences trend, it will not be elaborated further.

The presence of recurring patterns impacts feature selection for the algorithms used. All available datasets are experiencing recurring daily, weekly and annual patterns, as shown in *Section 4.2*. Therefore, the univariate power consumption is extended by calendric features (*Section 3.3.1*), to broaden the algorithms' prediction basis and improve prediction performance.

### 3.1.2 Pre-Processing Data

The only dataset requiring further pre-processing is Dataset 1 by Company A introduced in *Table 1*, showing major load peaks in manual data exploration (*Section 4.2.1*).

Without considering these load peaks as outliers and removing them, the regular electric load pattern would be reduced in importance during normalization and significantly harm prediction performance.

*Listing 1* shows the code used to remove the outliers. The z-score threshold of *Line 4* is selected empirically based on a all-time load plot, to remove all outliers excelling the regular electric load pattern.

Listing 1: Outlier removal for Dataset 1 according to *(stackoverflow 2020(c))*

```
1  import pandas as pd
2  from scipy import stats
3  df = pd.read_pickle("dataset_1.pkl")
4  mask = (np.abs(stats.zscore(df['power'])) > 0.18)
5  df.power = df.power.mask(mask).interpolate()
```

## 3.2 Baseline

A baseline is a meaningful reference point, a basis for the comparison of results of deep learning algorithms. Usually, a baseline is the simplest possible prediction, a random result or the most common prediction approach *(Brownlee 2014)*. This work utilizes the prediction method used by Company B and Company C, the so called "last-year-consumption" approach, to construct a baseline. In the "last-year-consumption", the electric energy consumption of the previous year is shifted to match the corresponding week days (Monday to Monday, Tuesday to Tuesday, ...) of the current year and is compensated for holidays according to *Listing 2*. The for loop of *Line 18* creates the averages to compensate past holidays, becoming work days, in the prediction period, with the average consumption of a workday (*"avg_workday"*) and to compensate past work days turning into holidays with the average consumption of a holiday (*"avg_holiday"*). In the for-loop of *Line 30*, those averages are applied to the prediction in cases of holiday/workday mismatches.

Listing 2: Code generating the Baseline

```
1  data #Standardized electric load consumption data as pandas dataframe
2  # 2018 is used as prediction for 2019
3  split_date = datetime.datetime(2019,1,1,0,0,0)
4  days = datetime.timedelta(364)
5  start_date=split_date-days #Algorithm takes values starting from beginning of
       2018
6  stop_date = split_date + datetime.timedelta(365) #Algorithm takes values until
       end of 2019
7  historic_data = data.loc[start_date:stop_date]
8  historic_index = historic_data.index
9  future_index = data.loc[split_date:stop_date].index
10 future_year = np.zeros(data.loc[split_date:stop_date].shape[0])
11
12 past_year_shape = historic_data.shape[0] - future_year.shape[0]
13 count_offdays = 0
14 sum_offdays = 0
15 count_workday = 0
16 sum_workday = 0
17
```

```
for i in range(0,past_year_shape):
        ci = historic_index[i] #ci = current index
        if (ci in at_holidays) or (calendar.weekday(ci.year, ci.month, ci.day)
            >=5):
                count_offdays = count_offdays + 1
                sum_offdays = sum_offdays + historic_data.iloc[i:i+1].values
                    [0][0]
        else:
                count_workday = count_workday + 1
                sum_workday = sum_workday + historic_data.iloc[i:i+1].values
                    [0][0]

avg_workday = sum_workday/count_workday
avg_offday = sum_offdays/count_offdays

for i in range(0, future_year.shape[0]):
        if(historic_index[i] in at_holidays)and(future_index[i] in at_holidays):
                future_year[i] = historic_data.iloc[i:i+1].values[0][0];
        else:
                if(future_index[i] in at_holidays):
                        future_year[i] = avg_offday;
                else:
                        if(historic_index[i] in at_holidays):
                                future_year[i] = avg_workday
                        else:
                                future_year[i] = historic_data.iloc[i:i+1].
                                    values[0][0];
```

## 3.3 Deep Learning Pipeline

Similar to the data analysis pipeline introduced in *Figure 6*, the deep learning experiments are implemented as a modular process, depicted in *Figure 7*.

The process starts by loading the pre-processed data and transforming the univariate electric load data into a multivariate dataset by adding calendric information as additional features (*Section 3.3.1*). Adding calendric information simplifies learning date and time based recurring patterns. Next, the available data is split into training and test sets (*Section 3.3.2*) and the actual training loop is started (*Section 3.3.3*). Finally, the methods to compare the results for each algorithm and dataset are introduced in *Section 3.4*

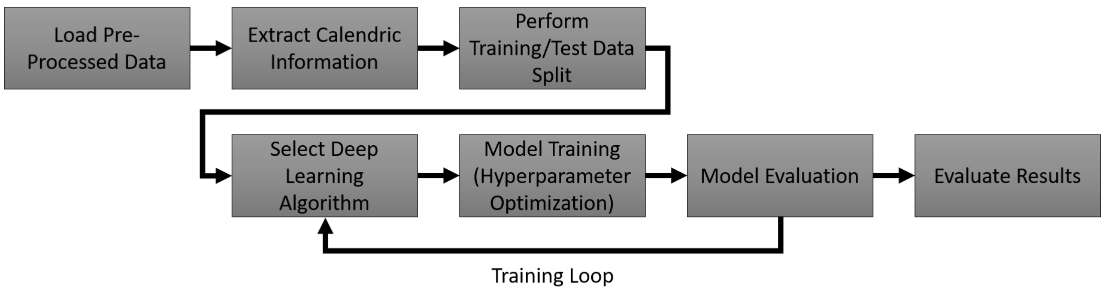
Load Pre-Processed Data
Extract Calendric Information
Perform Training/Test Data Split
Select Deep Learning Algorithm
Model Training (Hyperparameter Optimization)
Model Evaluation
Evaluate Results
Training Loop

Figure 7: Deployed Deep Learning Pipeline

### 3.3.1 Calendric Information Extraction

The datasets described in *Section 4.2* are limited to univariate historic electric consumption data, making it difficult to identify recurring patterns. Electric load demand is considered to be highly correlated with calendaric events, as for example time of day, weekday, overall season or holidays.

Providing algorithms with primed calendaric information enables them to directly learn the correlation between time patterns and energy consumption. *Listing 3* depicts the function used to add calendric information to the dataset.

Listing 3: Calendric Feature Extraction

```
import holidays

def extract_calendric_information(data):
        at_holidays = holidays.Austria()

        data['dow']=data.index.dayofweek
        data['m_year']=data.index.month
        data['q_hour'] = np.round(data.index.minute/15)
        data['hour']=data.index.hour
        data['isWorkday']=1 #Initialising 'isWorkday' with 1
        data['beforeHoliday']=0 #Initialising 'beforeHoliday' with 0
        filter1 = data.index.dayofweek==5 #Saturday
        filter2 = data.index.dayofweek==6 #Sunday
        #Setting 'isWorkday' to zero for Saturdays
        data.loc[filter1, 'isWorkday'] = 0
        #Setting 'isWorkday' to zero for Sundays
        data.loc[filter2, 'isWorkday'] = 0

        #The following loop iterates over all datapoints and assesses if
        #the datapoint is indeed a workday and the next day will be a workday.
        for index, row in data.iterrows():
                if(index in at_holidays):
                        data.loc[index, 'isWorkday']=0
                index_offset = index+pd.DateOffset(1)
                if(index_offset.dayofweek >= 5 or index_offset in at_holidays):
                        data.loc[index, 'beforeHoliday'] = 1

        return data
```

### 3.3.2 Training/Test Split

Machine learning algorithms infer rules, based on the data they are trained on, becoming proficient on the training data. As the algorithms are deployed on new, unseen data, it is important to assess their generalization abilities. This is achieved by applying the algorithms to the test set, comprised by data not used for training. It is created by splitting the provided dataset into a larger training set and a smaller test set. The test set must never be used for training or training decisions. Therefore, the training set is split further to create the actual training set and a validation set. The validation set is used during the model training process to determine the models performance after every epoch.

The algorithms in this work are designed to learn annual patterns, thus the minimum duration of the training data must be at least one year, to model one full annual cycle and the minimum duration of the test data should be one year too, to measure the performance on one full annual cycle. Dataset 2 is the smallest, providing a total of 21 month from January 2018 to September 2019 (*Section 4.2.2*). As the other datasets cover significantly larger timespans and include the year 2019 too, Tuesday

01.01.2019 00:00 is selected as universal training/test split date, defining all measurements of 2019 as test data. *Listing 4* shows the code used to split training and test set. During training, the training set will be randomly split into actual training and validation data for each subsequent training epoch, preventing the model from being subject to bias.

Listing 4: Performing Training/Test Split according to *(stackoverflow 2020(b))*

```python
import pandas as pd
import datetime

## This function requires a splitting date, not a splitting index ##
def training_test_split (data, splitting_date):
        training = data.loc[data.index<splitting_date]
        test = data.loc[data.index >= splitting_date]
        return training, test

split_date = pd.datetime(2019,1,1,0,0,0)
training, test = training_test_split(data.copy(),split_date)
```

### 3.3.3 Model Training Process

#### 3.3.3.1 Normalization

The deep learning algorithms used for this thesis operate best, if the provided data is mapped to the range of 0 to 1. This is achieved by normalizing the training set as introduced in *Section 2.4.1*, using MinMaxScaler of the sklearn library. *Listing 5* depicts the respective code.

Listing 5: Instatiation of normalization object according to *(scikit 2020)*

```python
scalingTransformer = MinMaxScaler(feature_range=(0,1)).fit(training.copy())
```

*scalingTransformer* is calibrated using only the training set, assuming the test set would not be available in production setting at the time of training. It is applied to the datasets in the *create_supervised_trainingset* function (*Line 10*), introduced in *Listing 6*

#### 3.3.3.2 Sample Generation

The deep learning algorithms require a supervised dataset with each input $X$ mapped to the corresponding output $Y$ and each of these input-output pairs is called a sample. The extended electric load demand is provided in a stream of datapoints as shown in *Figure 8*, one measurement for each feature every 15 minutes and has to be transformed to constitute a supervised problem. *Listing 6* shows the function *create_supervised_trainingset*, used to dynamically transform the raw data.

Listing 6: Transforming Time Series Data to Supervised Learning Problem inspired by *(Brownlee 2017)*

```python
1  def create_supervised_trainingset(data, output_shape, n_steps_in, n_steps_out,
       time_shift=1, univariate=True,  scalingTransformer=None):
2          n_steps_in = int(n_steps_in)
3          n_steps_out = int(n_steps_out)
4          time_shift = int(time_shift)
5          #Normalization has to be done, before creating the supervised dataset
6          #scalingTransformer has been initialized in Listing 5
7
```

```
8       if(scalingTransformer!=None):
9           orig = data.copy()
10          data = pd.DataFrame(scalingTransformer.transform(data))
11
12      if(time_shift<1):
13          return None
14      else:
15          X,y = list(),list()
16          i=0
17          while (i+n_steps_in+n_steps_out) < data.shape[0]:
18              end_x = i+n_steps_in
19              end_y = end_x + n_steps_out
20              seq_x  = data.iloc[i:end_x,:].to_numpy()
21              seq_y = data.iloc[end_x:end_y,0].to_numpy()
22              X.append(seq_x)
23              y.append(seq_y)
24              i = i+time_shift
25          Xa = np.array(X)
26          ya = np.array(y)
27          if(univariate):
28              Xa=Xa[:,:,0]
29              if(output_shape == "3D"):
30                  Xa = np.expand_dims(Xa,-1)
31          ## Multivariate datasets generate a three-dimensional
                datastructure. For ANN/MLP algorithms a two-dimensional
                structure is required.
32          if(output_shape=="2D" and Xa.ndim>2):
33              ## Flattening, with keeping the amount of samples the
                    same ##
34              Xa = Xa.reshape(Xa.shape[0], Xa.shape[1]*Xa.shape[2])
35
36      #2D: [sample, timesteps]
37      #3D: [samples, timesteps, features]
38      return Xa, ya, data,orig
```

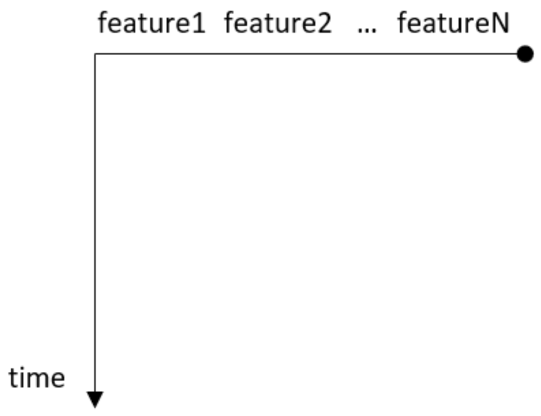

Figure 8: Structure of original data

In *Line 17* to *Line 26*, the data is transformed to a three-dimensional structure. Assuming multivariate data with N features, the original dataset is grouped into samples with a fixed number (*#steps_in*) of measurements per feature to form the input $X$ and *#steps_out* subsequent measurements of the target feature *feature1* are grouped to form the output $Y$. This work aims to predict a full day, day-ahead and therefore *#steps_out* is set to 96, the number of quarter hours a day. *#steps_in* is a variable for hyperparameter optimization, introduced in *Section 3.3.3.4*. *Line 24* utilizes the variable *time_shift*, determining the step-size between subsequent samples. *time_shift = 1* produces the maximum overlap, shifting only 15 minutes for each sample whereas *time_shift* = *96* shifts the subsequent samples by a full day, not producing an overlap. This is directly impacting the number of samples drawn from a given dataset, expanding the supervised dataset for small original datasets for better learning outcomes (useful for Dataset 2) and shrinking down large datasets to increase training speed (useful for Dataset 1).

The resulting three-dimensional structure is shown in *Figure 9* and required for CNN and LSTM models.

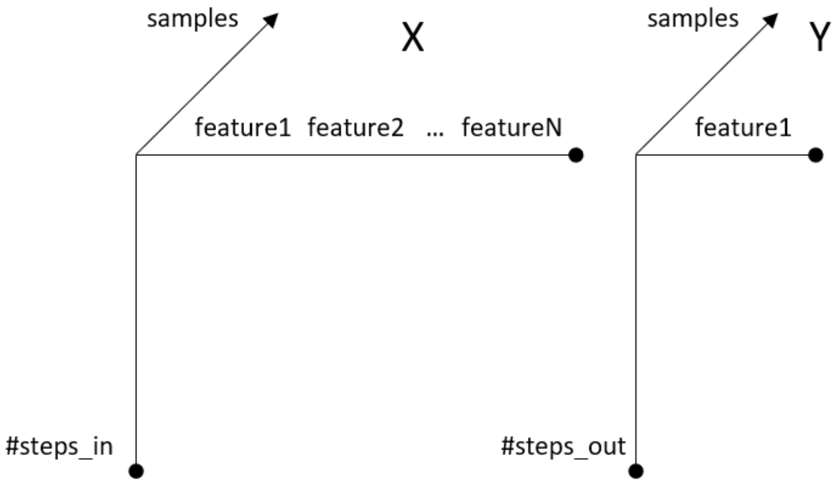

Figure 9: 3D shaped supervised dataset

Artificial neural networks require a two dimensional datastructure, each sample represented by one line in the matrix. The three-dimensional datastructure is folded *(Kourti 2003)* into a two dimensional datastructure, by keeping the sample-axis static and flattening the two-dimensional time-feature matrix (*Line 34*). The result is an array with the data of each feature subsequently aligned, shown in *Figure 10*

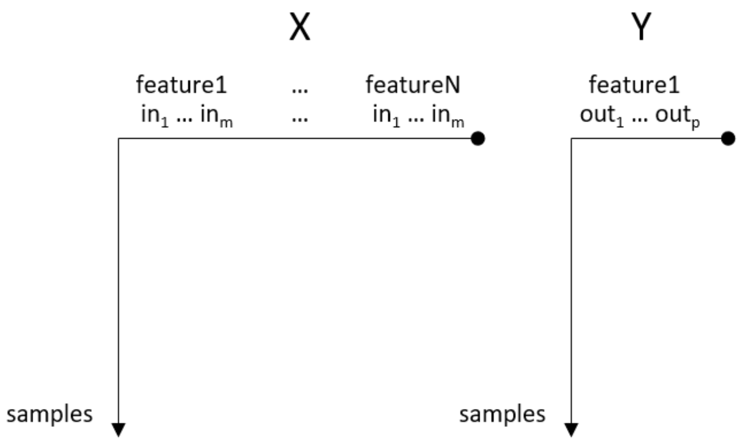

Figure 10: 2D shaped supervised dataset with $m$ as *#steps_in* and $p$ as *#steps_out*

### 3.3.3.3 Model Definition

The function *define_model* (*Listing 7*) is designed to easily support further models and to be included in automated hyperparameter optimization (*Section 3.3.3.4*). Currently, basic artificial neural networks (MLP in *Listing 7*), CNNs and LSTMs are supported. The number of neurons, number of hidden layers as well as kernel and pooling size for CNNs are variables for hyperparmeter optimization.

Listing 7: Deep Learning Model Definition according to *(Brownlee 2018c)*

```
def define_model(model, config, actv='relu'):

        input_shp, n_output, n_neurons, batch_size, hidden_layers, n_kernel,
            n_pool, shapeX, shapeY = config

        n_neurons = int(n_neurons)
        batch_size=int(batch_size)
        hidden_layers = int(hidden_layers)
        n_kernel = int(n_kernel)
        n_pool =int(n_pool)

        if(model == "MLP"):
                model = tf.keras.models.Sequential()
                model.add(tf.keras.layers.Dense(n_neurons, activation=actv,
                    input_shape=input_shp))
                for i in range(1,hidfden_layers):
                        model.add(tf.keras.layers.Dense(n_neurons, activation=
                            actv))
                model.add(tf.keras.layers.Dense(n_output[0]))
                model.compile(loss='mse', optimizer='adam')

        if(model == "CNN"):
                model = tf.keras.models.Sequential()
                model.add(tf.keras.layers.Conv1D(filters=n_neurons, kernel_size=
                    n_kernel, activation=actv, input_shape=input_shp))
                for i in range(1,hidden_layers):
                        model.add(tf.keras.layers.Conv1D(filters=n_neurons,
                            kernel_size=n_kernel, activation=actv))
                model.add(tf.keras.layers.MaxPooling1D(pool_size=n_pool))
                model.add(tf.keras.layers.Flatten())
                model.add(tf.keras.layers.Dense(n_output[0]))
                model.compile(loss='mse', optimizer='adam')

        if(model == "LSTM"):
                model = tf.keras.models.Sequential()
                if(stateful):
                        batch_input_sh = (batch_size,)+input_shp
                        model.add(tf.keras.layers.Lambda(lambda x:x,
                            batch_input_shape=batch_input_sh))
                else:
                        model.add(tf.keras.layers.Lambda(lambda x:x, input_shape
                            =input_shp))
                for i in range(1,hidden_layers):
                        model.add(tf.keras.layers.LSTM(n_neurons,
                            return_sequences=True, stateful=stateful))
                model.add(tf.keras.layers.LSTM(n_neurons, stateful=stateful))
                model.add(tf.keras.layers.Dense(n_output[0]))
                model.compile(loss='mse', optimizer='adam')

        return model
```

### 3.3.3.4 Hyperparameter Optimization

Hyperparameter optimization is the problem of optimizing a loss function, by searching the space of possible model and training process configurations *(J. S. Bergstra* *et al. 2011, J. Bergstra et al. 2015)* to improve the overall model performance. Hyperparameters can be discret, ordinal or continuous values *(J. S. Bergstra et al. 2011)*. Possible approaches for hyperparameter optimization are *(J. Bergstra et al. 2015)*

(a) Manual tuning

(b) Grid search

(c) Random search

(d) Bayesian search

Manual hyperparameter optimization relies on human experience and quickly becomes infeasible for high-dimensional search spaces. Grid-search sets fixed and discrete values to test each algorithm, fixing the parameters to a specific grid. In random search, parameters are selected randomly, within some pre-defined bounds. For high-dimensional hyperparemeter optimization problems, random search has proven advantageous over manual- or grid-search *(J. Bergstra et al. 2015)*. The Bayesian search strategy executes a few random search runs to obtain measurements of the multidimensional fitness surface of the loss function, before applying the Bayesian search gradient in gradient decent manner to approximate the model configuration with minimized loss function *(J. Bergstra et al. 2015)*.

This work uses *hyperopt*, a Bayesian search library for python introduced by *(J. Bergstra et al. 2015)*.

The tuned hyperparameters are *learning rate*, *number of neurons*, *activation function*, *number of hidden layers*, *input data window size in days ("days_in")*, *batch size* as well as *kernel size* and *pooling window size* for CNNs. The search space is defined according to *Listing 8*. The desired model algorithm was selected manually for each experiment.

Listing 8: Hyperopt Search-Space Definition according to *(J. Bergstra et al. 2013)*

```
space = {
        'learning_rate':hp.loguniform('learning_rate',np.log(0.005),np.log(0.9))
            ,           'model_algorithm':hp.choice('model_algorithm',
        [{'model_algorithm':'MLP','data_output_shape':'2D'},
        {'model_algorithm':'CNN','kernel_size':hp.quniform('kernel_size',2,10,1)
            , 'n_pool':hp.quniform('n_pool',1,5,1), 'data_output_shape':'3D'},
        {'model_algorithm':'LSTM','data_output_shape':'3D'}]),
        'neurons':hp.quniform('neurons',5,100,1),
                'actv':hp.choice('actv',['relu', 'tanh']),
        'hidden_layers':hp.quniform('hidden_layers',1,6,1),
        'days_in':hp.quniform('days_in',3,21,1),
        'batch_size':hp.quniform('batch_size', 20,200,1)

}
```

The *hyperopt* function *"fmin"* expects the space object defined in *Listing 8* and the loss function to be optimized. In this case, the loss function (objective function) defines the individual models to be trained and evaluated, as shown in *Listing 9*. *"EPOCHS"* is a global variable, defining the number of training epochs for one model. *"params"* is the configuration object for one specific experiment created by hyperopts' *"fmin"* based on the space object defined in *Listing 8*.

Listing 9: Hyperopt Objective-Function Definition

```
def objectiveFunction(params):
        epochs=EPOCHS
        batch_size=100
        kernel_size=0
        n_pool=0
        actv = params['actv']
        hidden_layers = params['hidden_layers']
        learning_rate=params['learning_rate']
        neurons=params['neurons']
        univariat = params['univariat_data']
        days_in = params['days_in']
        batch_size = int(params['batch_size'])

        algo_object = params['model_algorithm']
        model_algorithm = algo_object['model_algorithm']
        output_shape = algo_object['data_output_shape']

        if(model_algorithm == 'CNN'):
                kernel_size = algo_object['kernel_size']
                n_pool=algo_object['n_pool']


        Xtr, ytr, a, b = create_supervised_trainingset(training.copy(),
            output_shape=output_shape, scalingTransformer=scalingTransformer,
            n_steps_in=int(96*days_in), n_steps_out=96,  univariate=univariat,
            time_shift=global_timeshift)
        model = define_model(model_algorithm,[Xtr.shape[1:],ytr.shape[1:],
            neurons,epochs,batch_size,hidden_layers,kernel_size,n_pool, Xtr.
            shape, ytr.shape], actv=actv)
        doShuffle=True
        if(take_best_val_model):
                callbacks.append(TakeMinWeights())
        history = model.fit(Xtr,ytr,epochs=epochs,batch_size=batch_size,
            validation_split=0.2, shuffle=doShuffle, callbacks=callbacks,
            verbose=verbose))


        val_losses = history.history['val_loss']
        training_losses = history.history['loss']
        if(take_best_val_model):
                val_loss=min(val_losses)
        else:
                val_loss = val_losses[len(val_losses)-1]
        of_connection = open(out_file, 'a')
        writer = csv.writer(of_connection)
        time_diff = int(time.time())-gobaltime
        #All results of a hyperopt run are collected in a csv for rapid
            evaluation
        writer.writerow([time_diff, val_loss, params, val_losses,training_losses
            , time.time()])
        #Independent of the model performance, each model is saved for later
            evaluation using the test dataset.
        model_path = trainpath+'/models/hyperopt_search_%s_model_%s'%(
            experiment_nr, time_diff)
        of_connection.close()
        model.save(model_path)

        return {'loss': val_loss, 'params': params, 'history':history.history, '
            status':STATUS_OK, 'model':model, 'train_loss':training_losses}
```

## 3.4 Performance Evaluation

### 3.4.1 Performance Metrics

Performance metrics are vital components for the evaluation of regression models *(Botchkarev 2019)*. This work utilizes four performance metrics, the mean square error, the root mean square error, the mean absolute error, and the cumulative absolute error to evaluate the forecasting accuracy. These metrics allow comparison between models and with the baseline.

Mean square error (MSE) is the mean value of the sum of squared differences, calculated as follows

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{20}$$

The mean absolute error (MAE) is the mean value of the sum of absolute differences *(Bouktif et al. 2018)*, calculated as follows

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i| \tag{21}$$

The root mean square error (RMSE) is the root of the mean value of the sum of squared differences (MSE), calculated as follows *(Bouktif et al. 2018)*

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2} \tag{22}$$

The RMSE is large for large errors and penalizes outliers more severely than MAE when used during model training *(Bouktif et al. 2018)*. The cumulative absolute error (CAE) is the sum of absolute differences and is used to assess the absolute improvements. The CAE is calculated according to *Equation 23*.

$$CAE = \sum_{i=1}^{N}|y_i - \hat{y}_i| \tag{23}$$

For all equations listed above, $y_i$ is the observed value, $\hat{y}_i$ the predicted value and $N$ the total number of observations.

### 3.4.2 Training and Validation Loss Plots

Training and validation loss plots give a good indication, whether a training process is going to deliver promising results, or not. The training loss value for a specific epoch is the mean loss of each batch processed within. The validation loss value of a specific epoch is the error of the model applied to the validation set. Plotting training- and validation loss for each epoch of a models training can give valuable insights of training performance. *Figure 11* shows an ideal epoch-loss plot with the training loss converging towards zero and the validation loss following closely. In *Figure 12*, the validation loss starts diverging from the training loss at epoch 60. This indicates that the model is overfitting, a common problem of deep learning algorithms. When overfitting occurs, the algorithm learns the trainings data too closely, sacrificing generalization abilities. According to Brownlee *(Brownlee 2018b)* possible countermeasures are

(a) **Weight Regularization**
An indicator for overfitting can be large network weights. With large network weights, a small change in network input might result in a large change in the network output, reducing the networks reliability. Weight regularization encourages small weights during network training. *(Brownlee 2018g)*

(b) **Activity Regularization**
Activity regularization encourages neural networks to learn sparse features or internal representations of raw observations. *(Brownlee 2018e)*

(c) **Dropout**
Dropout is a computationally cheap way to regularize a deep neural network. It works by probabilistically removing inputs to a layer, which has the effect of simulating many networks with a different network structure each. *(Brownlee 2018f)*

(d) **Noise**
Noise has shown to have a regularizing effect on under-constrained neural network models with a small training dataset. *(Brownlee 2018d)*

(e) **Early Stopping**
Choosing the training duration is a major challenge in training neural network models. Too little training will result in underfitted training- and test sets. Too much training will result in an overfitted model on the training set and poor performance on the test set. Early stopping aborts the training process and saves the model on a given condition before the model starts overfitting the training data, improving generalization. *(Brownlee 2018a)*

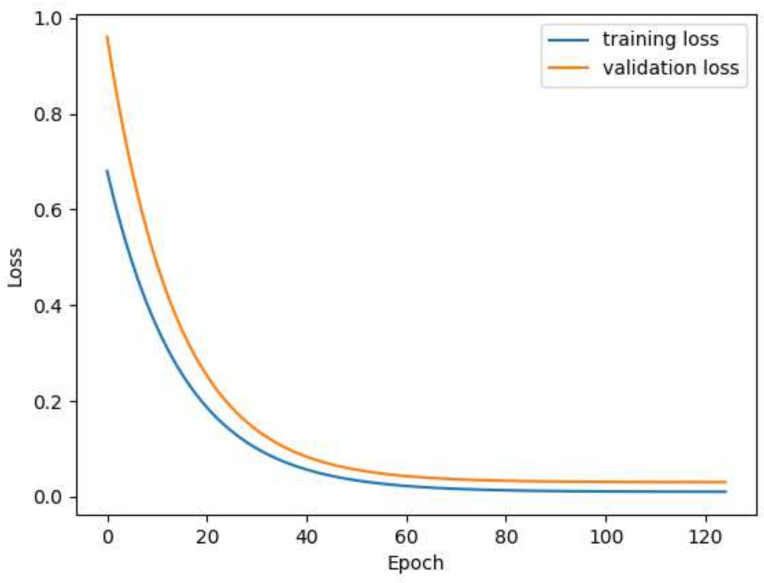

Figure 11: Ideal training and validation loss progression

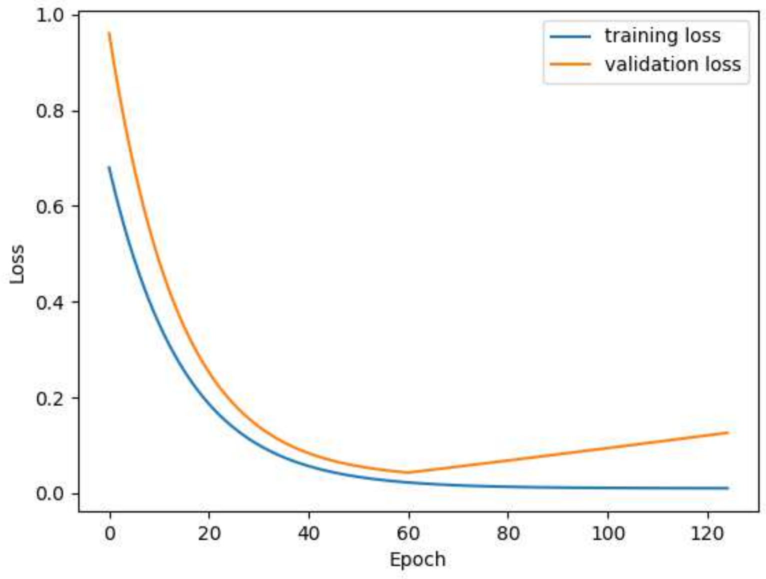

Figure 12: Training and validation loss progression with overfitting

In this thesis, a combination of dropout, weight constraint and a special form of early stopping (“take_best” approach) have been applied.

By utilizing *hyper_opt* hyperparameter optimization, a multitude of models is trained in one batch, with the same number of epochs each. To select the best model before overfitting starts, the *'take_best'* methodology is applied as callback to the model training process (*Listing 9, Line 27*). In this methodology, the best performing weights on the validation set (*val_loss*) are stored (*Listing 10, Line 19*) and reloaded into the model after training has finished (*Listing 10, Line 23*). This ensures that the best model of a training is returned and stored as final model.

Listing 10: Callback Function "TakeMinWeights" inspored by *(Core 2020)*

```
1  #Callback class, provided to TensorFlow
2  class TakeMinWeights(tf.keras.callbacks.Callback):
3          def __init__(self):
4                  super(TakeMinWeights, self).__init__()
5  
6          #Initialize best_weights with none.
7          self.best_weights = None
8  
9          def on_train_begin(self, logs=None):
10                 # Initialize the best as infinity.
11                 self.best = np.Inf
12 
13         def on_epoch_end(self, epoch, logs=None):
14                 current = logs.get('val_loss')
15                 #Check if current validation loss is better, than the best
16                 if np.less(current, self.best):
17                         #If yes, replace.
18                         self.best = current
19                         self.best_weights = self.model.get_weights()
20 
21         def on_train_end(self, logs=None):
22                 #Load best weights into model, before saving.
23                 self.model.set_weights(self.best_weights)
```

### 3.4.3 Result Evaluation

The obtained prediction is transformed back to the original space. In production settings, the prediction in original space would be used for demand planning.

To compare the results across datasets, standardisation (*Section 2.4.2*) is applied to the predicted data. *Listing 11* shows the function *scale_backY* used to transform the prediction from normalized to original space (*Line 1*) and the function *evalStandardScaling* used to standardize the predicted data (*Line 7*).

Listing 11: Transformation of Predicted Data inspired by *(stackoverflow 2020(a))*

```
1  def scale_backY(yData, scalingTransformer, nrColumns):
2          y_reshaped = np.zeros((yData.shape[0]*yData.shape[1],nrColumns))
3          y_reshaped[:,0] = np.reshape(yData,(yData.shape[0]*yData.shape[1]))
4          scaled_back = scalingTransformer.inverse_transform(y_reshaped)
5          return scaled_back[:,0]
6  
7  def evalStandardScaling(prediction, ground_truth):
8          #Standard Scaler
9          df=pd.DataFrame(ground_truth)
10         df= df.append(pd.DataFrame(prediction))
11         df.reset_index()
12         robust_transformer = RobustScaler().fit(df)
13         pred_rb = robust_transformer.transform(pd.DataFrame(prediction))
14         gt_rb = robust_transformer.transform(pd.DataFrame(ground_truth))
15         return pred_rb, gt_rb
```

Standardized predicted data is used for all plots and performance metrics of *Section 4*.

# 4 Experiments and Results

## 4.1 Infrastructure and Software

The data preprocessing and baseline experiments were run on a Lenovo ThinkPad Yoga 370 with an Intel Core i7 2.9 GHz CPU and 16GB RAM running Windows 10. Data preprocessing and baseline experiments were conducted with Python 3.7.3, and Spyder 3.3.3 was used as IDE. Additionally the python packages pandas, matplotlib, datetime, sklearn, calendar, holiday, hyperopt and numpy have been used.

Machine learning experiments were executed on Google Cloud Compute Engine using the provided Deep Learning VM Image in region europe-west1-d with TensorFlow 2.1 environment on a machine with 2 CPUs, 13 GB RAM and a NVIDIA Tesla K80 x1 GPU. Code was deployed using pre-configured Juypter Lab.

## 4.2 Introduction and Exploration of Datasets

In this thesis three real-world industrial electricity consumption datasets have been analysed, introduced in *Table 1*.

Table 1: Introduction of Datasets

| | | |
|---|---|---|
| Company A | Dataset 1 | The provided dataset has been recorded at a plant producing medical packaging. |
| Company B | Dataset 2 | The facility in which this data set was recorded produces packaging for food & beverages, fast-moving-consumer-goods and hygiene products. |
| Company C | Dataset 3 | This dataset was provided by an electric utility company aiming to improve their in-house forecasting. It was recorded at an industrial plant, however the goods produced are unknown. |

### 4.2.1 Dataset Exploration Company A

Company A is a plastic packaging manufacturer and the given plant is producing pharmaceutical packaging. The dataset holds five years of electric power consumption data, between January 2013 and December 2019. *Figure 13* depicts the electric power consumption for 2018, showing major load peaks in October and December. It is known that this company is experiencing load peaks frequently. For this work, the load peaks will be considered as outliers and are removed according to *Section 3.1.2*. *Figure 14* shows the electric power consumption of Company A for 2018 with outliers removed. Some peaks still remained, but within reasonable bounds. *Figure 14* indicates a cyclic yearly pattern with a increased energy consumption in Summer, which is supported by *Figure 15*, showing monthly box plots for 2018 and 2019. The box plot of *Figure 15* suggests a seasonal pattern.

*Figure 16* shows the power consumption in the range February 1st 2018 until February 14th 2018, with February 3rd and February 10th being Saturdays and February

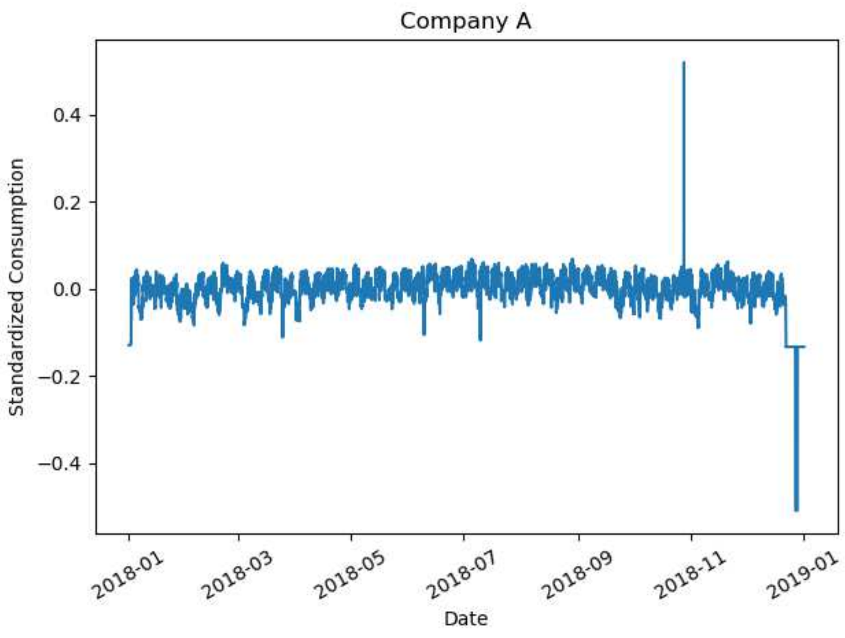

Figure 13: Electric power consumption of Company A in 2018

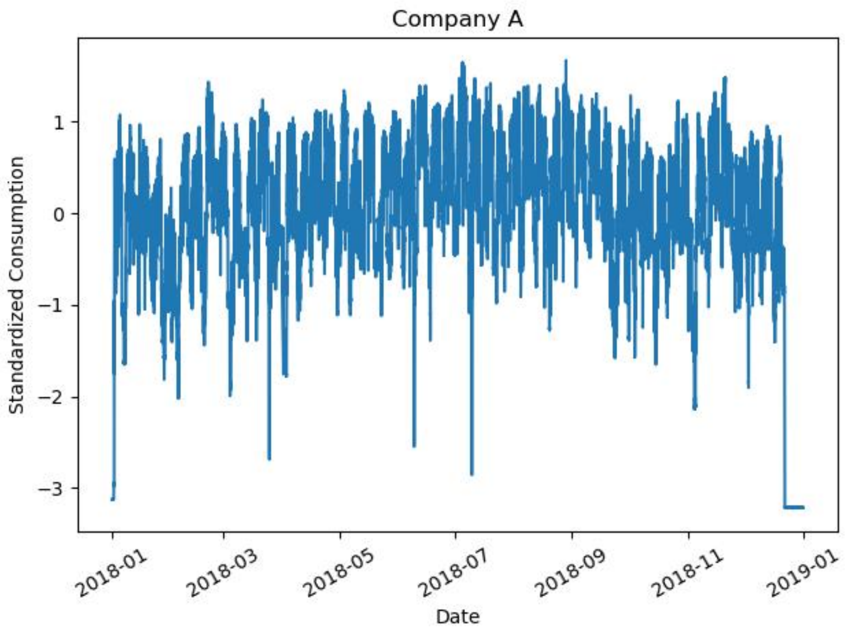

Figure 14: Electric power consumption of Company A in 2018 with outliers removed

4th and February 11th being Sundays. A drop in power consumptions can clearly be seen for weekends, however, *Figure 16* does not indicate any further weekly patterns.

*Figure 17* shows the power consumption of two subsequent days, Tuesday February 6th 2018 00:00 until Wednesday February 8th 2018 23:59, not indicating a predominant inter-day pattern, suggesting a 4-shift-work cycle being run by Company A.

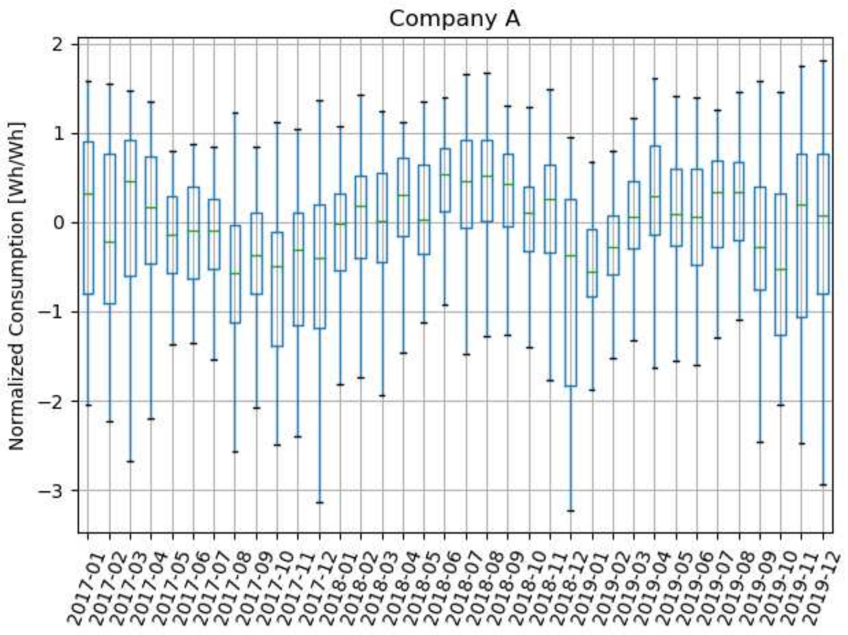

Figure 15: Electric power consumption box plot of Company A in 2017, 2018 and 2019

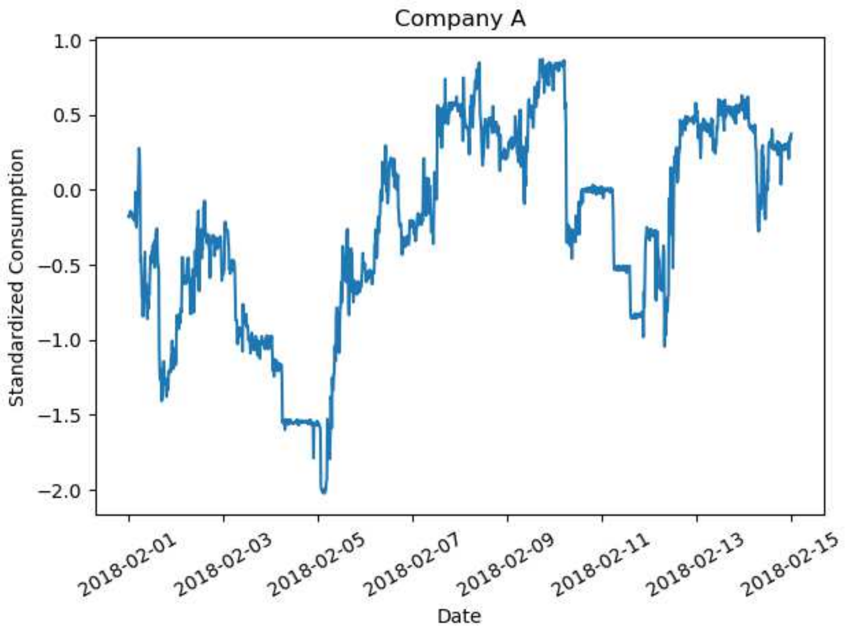

Figure 16: Electric power consumption of Company A in the range February 1st 2018 until February 14th 2018

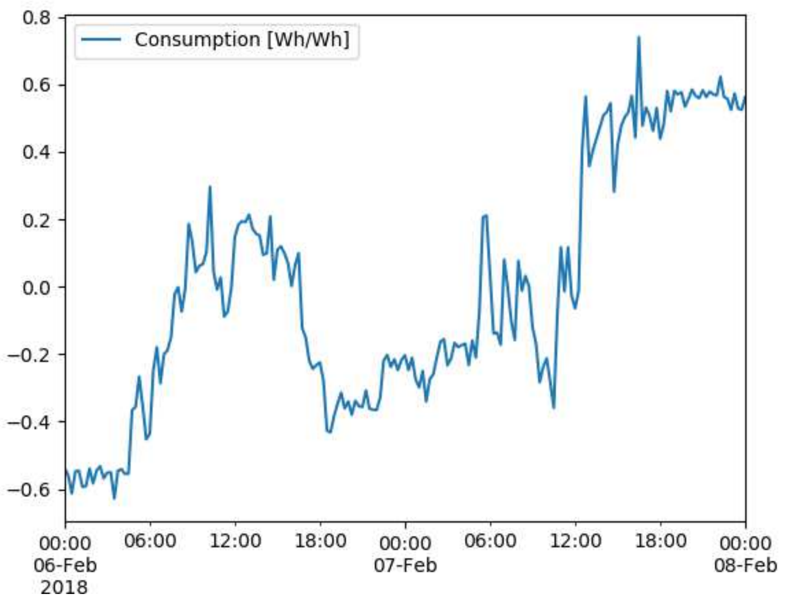

Figure 17: Electric power consumption of Company A in the range February 6th 2018 00:00 until February 8th 2018 23:59

### 4.2.2 Dataset Exploration Company B

Company B is a paper goods manufacturer and this specific plant is producing packaging for food & drinks, fast-moving-consumer-goods and hygiene products. The dataset provides one year and nine month of electric power consumption data, between January 2018 and September 2019. *Figure 18* depicts the electric power consumption for 2018, not showing any significant outliers nor indicating any seasonal trend. This is supported by *Figure 19* showing the box plot for 2018 and 2019. The median drop in *2018-12* of *Figure 19* is due to a two week production stop during the Christmas period.

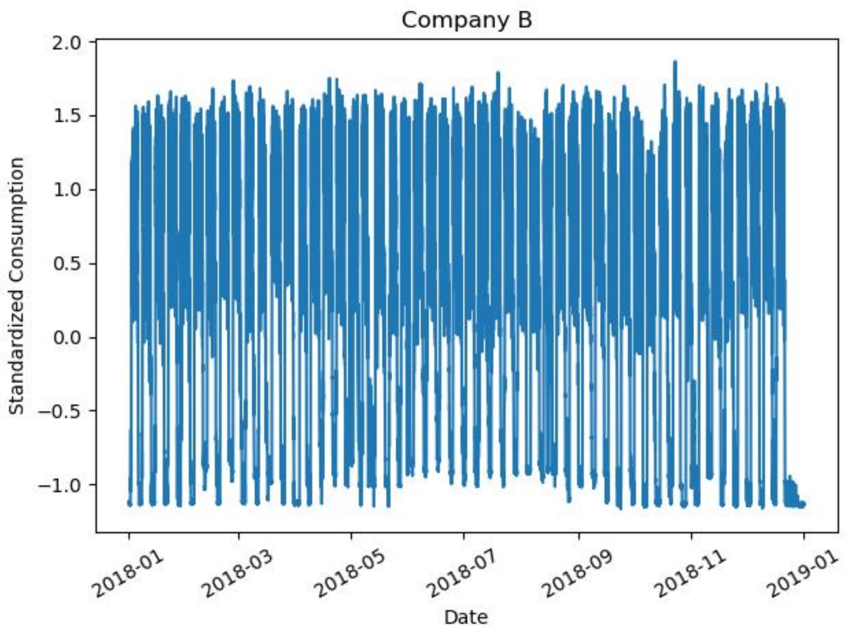

Figure 18: Electric power consumption of Company B in 2018

*Figure 20* shows the power consumption in the range February 1st 2018 until February 14th 2018, with February 3rd and February 10th being Saturdays and February 4th and February 11th being Sundays. A drop in power consumptions can clearly be seen for weekends. The end of each workday is clearly identifiable by an overnight drop in power consumption.

*Figure 21* shows the power consumption of two subsequent days, Tuesday February 6th 2018 00:00 until Wednesday February 8th 2018 23:59 with the power consumption starting at 5:00 and dropping again at 21:00, representing a total 16 hours of active work, thus suggesting a non-overlapping two shift work cycle.

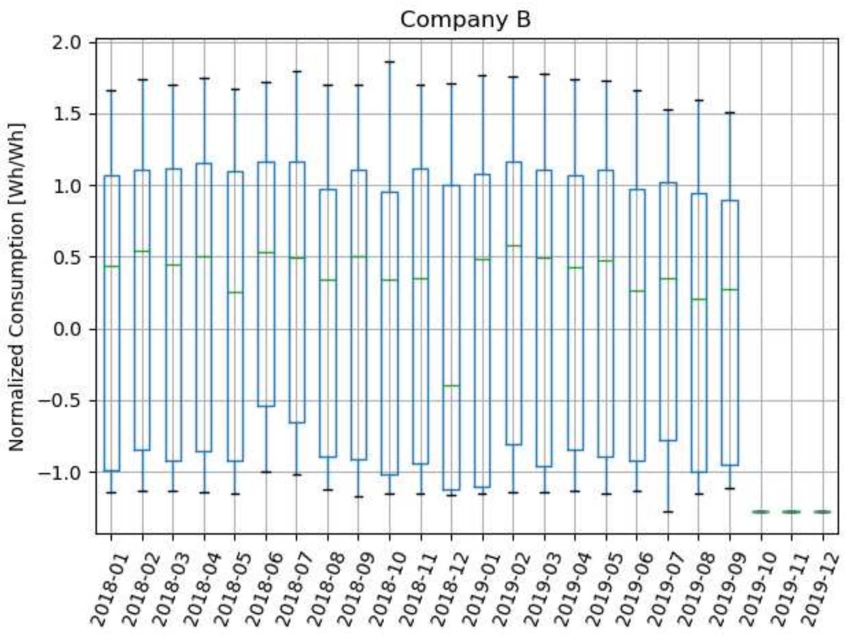

Figure 19: Electric power consumption box plot of Company B in 2018 and 2019

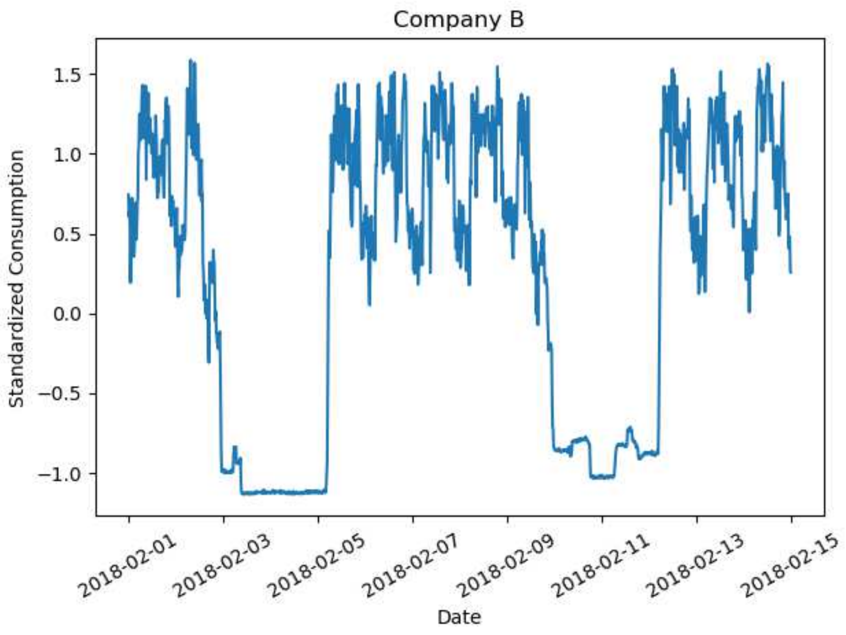

Figure 20: Electric power consumption of Company B in the range February 1st 2018 until February 14th 2018

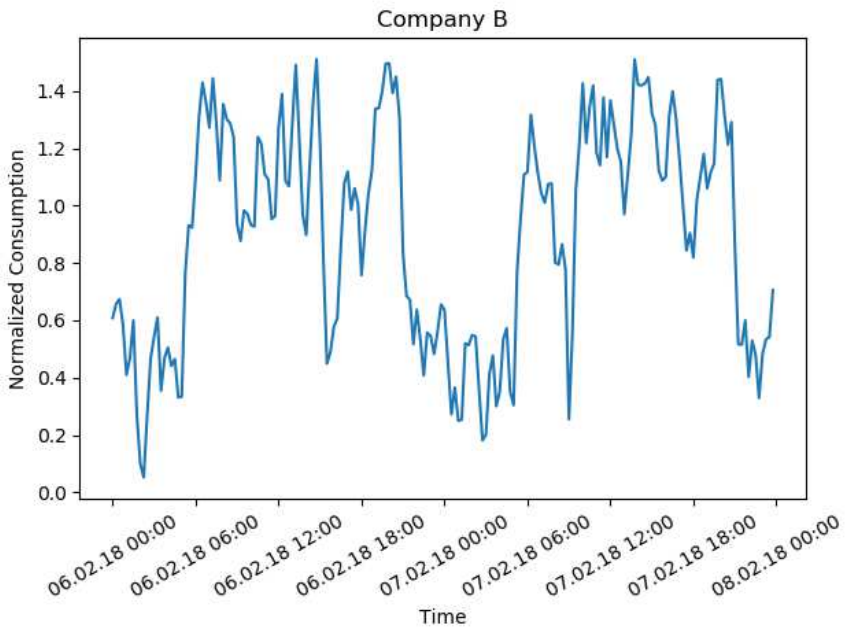

Figure 21: Electric power consumption of Company B in the range February 6th 2018 00:00 until February 8th 2018 23:59

### 4.2.3 Dataset Exploration Company C

The dataset of Company C has been provided by a utility company and its field of operation is unknown. The dataset holds five years of electric power consumption data, between November 2015 and February 2020. *Figure 22* depicts the electric power consumption for 2018, indicating two weeks of work interruptions in August 2018 and three weeks in October 2018, not showing any significant load peaks. The box plot shown in *Figure 23* suggests a minor seasonality, with electricity consumption having its yearly minimum between July and August.

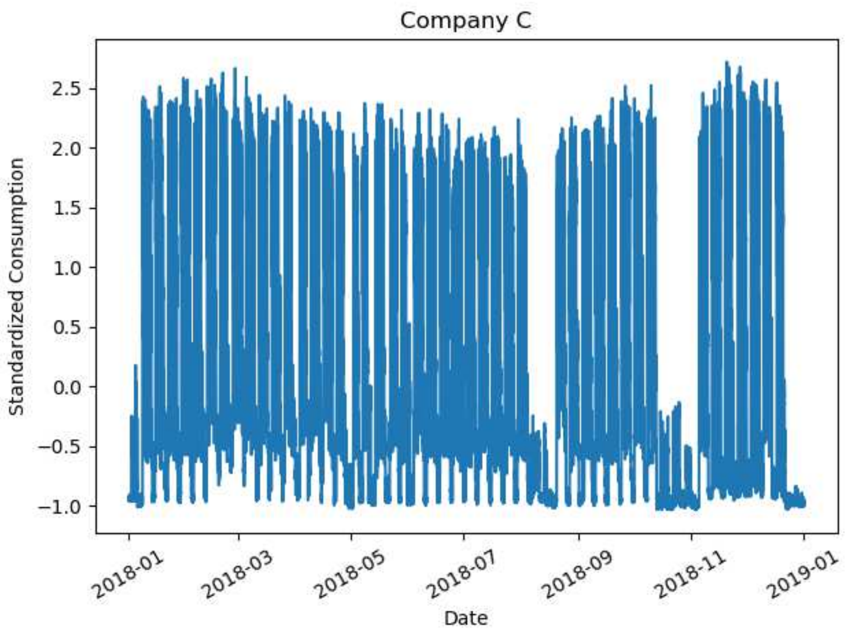

Figure 22: Electric power consumption of Company C in 2018

*Figure 24* shows the power consumption in the range February 1st 2018 until February 14th 2018, with February 3rd and February 10th being Saturdays and February 4th and February 11th being Sundays. A drop in power consumptions can clearly be seen for weekends. The end of each workday is clearly identifiable by an overnight drop in power consumption. It is important to point out, that some Fridays also show a significant drop in power consumption compared to the other workdays, suggesting a reduced workload.

*Figure 25* shows the power consumption of two subsequent days, Tuesday February 6th 2018 00:00 until Wednesday February 8th 2018 23:59 with the power consumption starting at 6:30 and dropping again at 16:30, representing a total 10 hours of active work, thus suggesting a overlapping two shift work cycle.

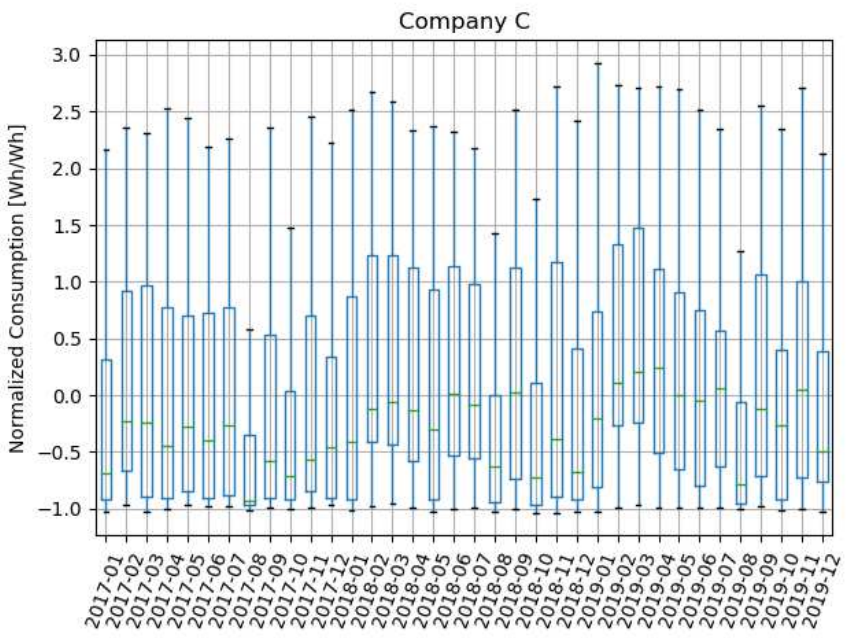

Figure 23: Electric power consumption box plot of Company B in 2017, 2018 and 2019

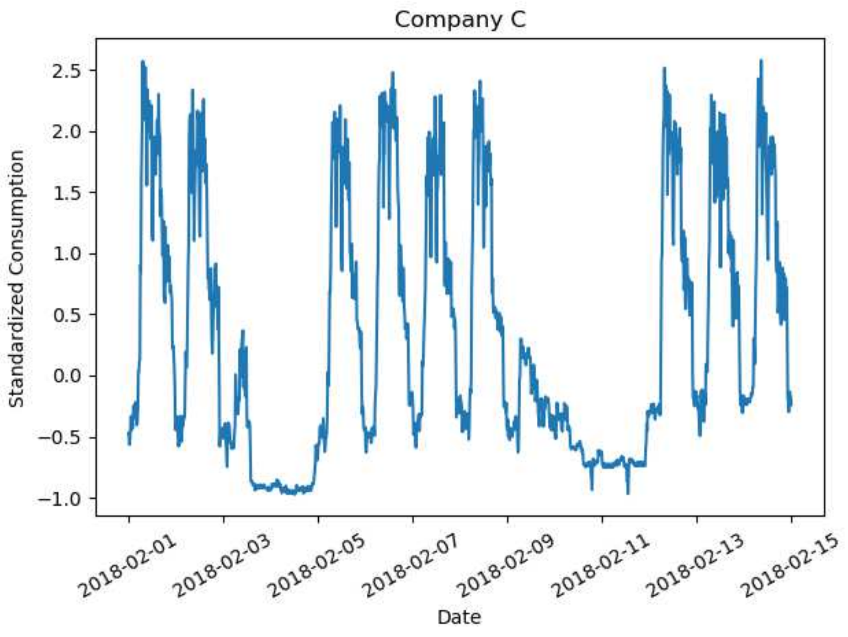

Figure 24: Electric power consumption of Company C in the range February 1st 2018 until February 14th 2018

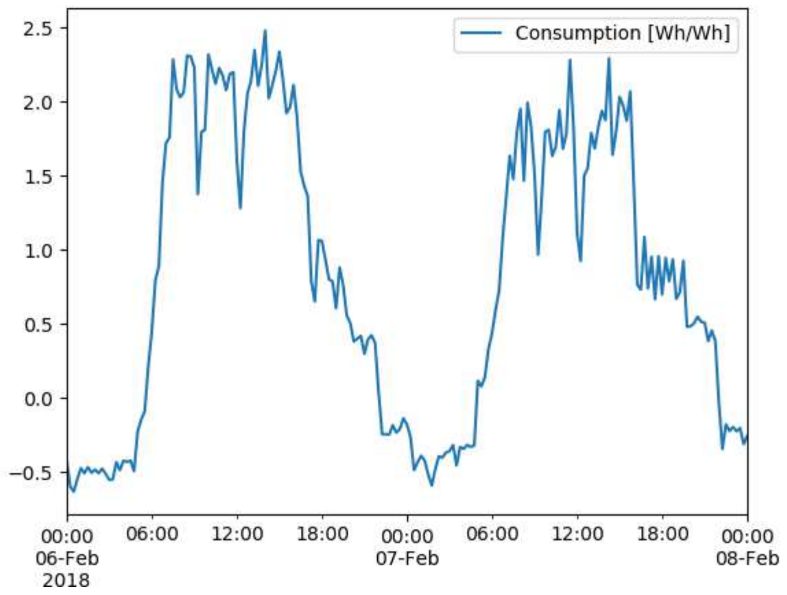

Figure 25: Electric power consumption of Company C in the range February 6th 2018 00:00 until February 8th 2018 23:59

## 4.3 Demand Prediction

In this Section, the prediction results for the baseline algorithm, neural network models, convolution neural network models and long-short term memory models are discussed for each dataset independently. The overall results across datasets are discussed in *Section 5*

To assess the prediction results across models, the performance metrics introduced in *Section 3.4* are used and the predicted and actual ("ground truth") standardized electricity consumption is plotted for the test set.

Additionally, four comparison time ranges have been selected, aiming to obtain a zoomed-in view on time periods of interest:

(a) **Zoom 1** (1.3.2019 - 16.3.2019)
was selected to represent two normal weeks, without holidays

(b) **Zoom 2** (13.5.2019 - 26.5.2019)
was selected because Monday, March 21st 2019 was Whit Monday, a public holiday, and is therefore directly impacting the baseline predictions

(c) **Zoom 3** (17.6.2019 - 1.7.2019)
includes Corpus Christi 2019 (Thursday June 20th 2019), a public holiday

(d) **Zoom 4** (29.7.2019 - 25.8.2019)
includes Assumption Day 2019 (August 15th 2019) and company holidays or maintenance shutdowns for Company C

The performance metrics presented in the following sections have been calculated using standardized data *(Section 2.4.2)* and are without unit.

### 4.3.1 Processing *Dataset 1*

Following, Dataset 1 is predicted using the Baseline and the best performing neural network model, convolution neural network model and long short-term memory model.

#### 4.3.1.1 Baseline

*Figure 26* shows the baseline prediction, as introduced in *Section 3.2*, for Dataset 1. The overall, seasonal pattern of 2018 is matching the seasonal pattern of 2019, but the predictions show significant divergence with an overall RMSE of $0.798$ and absolute error of $20115$, suggesting improvement potential. The error statistics listed in *Table 2* have been calculated using standardized energy demand and therefore do not have a unit. The drop in electricity consumption at the end of 2019 is explained by Christmas and New-Year holidays. *Table 2* lists the performance metrics for the baseline of Dataset 1.

*Figure 27* depicts Zoom 1 of the baseline prediction for Dataset 1, suggesting major divergences between ground truth and prediction. This is confirmed by a high RMSE value, compared to the other zoom segments.

*Figure 28* is depicting Zoom 2, with the algorithms holiday compensation of Whit Monday 2020 clearly recognisable from May 20th to May 21st. For Zoom 2, the

Table 2: Dataset 1 Baseline Error Statistics

| | MAE [Wh/Wh] | MSE [Wh/Wh] | RMSE [Wh/Wh] | CAE [Wh/Wh] |
|---|---|---|---|---|
| Full | 0.574 | 0.637 | 0.798 | 20115 |
| Zoom 1 | 0.524 | 0.411 | 0.641 | 754 |
| Zoom 2 | 0.327 | 0.169 | 0.411 | 408 |
| Zoom 3 | 0.544 | 0.438 | 0.662 | 731 |
| Zoom 4 | 0.339 | 0.177 | 0.420 | 879 |

historic values used for the baseline prediction, show a better resemblance with the ground truth than in Zoom 1, confirmed by a lower RMSE value of $0.169$.

*Figure 29* shows the baseline prediction for Zoom 3. The holiday correction to the mean off-days is clearly identifiable. For Zoom 3, predicted consumption is higher than actual consumption levels resulting in a higher RMSE value compared to Zoom 2 and a similar error compared to Zoom 1.

*Figure 30* is showing the baseline prediction for Zoom 4. The prediction approximates the ground truth comparatively well, resulting in a low RMSE value of $0.177$. The holiday compensations introduced by the baseline for 15.8.2018 (Assumption Day 2018) and 16.8.2019 (Assumption Day 2019) are clearly visible.

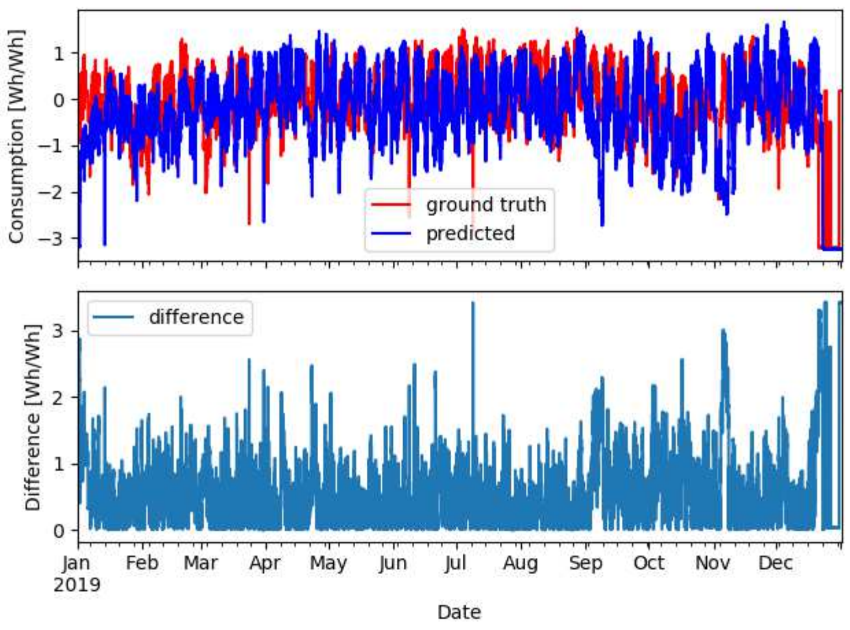

Figure 26: Baseline Prediction for Dataset 1

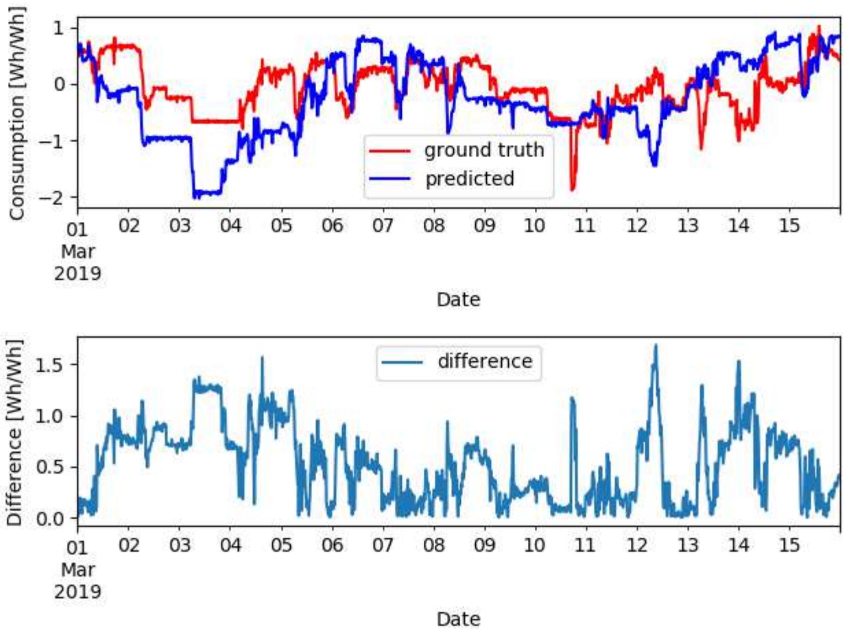

Figure 27: Baseline Prediction for Dataset 1, Zoom 1 (01.03.2019 until 16.03.2019)

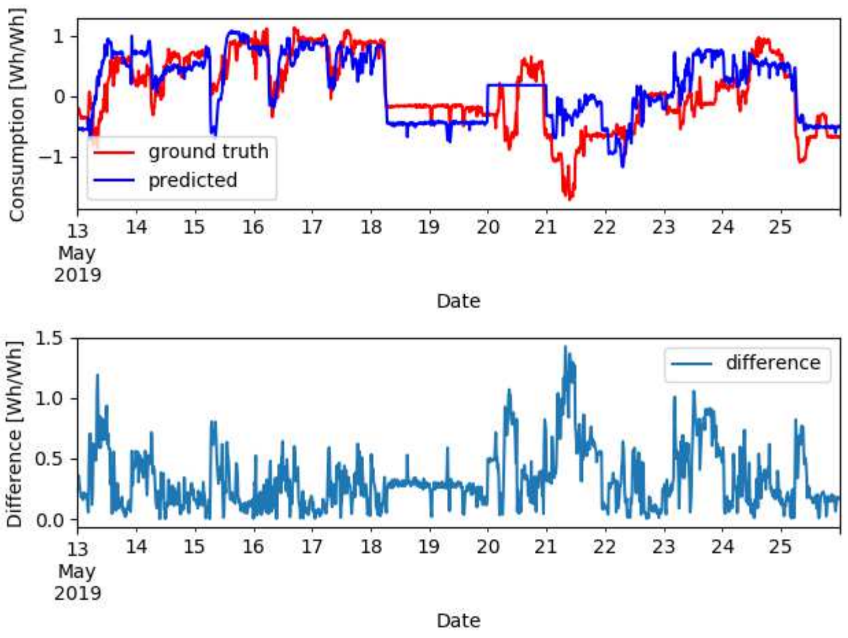

Figure 28: Baseline Prediction for Dataset 1, Zoom 2 (Whit Monday 2018)

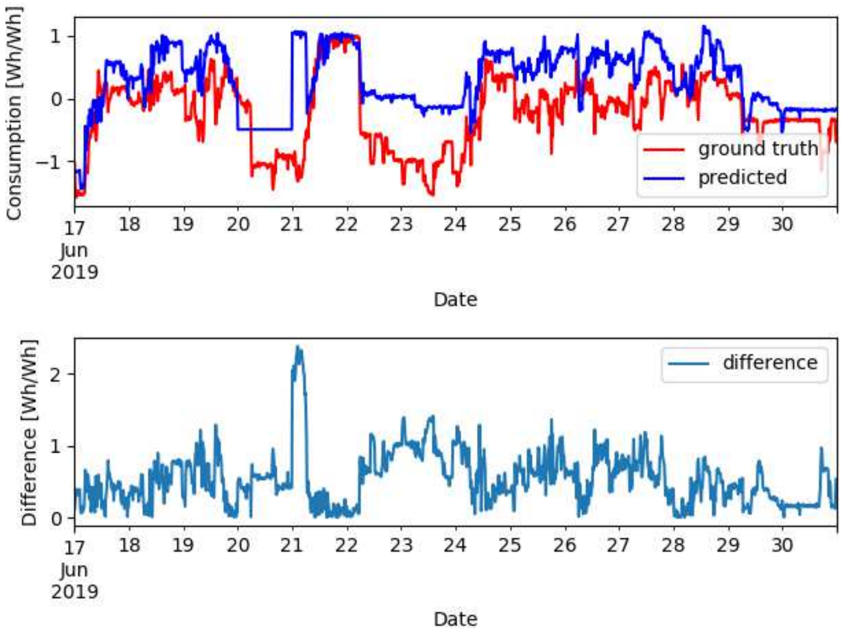

Figure 29: Baseline Prediction for Dataset 1, Zoom 3 (Corpus Christi 2019)

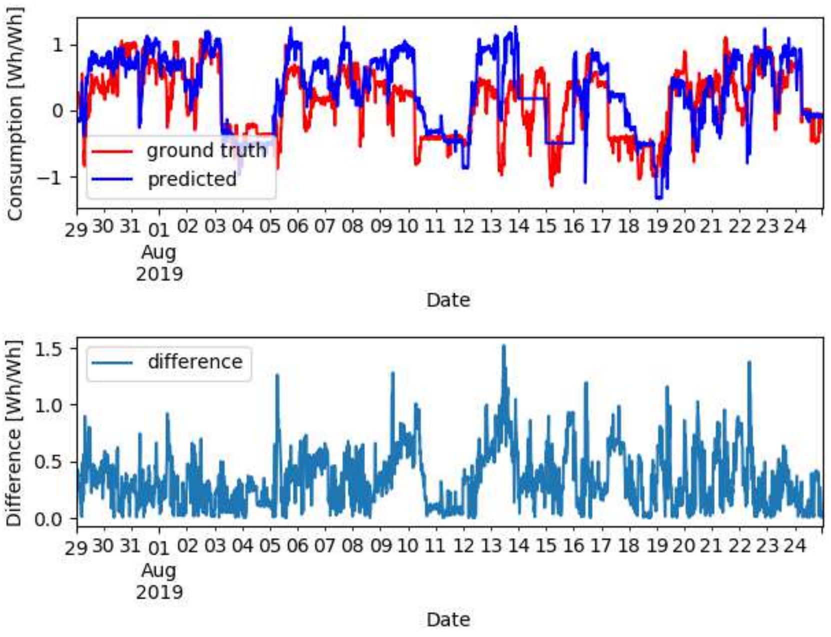

Figure 30: Baseline Prediction for Dataset 1, Zoom 4 (29.07.2019 until 25.08.2019)

### 4.3.1.2 Artificial Neural Network

Table 3: Dataset 1 ANN Error Statistics

| | MAE [Wh/Wh] | MSE [Wh/Wh] | RMSE [Wh/Wh] | CAE [Wh/Wh] | RMSE [%] | CAE [%] |
|---|---|---|---|---|---|---|
| Full | 0.33 | 0.214 | 0.462 | 11310 | 42 | 44 |
| Zoom 1 | 0.256 | 0.113 | 0.336 | 369 | 48 | 51 |
| Zoom 2 | 0.303 | 0.189 | 0.435 | 378 | -6 | 7 |
| Zoom 3 | 0.342 | 0.19 | 0.435 | 460 | 34 | 37 |
| Zoom 4 | 0.266 | 0.112 | 0.335 | 689 | 20 | 22 |

A total of 41 ANN models have been tested on Dataset 1, with an overall mean RMSE of $0.559$ and a mean absolute error of $13947$.

The selected model showed the best prediction performance of ANN models trained on Dataset 1. It used 7 days as input and had 3 hidden layers, with 58 neurons each and deployed relu as activation function.

The prediction for 2019, shown in *Figure 31*, resembles the ground truth better than baseline, especially from July 2019. However, until July 2019, the model predicts many drops in consumption not occurring in the baseline.

Overall the model achieves an RMSE of $0.462$ and an absolute error of $11310$, providing improvements of $42\%$ and $44\%$ respectively.

*Figure 32* shows the ANN prediction of Zoom 1. Until March 11th the ground truth presents a weeky pattern, with consumption drops in the early morning hours and at weekends (weekends: March 2nd to March 3rd and March 8th to March 10th). The ANN model learned to approximate the weekly pattern, but starts to diverge from March 11th. The ANN model achieved an RMSE value of $0.336$ and a absolute error of $369$ compared to baseline. This is an improvement of $48\%$ and $51\%$, respectively.

*Figure 33* depicts prediction and ground truth for Zoom 2, with the model accurately predicting the daily and weekly consumption patterns. Whit Monday 2018 does not impact the prediction in Zoom 2 (19.5.2019 - 20.5.2019). However, a consumption drop at night between March 20th and March 21st results in prediction inaccuracies, persisting until the end of March 21st. With the beginning of March 22nd, the prediction is again a good estimate for the ground truth. The reason for the inaccuracy lasting a bit more than 24 hours is to be found in the assumption, that, from midnight of a given day, the next 24 hours have to be predicted., limiting the algorithms capabilities to adapt to changes.

For Zoom 2 the ANN model achieves an RMSE value of $0.435$ and a cumulative absolute error of $378$, performing by $6\%$ worse on the RMSE than baseline, but $7\%$ better on CAE.

In Zoom 3, shown in *Figure 34*, the model roughly approximates the ground truth, but the intra-day consumption characteristics are different compared to Zoom 2, worsening the prediction performance. The ANN model accounts for Corpus Christi 2019 by reducing the predicted consumption level, but is too conservative in its prediction for the following bridging day. The ANN model achieves an RMSE value of

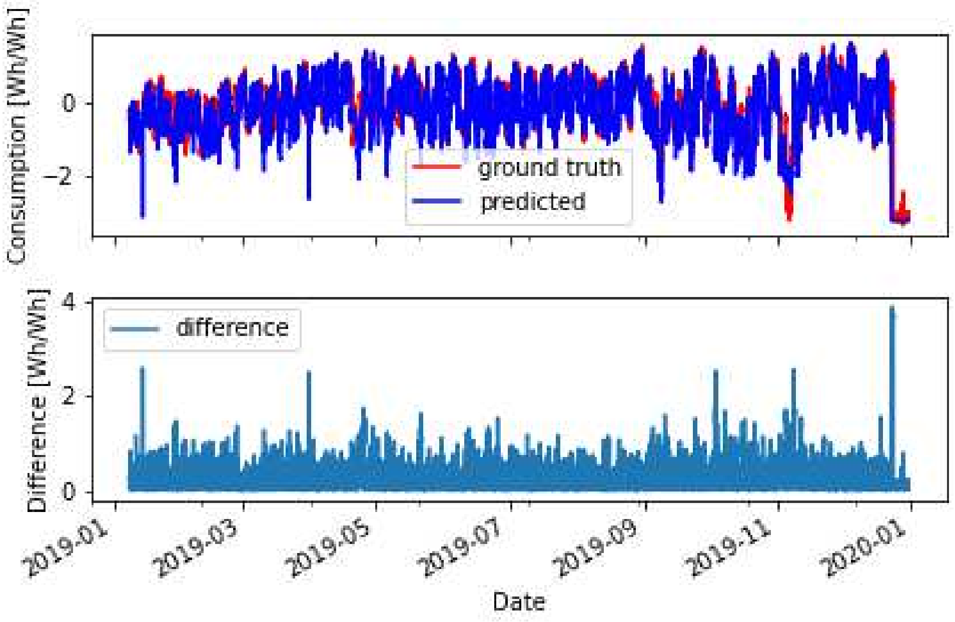

Figure 31: ANN Prediction for Dataset 1

$0.435$ and an cumulative absolute error of $460$, representing an improvement of $34\%$ and $37\%$ compared to the baseline.

In Zoom 4, the ANN model follows the weekly consumption pattern of the ground truth and accounts for Assumption Day 2019 while bouncing back comparatively slow the day after. The ANN model achieves an RMSE value of $0.335$ and an cumulative absolute error of $689$, representing an improvement of $20\%$ and $22\%$, respectively.

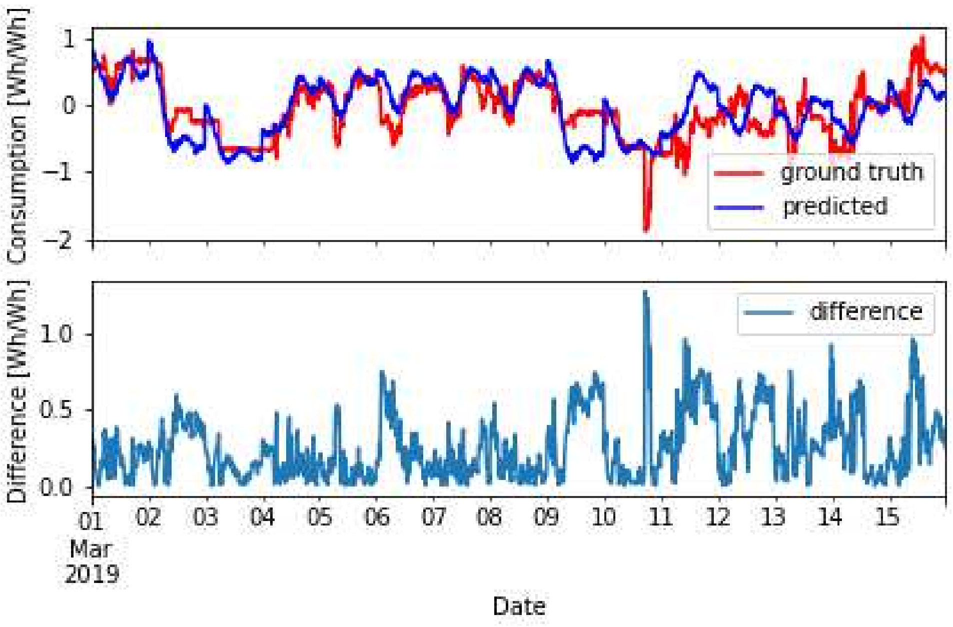

Figure 32: ANN Prediction for Dataset 1, Zoom 1 (01.03.2019 until 16.03.2019)

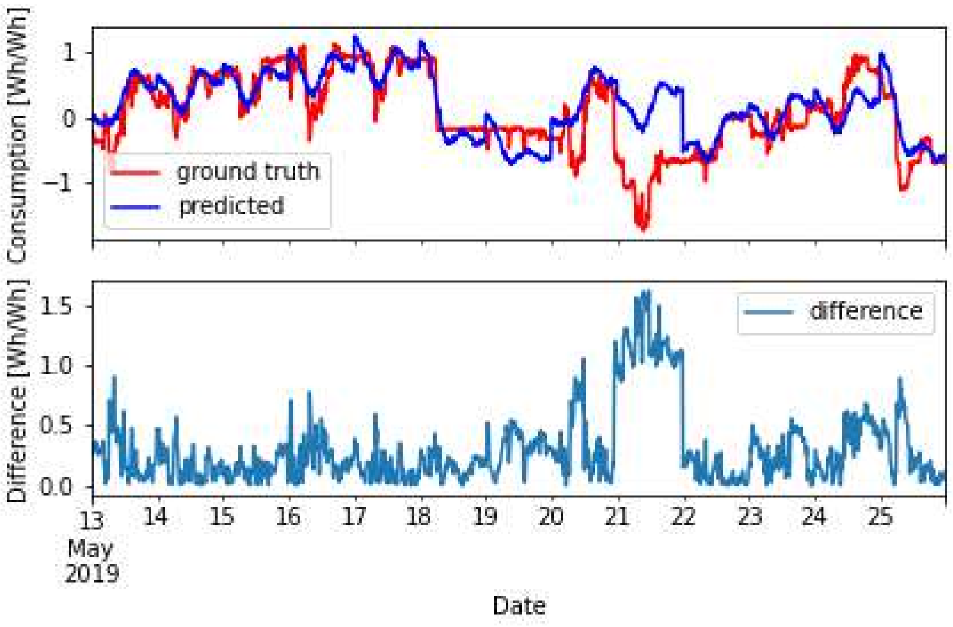

Figure 33: ANN Prediction for Dataset 1, Zoom 2 (Whit Monday 2018)

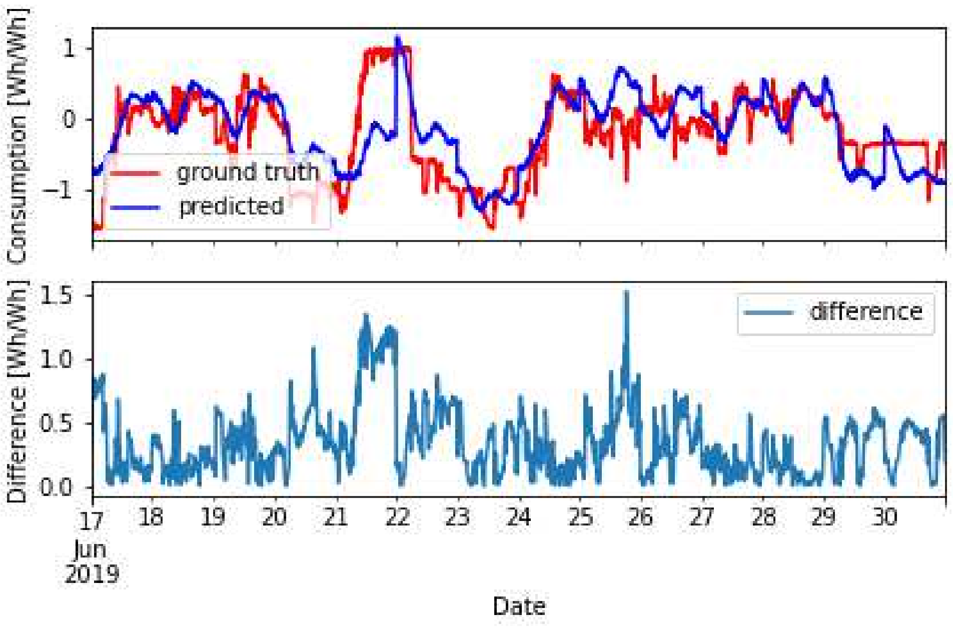

Figure 34: ANN Prediction for Dataset 1, Zoom 3 (Corpus Christi 2019)

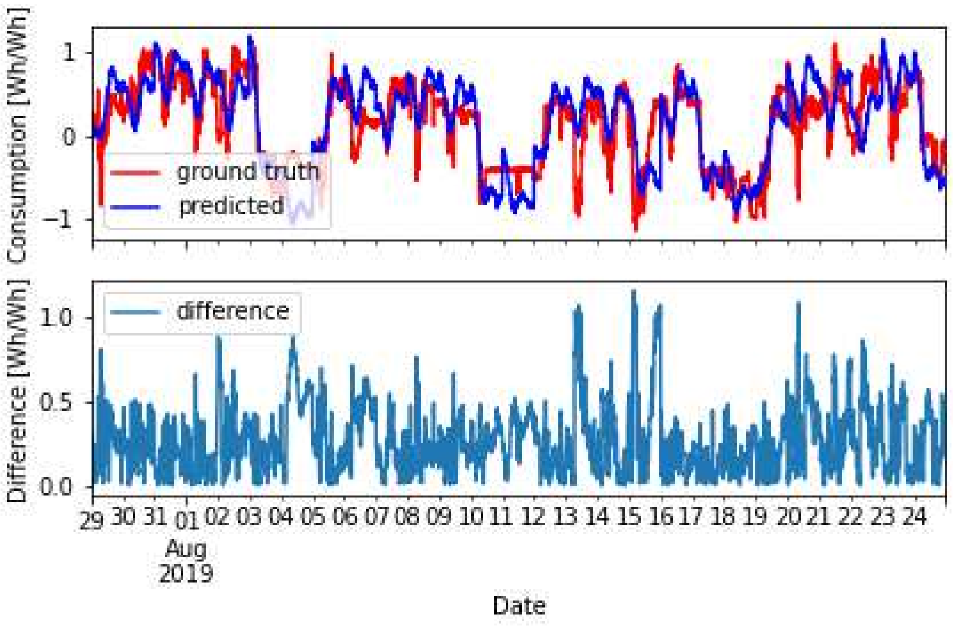

Figure 35: ANN Prediction for Dataset 1, Zoom 4 (29.07.2019 until 25.08.2019)

### 4.3.1.3 Convolution Neural Network

Table 4: Dataset 1 CNN Error Statistics

| | MAE [Wh/Wh] | MSE [Wh/Wh] | RMSE [Wh/Wh] | CAE [Wh/Wh] | RMSE [%] | CAE [%] |
|---|---|---|---|---|---|---|
| Full | 0.338 | 0.235 | 0.485 | 11686 | 39 | 42 |
| Zoom 1 | 0.271 | 0.132 | 0.364 | 391 | 43 | 48 |
| Zoom 2 | 0.31 | 0.226 | 0.476 | 387 | -16 | 5 |
| Zoom 3 | 0.386 | 0.24 | 0.49 | 519 | 26 | 29 |
| Zoom 4 | 0.271 | 0.119 | 0.345 | 702 | 18 | 20 |

For Dataset 1, a total of 35 CNN models have been trained and tested, of which 26 achieved better results than baseline, with an overall mean RMSE of $0.907$ and a mean absolute error of $26442$.

The selected model showed the best prediction performance of all CNN models trained on Dataset 1. It used 4 hidden layers, with 39 neurons each a kernel size of 3, a pooling size of 5, 4 days as input and relu as activation function. *Table 4* lists the error statistics for the whole dataset ("full"), Zoom 1, Zoom 2, Zoom 3 and Zoom 4.

*Figure 36* shows prediction and ground truth using the CNN model for the whole test data. Similar to *Section 4.3.1.2*, the model is predicting drops in electricity demand up to July 2019 that are not present in the ground turth. Overall, the CNN model performs slightely worse than the ANN model presented in *Section 4.3.1.2* with an achieved RMSE value of $0.485$ and an absolute error of $11686$, representing an improvement of $39\%$ and $42\%$ respectively.

*Figure 37*, *Figure 38*, *Figure 39* and *Figure 39* show the graphs for Zoom 1, Zoom 2, Zoom 3 and Zoom 4, respectively.

Similar to the ANN model presented in *Section 4.3.1.2*, the model has learned the weekly pattern for the first half of Zoom 1. Starting from March 10th, the CNN model has difficulties to accurately predict the volatile energy consumption data.

In Zoom 2, the model is approximating the daily patterns of the first week and first weekend very well, is over estimating the consumption of March 20th and has, like the ANN model, a 24h delay before adapting to the dropped energy consumption of March 21st.

In Zoom 3, the CNN model is over-estimating the consumption for normal weekdays and Corpus Christi (March 20th), but is tremendously under-estimating the consumption for the bridge day (March 21st).

For Zoom 4, the CNN model is generally approximating the demand curve, however, compared to the ANN model, the CNN model is over-estimating many of the normal load peaks.

*Table 4* lists the performance metrics for the whole testset, Zoom 1, Zoom 2, Zoom 3 and Zoom 4.

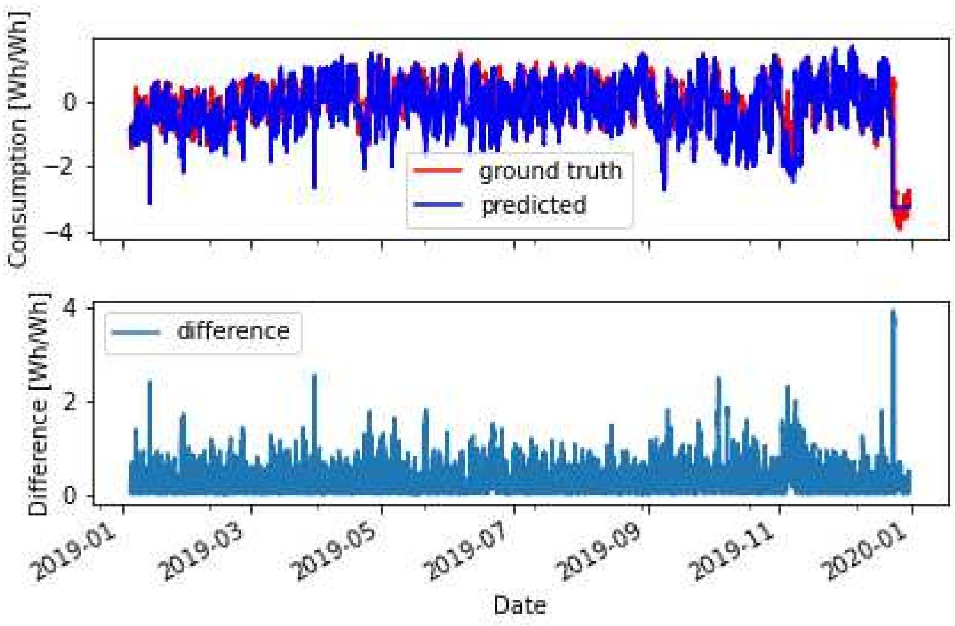

Figure 36: CNN Prediction for Dataset 1

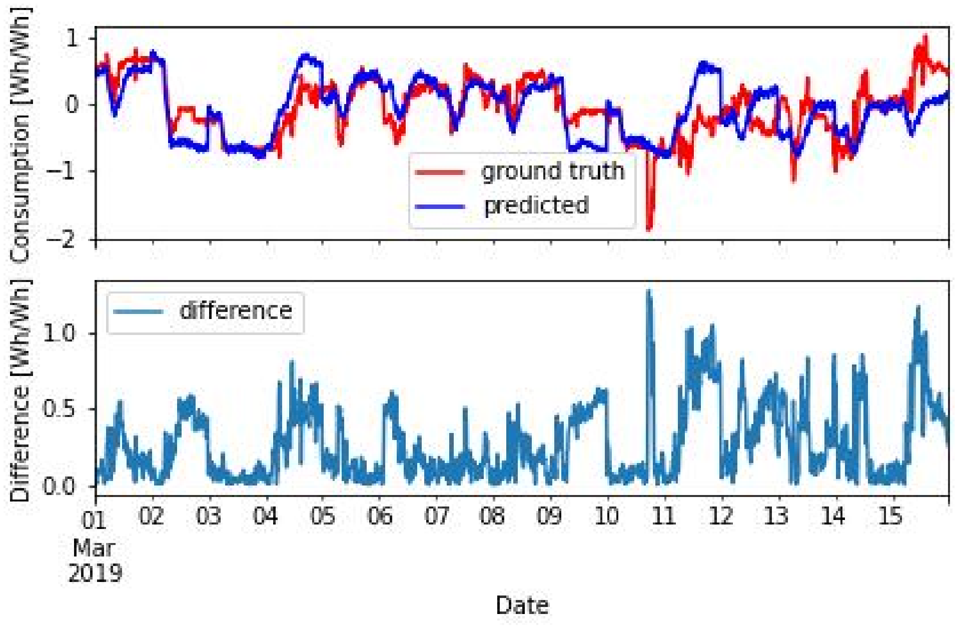

Figure 37: CNN Prediction for Dataset 1, Zoom 1 (01.03.2019 until 16.03.2019)

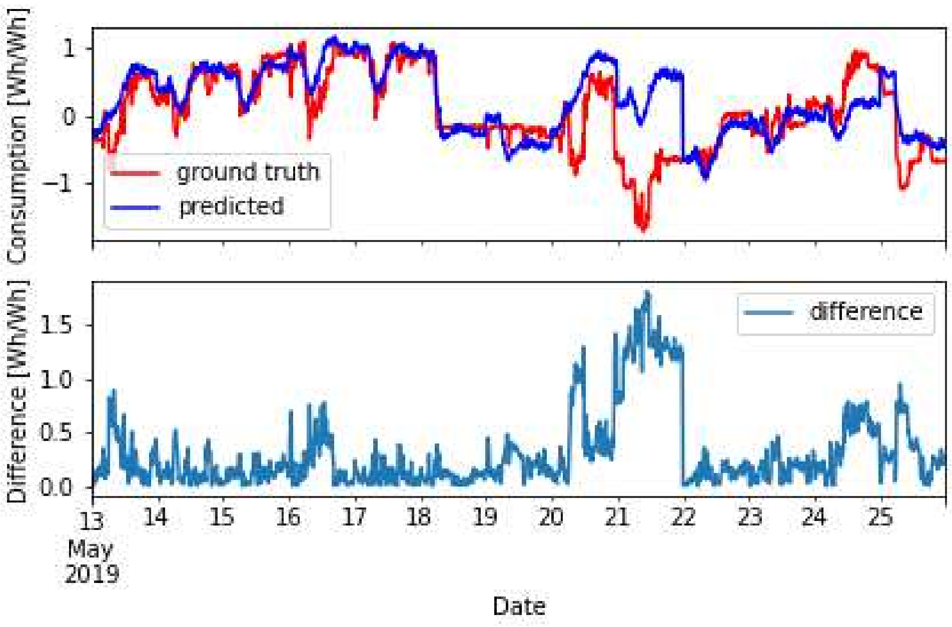

Figure 38: CNN Prediction for Dataset 1, Zoom 2 (Whit Monday 2018)

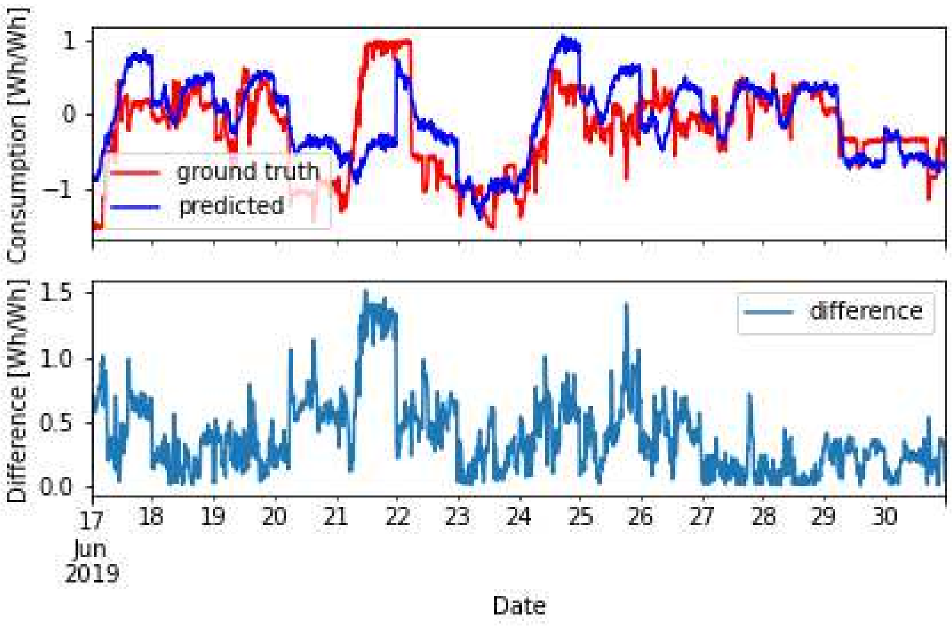

Figure 39: CNN Prediction for Dataset 1, Zoom 3 (Corpus Christi 2019)

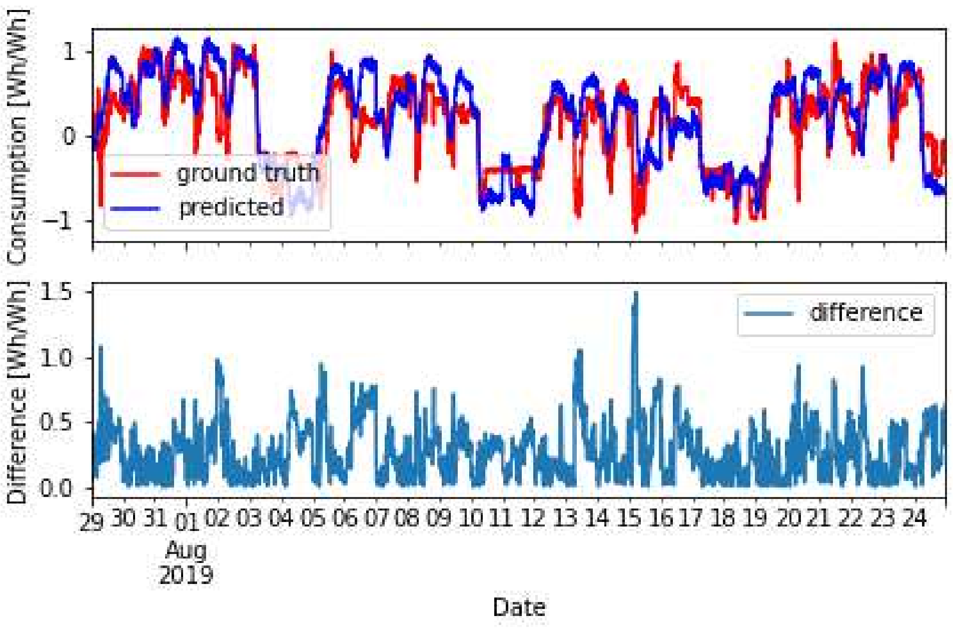

Figure 40: CNN Prediction for Dataset 1, Zoom 4 (29.07.2019 until 25.08.2019)

#### 4.3.1.4 Long-Short Term Memory

Table 5: Dataset 1 LSTM Error Statistics

| | MAE [Wh/Wh] | MSE [Wh/Wh] | RMSE [Wh/Wh] | CAE [Wh/Wh] | RMSE [%] | CAE [%] |
|---|---|---|---|---|---|---|
| Full | 0.318 | 0.195 | 0.441 | 10840 | 45 | 46 |
| Zoom 1 | 0.274 | 0.127 | 0.357 | 395 | 44 | 48 |
| Zoom 2 | 0.269 | 0.118 | 0.344 | 336 | 16 | 18 |
| Zoom 3 | 0.295 | 0.139 | 0.373 | 397 | 44 | 46 |
| Zoom 4 | 0.267 | 0.108 | 0.328 | 693 | 22 | 21 |

A total of 9 LSTM models have been tested on Dataset 1, achieving a mean RMSE of $0.469$, a mean absolute error of $11650$ and every model performed better than the Baseline. The LSTM training process took on day (24h) per model, for Dataset 1, limiting the number of trained models.

The selected LSTM model showed the best prediction performance of LSTM models trained on Dataset 1. It used 3 hidden layers, with 94 neurons each, *relu* as activation function and utilized 9 days of input data.

*Figure 41* depicts prediction and ground truth for Dataset 1, using the LSTM model for the whole test dataset, showing that the model has a tendency of over estimating. Further, the LSTM model is forecasting drops in energy demand not present in ground truth data, similar to the ANN an CNN models. Overall, the LSTM model achieves an RMSE of $0.441$ and a cumulative absolute error of 10840, achieving an improvement of $45\%$ and $56\%$ compared to the Baseline.

*Figure 42*, *Figure 43*, *Figure 44* and *Figure 45* show the graphs for Zoom 1, Zoom 2, Zoom 3 and Zoom 4.

Similar to the ANN and CNN models, evaluated in *Section 4.3.1.2* and *Section 4.3.1.3*, the model has learned the weekly pattern for the first half of Zoom 1 and produces error, when the ground truth experiences a negative peak on Match 10th. The LSTM prediction requires slightly above 24h to adapt to the change in electricity demand, due to the prediction precondition introduced in *Section 4.3.1.2*. On Zoom 1, the model achieved an RMSE of $0.357$ and an cumulative absolute error of $395$, improving by $44\%$ and $48\%$, respectively.

In Zoom 2, the model is anticipating the load demand for the work week between March 13th and March 18th, but it is under-estimating the demand compared to the ANN and CNN model. The load peak of March 20th is predicted with less error compared to the CNN model and the following predicted peak (March 21st) is smaller than for the ANN and CNN model. Further, the LSTM model is underestimating the demand for the period from March 22nd, compared to *Figure 33* and *Figure 38*. The LSTM model achieved an RMSE of $0.344$ and a cumulative absolute error of $336$, improving by $16\%$ and $18\%$ compared to the Baseline.

In Zoom 3, the LSTM model is producing less error during Corpus Christi 2019 (June 20th) and bounces back faster for the bridging day (June 21st), compared to the ANN and CNN model. The LSTM model achieved an RMSE of $0.373$ and a

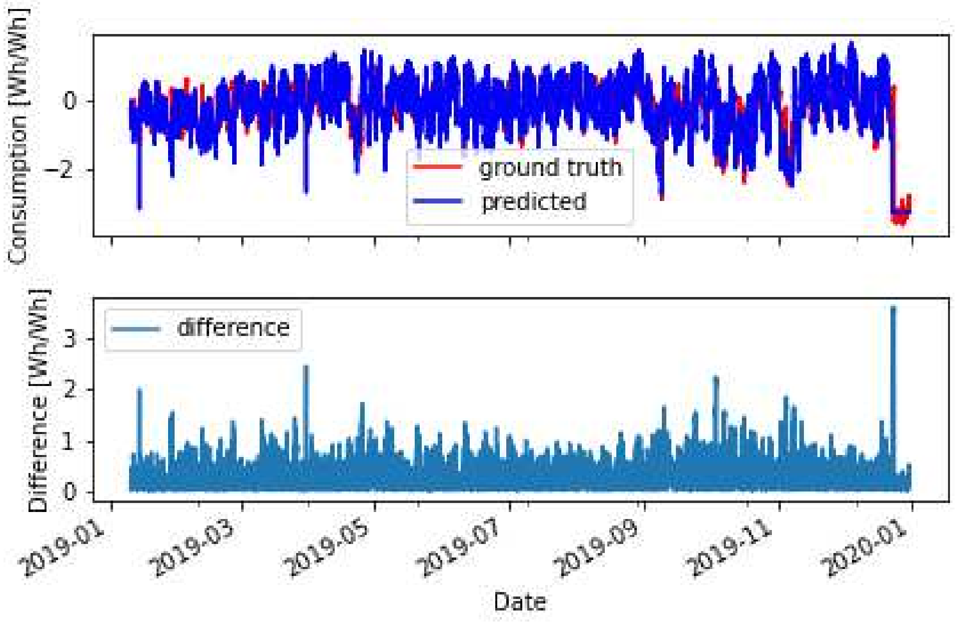

Figure 41: LSTM Prediction for Dataset 1

cumulative absolute error of 397 for Zoom 3, improving by $44\%$ and $46\%$ compared to the Baseline.

Assessing Zoom 4, the LSTM mdoel is generally less over-predicting and more often under-predicting, compared to the ANN and CNN model. It achieves an RMSE of $0.328$ and a cumulative absolute error of $693$, improving by $22\%$ and $21\%$ compared to the Baseline.

#### 4.3.1.5 Summary

Dataset 1 was provided by a pharmaceutical packaging manufacturer, with a high volatility in load demand. The bes prediction results were achieved by the LSTM model, reducing the error by $45\%$ for the RMSE and $46\%$ for the cumulative absolute error, compared to baseline. Performing worst was the CNN model, with an improvement of $39\%$ and $42\%$, respectively. The ANN model achieved an improvement of $42\%$ (RMSE) and $44\%$ (CAE), achieving just $3\%$ or $4\%$ worse results, than the LSTM model. With on hour average training time, the ANN models have been significantly cheaper to train than the LSTM models, requiring an average of 24h each on Dataset 1. Depending on the required retraining frequency, it may be advisable to use the cheaper ANN instead of the more accurate LSTM model.

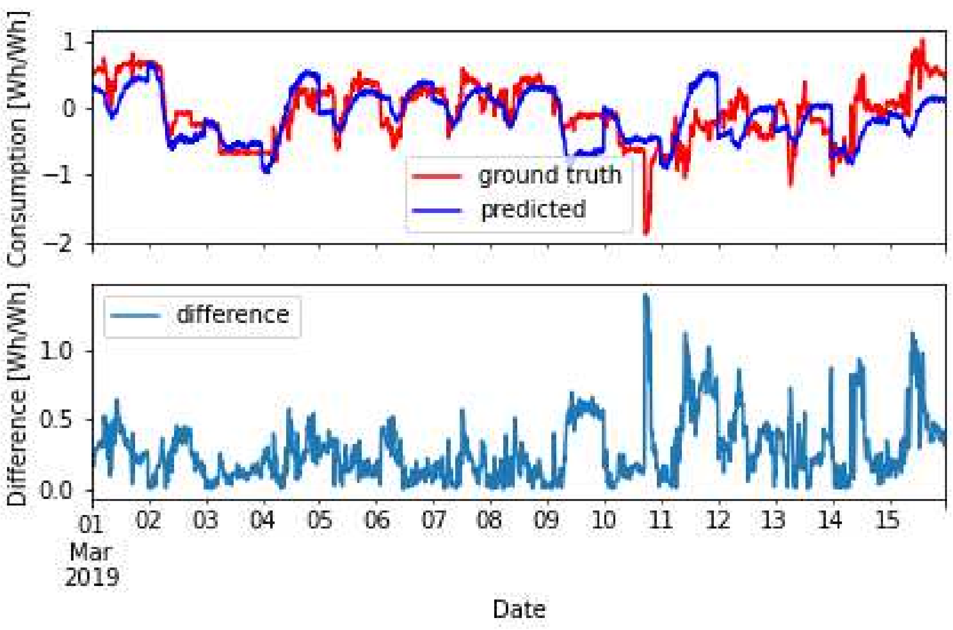

Figure 42: LSTM Prediction for Dataset 1, Zoom 1 (01.03.2019 until 16.03.2019)

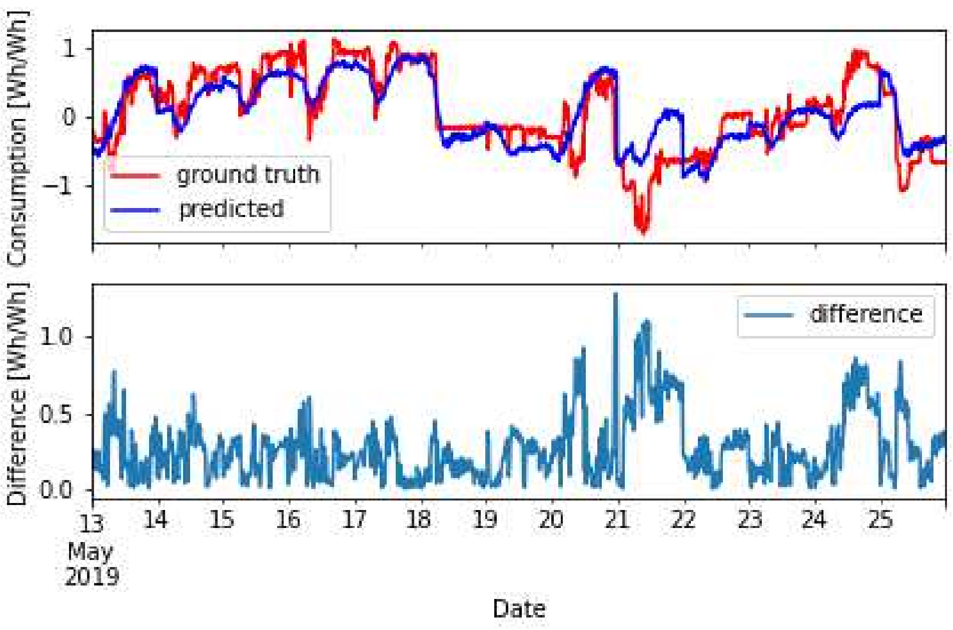

Figure 43: LSTM Prediction for Dataset 1, Zoom 2 (Whit Monday 2018)

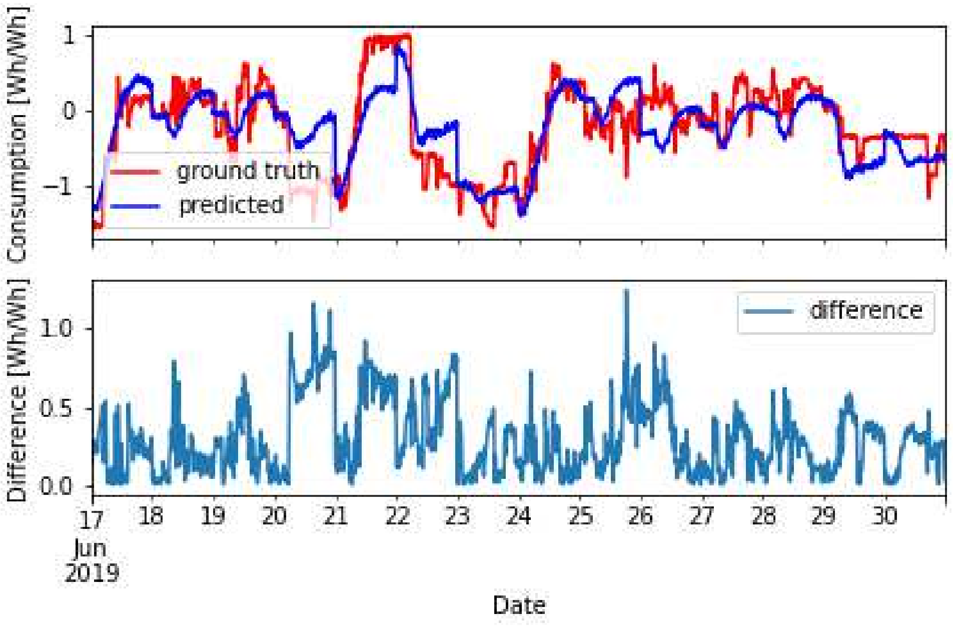

Figure 44: LSTM Prediction for Dataset 1, Zoom 3 (Corpus Christi 2019)

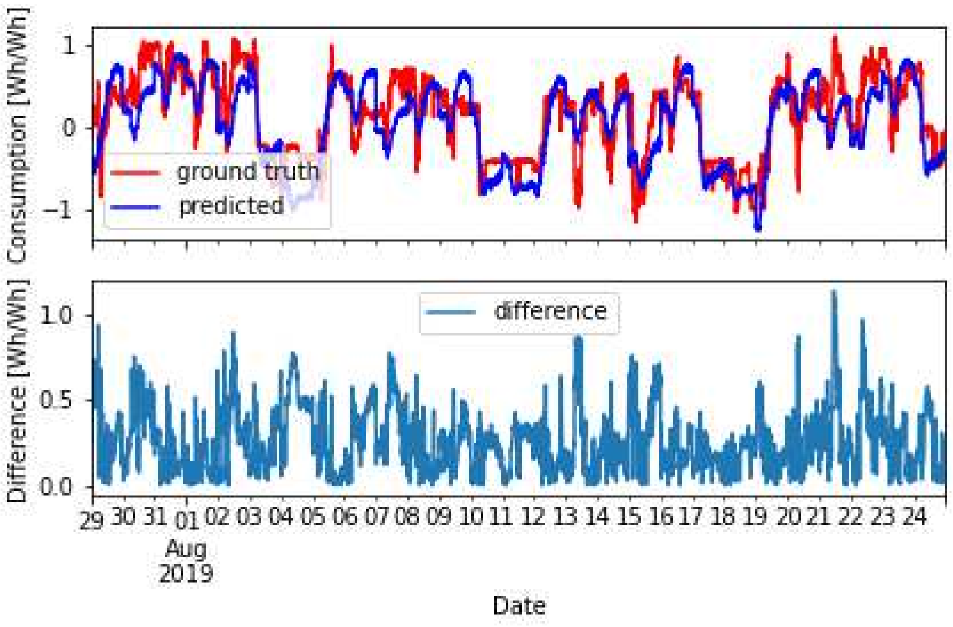

Figure 45: LSTM Prediction for Dataset 1, Zoom 4 (29.07.2019 until 25.08.2019)

### 4.3.2 Processing *Dataset 2*

Following, the test data of Dataset 2 is predicted using the Baseline and the best performing neural network model, convolution neural network model and long-short term memory model.

#### 4.3.2.1 Baseline

Table 6: Dataset 2 Baseline Error Statistics

| | MAE [Wh/Wh] | MSE [Wh/Wh] | RMSE [Wh/Wh] | CAE [Wh/Wh] |
|---|---|---|---|---|
| Full | 0.141 | 0.041 | 0.202 | 3693 |
| Zoom 1 | 0.178 | 0.071 | 0.266 | 257 |
| Zoom 2 | 0.148 | 0.038 | 0.196 | 185 |
| Zoom 3 | 0.156 | 0.047 | 0.217 | 209 |
| Zoom 4 | 0.134 | 0.040 | 0.200 | 349 |

*Figure 46* shows the Baseline prediction, using the baseline algorithm introduced in *Section 3.2*, for all test data of Dataset 2. The weekly demand cycles are clearly recognisable in the full plot. *Figure 46* suggests that the baseline is overestimating the demand from June 2019 until October 2019. Overall, the Baseline achieved an RMSE of $0.202$ and a cumulative absolute error of 3693 on Dataset 2. *Figure 47*, *Figure 48*, *Figure 49* and *Figure 50* are depicting the baseline predictions for Zoom 1, Zoom 2, Zoom 3 and Zoom 4.

For Zoom 1, the baseline is predicting a steep drop in Friday March 8th, producing significant error. In Zoom 2, the historic data used for the Baseline accurately predicts the energy demand of the test data. The holiday compensation for Whit Monday (March 20th) is clearly recognisable.

Looking at Zoom 3, the holiday adjustment for Corpus Christi 2019 can be detected easily and the bridgeday prediction (March 21st) is predicting the amplitude correctly. For Zoom 3, the Baseline achieves an RMSE of $0.217$ and a cumulative absolute error of $209$.

In Zoom 4, the adjustments for Assumption Day 2018 and 2019 are clearly recognisable on March 14th and March 15th, respectively. The Baseline achieves an RMSE of $0.2$ and a cumulative absolute error of $249$ for Zoom 4.

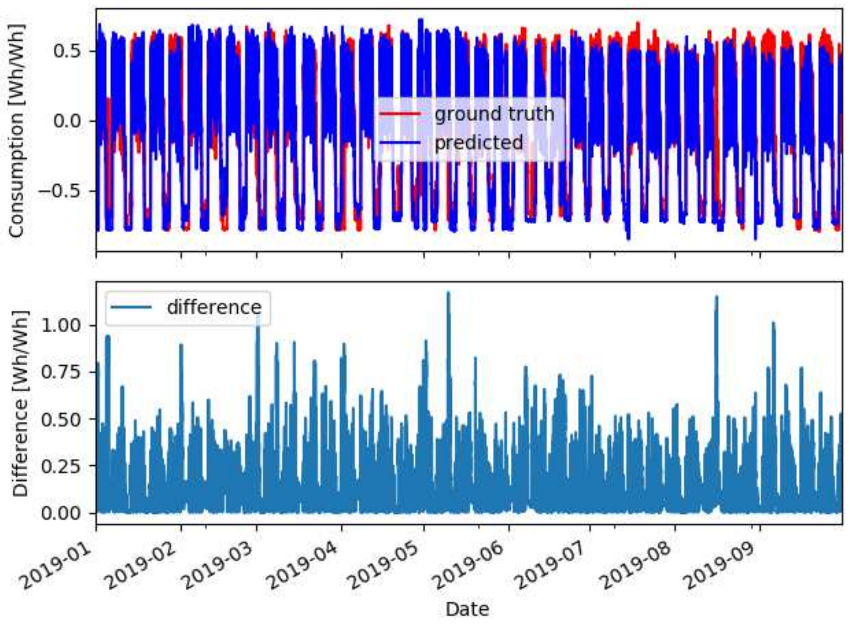

Figure 46: Baseline Prediction for Dataset 2

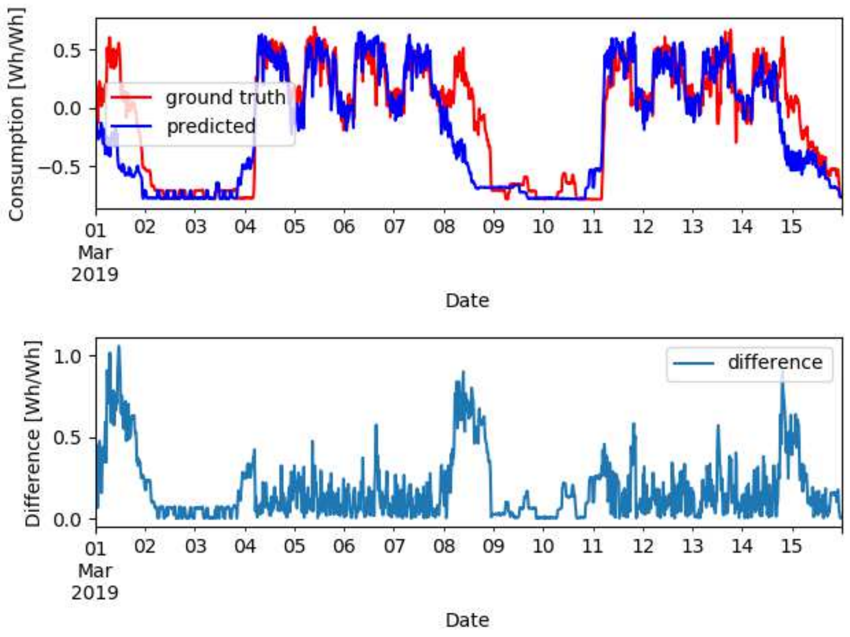

Figure 47: Baseline Prediction for Dataset 2, Zoom 1 (01.03.2019 until 16.03.2019)

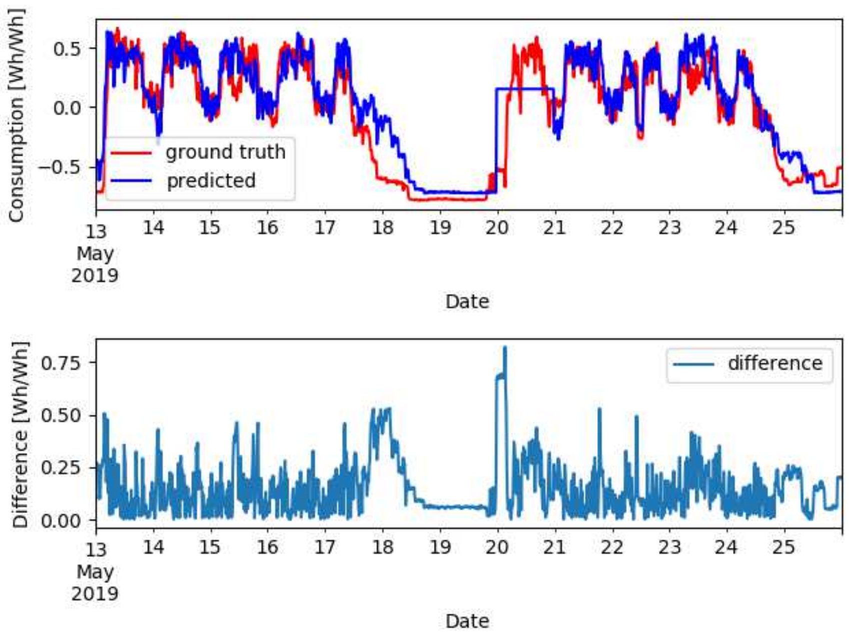

Figure 48: Baseline Prediction for Dataset 2, Zoom 2 (Whit Monday 2018)

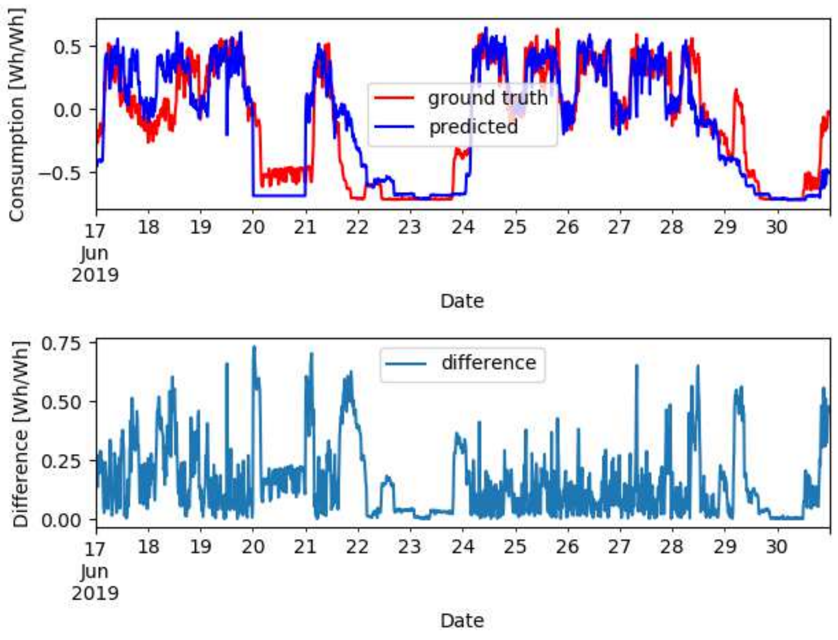

Figure 49: Baseline Prediction for Dataset 2, Zoom 3 (Corpus Christi 2019)

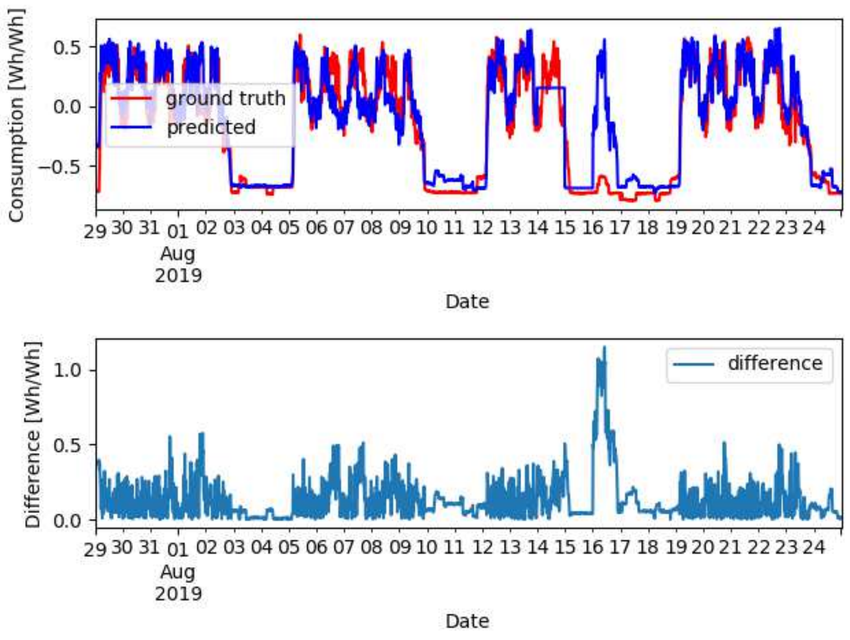

Figure 50: Baseline Prediction for Dataset 2, Zoom 4 (29.07.2019 until 25.08.2019)

#### 4.3.2.2 Artificial Neural Network

Table 7: Dataset 2 ANN Error Statistics

| | MAE [Wh/Wh] | MSE [Wh/Wh] | RMSE [Wh/Wh] | CAE [Wh/Wh] | RMSE [%] | CAE [%] |
|---|---|---|---|---|---|---|
| Full | 0.112 | 0.023 | 0.152 | 2866 | 25 | 22 |
| Zoom 1 | 0.126 | 0.03 | 0.174 | 181 | 35 | 30 |
| Zoom 2 | 0.11 | 0.02 | 0.142 | 137 | 28 | 26 |
| Zoom 3 | 0.143 | 0.036 | 0.191 | 192 | 12 | 8 |
| Zoom 4 | 0.102 | 0.018 | 0.134 | 265 | 33 | 24 |

For Dataset 2, a total of 30 ANN models have been trained and tested. 11 models performed better than baseline, with an overall RMSE of $0.283$ and a cumulative absolute error of $5728$. The selected model achieved an RMSE of $0.152$ and a cumulative absolute error of $2866$ on the test data. This is an improvement of $25\%$ and $22\%$, compared to the baseline.

*Figure 51* shows the prediction of the test data for Dataset 2, using the selected ANN model. *Figure 52*, *Figure 53*, *Figure 54* and *Figure 55* depict prediction and ground truth for Zoom 1, Zoom 2, Zoom 3 and Zoom 4.

When comparing Zoom 1 and Zoom 2, it can be seen, that the ANN model learned a specific pattern, approximating the actual consumption pattern. In Zoom 1, the time at the beginning of March 4th and March 11th produces a large error, not occurring in Zoom 3, indicating that the model learned a "generalized" pattern that is interrupted by irregularities. In Zoom 3, this "generalized" pattern is interrupted by Corpus Christi 2019, indicating that the model can cope with calendric variation.

The ANN model has learned to predict Fridays as work days, with slightly reduced load (e.g. May 17th or May 24th in Zoom 2). However, Fridays are not necessarily following the learned,"generalized' pattern (for example Friday March 15th in Zoom 1), deteriorating the prediction performance. The reason for the demand divergence can probably be found in the plants order situation or operative planning, not providing a learnable signal. Without a learnable signal, the algorithm is not able to discriminate between normal and reduced load Fridays.

The ANN model is accurately predicting the consumption drop, inflicted by Assumption Day 2019 (August 15th), but it is not expecting a bridge day with reduced demand on August 16th, introducing error by significantly over-estimating the load demand. This error could be circumvented by adding plant internal planning information to the algorithms input.

For Zoom 1, the algorithm achieved an RMSE of $0.174$ and a cumulative absolute error of $181$, improving by $35\%$ and $30\%$ compared to the baseline. For Zoom 2, an RMSE of $0.142$ and a cumulative absolute error of $137$ was achieved, improving the prediction by $28\%$ and $26\%$ respectively. In Zoom 3, the ANN model is performing worse than in Zoom 1, Zoom 2 or Zoom 4, achieving an RMSE of $0.191$ and a cumulative absolute error of $192$, improving the baseline by just $12\%$ and $8\%$.

Lastly, the ANN model was able to improve the RMSE by $33\%$ and the cumulative absolute error by $24\%$ for Zoom 4, compared to the baseline, achieving an RMSE value of $0.134$ and a cumulative absolute error of $265$.

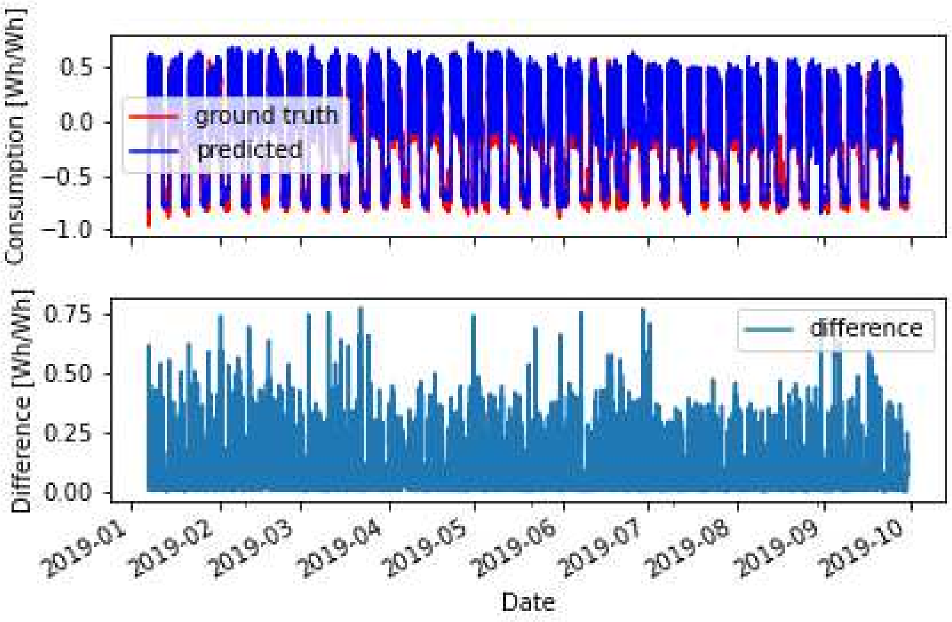

Figure 51: ANN Prediction for Dataset 2

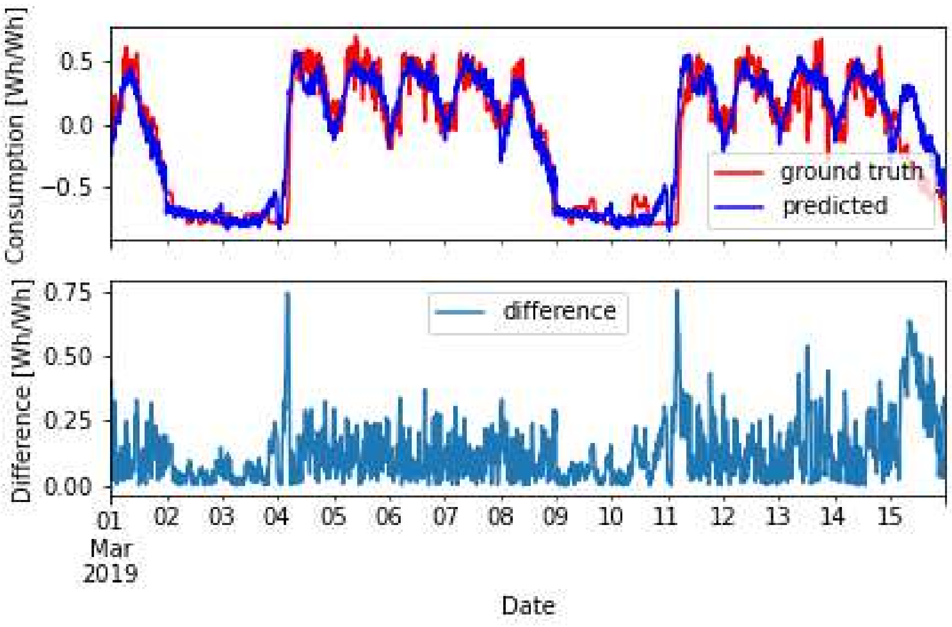

Figure 52: ANN Prediction for Dataset 2, Zoom 1 (01.03.2019 until 16.03.2019)

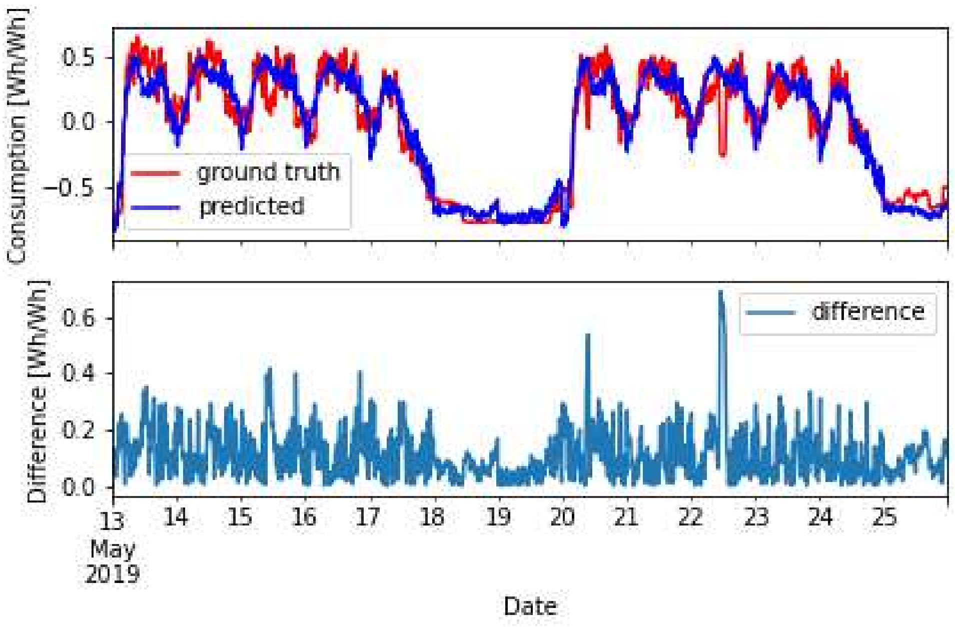

Figure 53: ANN Prediction for Dataset 2, Zoom 2 (Whit Monday 2018)

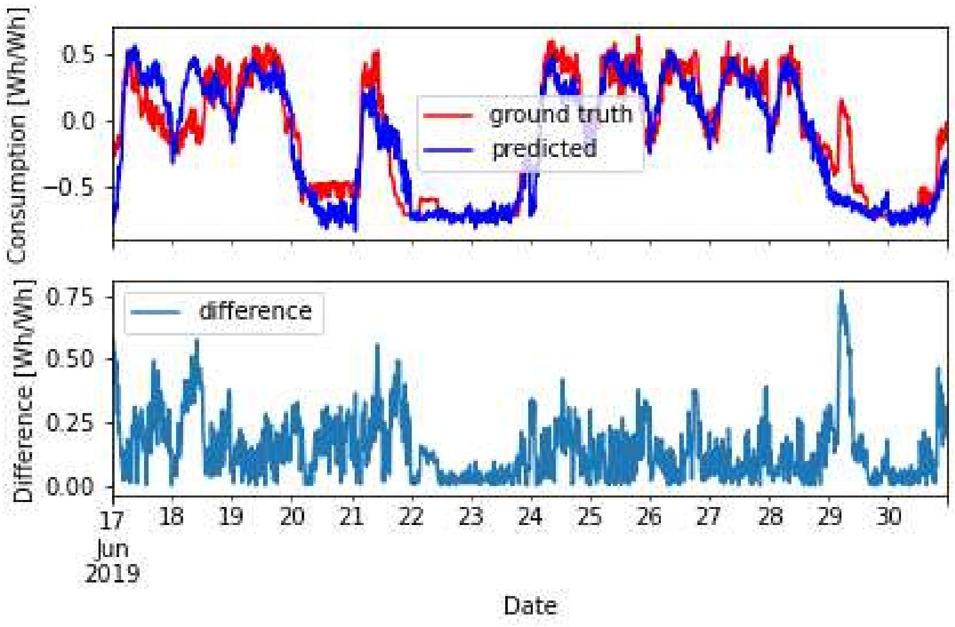

Figure 54: ANN Prediction for Dataset 2, Zoom 3 (Corpus Christi 2019)

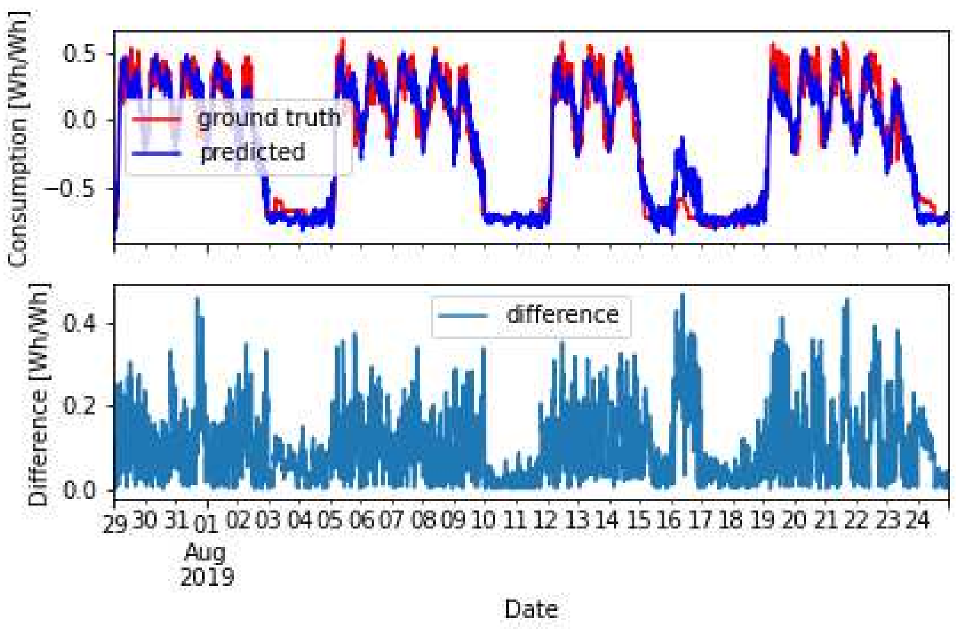

Figure 55: ANN Prediction for Dataset 2, Zoom 4 (29.07.2019 until 25.08.2019)

#### 4.3.2.3 Convolution Neural Network

Table 8: Dataset 2 CNN Error Statistics

| | MAE [Wh/Wh] | MSE [Wh/Wh] | RMSE [Wh/Wh] | CAE [Wh/Wh] | RMSE [%] | CAE [%] |
|---|---|---|---|---|---|---|
| Full | 0.113 | 0.023 | 0.151 | 2872 | 25 | 22 |
| Zoom 1 | 0.125 | 0.026 | 0.163 | 179 | 39 | 30 |
| Zoom 2 | 0.095 | 0.016 | 0.126 | 119 | 36 | 36 |
| Zoom 3 | 0.166 | 0.048 | 0.22 | 224 | -1 | -7 |
| Zoom 4 | 0.105 | 0.021 | 0.146 | 272 | 27 | 22 |

A total of 35 CNN models have been trained on Dataset 2, of which 12 performed better than the Baseline established in *Section 4.3.2.1*. All CNN models combined achieved an overall mean RMSE of $0.567$ and a mean cumulative error of $18404$. The selected model is comprised of 6 hidden layers with 80 neurons each, uses a kernel size of 3 and a pooling size of 4, relu as activation function and 8 days of input data.

*Figure 56* shows the prediction and ground truth of Dataset 2. The selected CNN model achieved an RMSE value of $0.151$, improving the prediction accuracy by $25\%$ compared to the Baseline and a cumulative absolute error of $2872$, improving the prediction accuracy by $22\%$.

*Figure 57*, *Figure 58*, *Figure 59* and *Figure 60* depict prediction, ground truth and error for Zoom 1, Zoom 2, Zoom 3 and Zoom 4.

The predicted demand pattern shown in Zoom 1 indicates, that the model learned a general regression function, smoothing the load patterns. Due to this smoothing property, the model performs bad in predicting steep demand changes, causing the prediction error to increase each Monday (e.g. March 4th or March 11th in Zoom 1). The absolute errors for work days are comparable with the absolute errors produced by the ANN model, levelling around 0.2 for both.

Generally, the CNN model has difficulties to accurately predict the demand for the days before and after Corpus Christi 2019, generating absolute errors of about $0.5$, exceeding the work week error of Zoom 1 twice.

In Zoom 4, the CNN model is achieving results comparable to the ANN model, producing absolute errors between 0.2 and 0.25 for normal work weeks. However, the bridge day after Mary Assumption (August 16th) is significantly overestimated, generating absolute errors of 0.75 and more, compared to the ANN model producing absolute errors of $0.4 - 0.5$ in *Figure 55*.

The CNN model achieved an RMSE of $0.163$ and a cummulative absolute error of 179, for Zoom 1, improving by $39\%$ and $30\%$. For Zoom 2, it achieved an RMSE of $0.126$, improving by $36\%$ and a cumulative absolute error of $119$, improving by $36\%$. On Zoom 3, the model achieved worse results compared to the Baseline, scoring an RMSE of 0.22 and a cumulative absolute error of 224. For Zoom 4, the model achieved a improvement of $27\%$ and $22\%$ respectively, producing an RMSE value of $0.146$ and a cumulative absolute error of $272$.

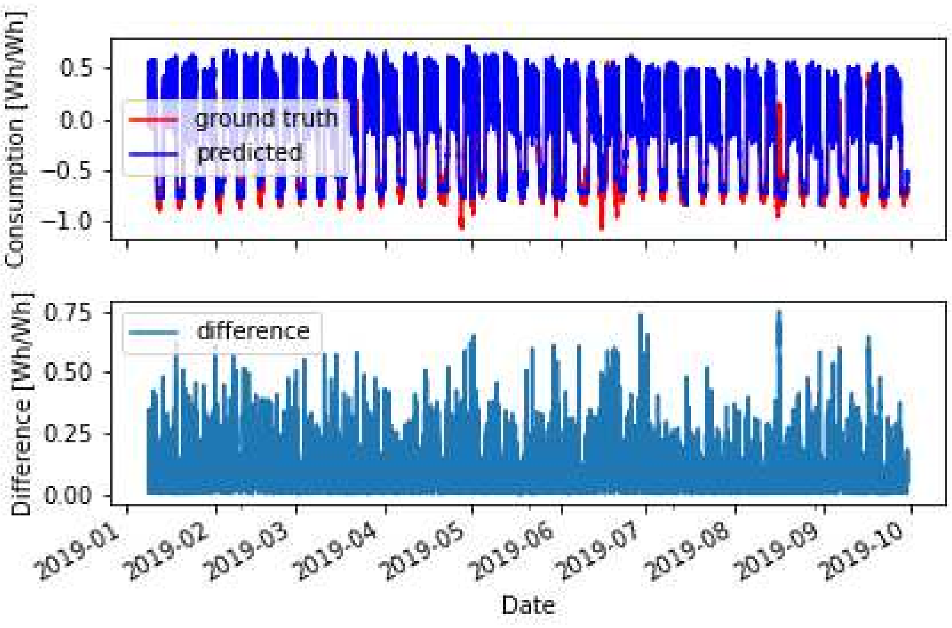

Figure 56: CNN Prediction for Dataset 2

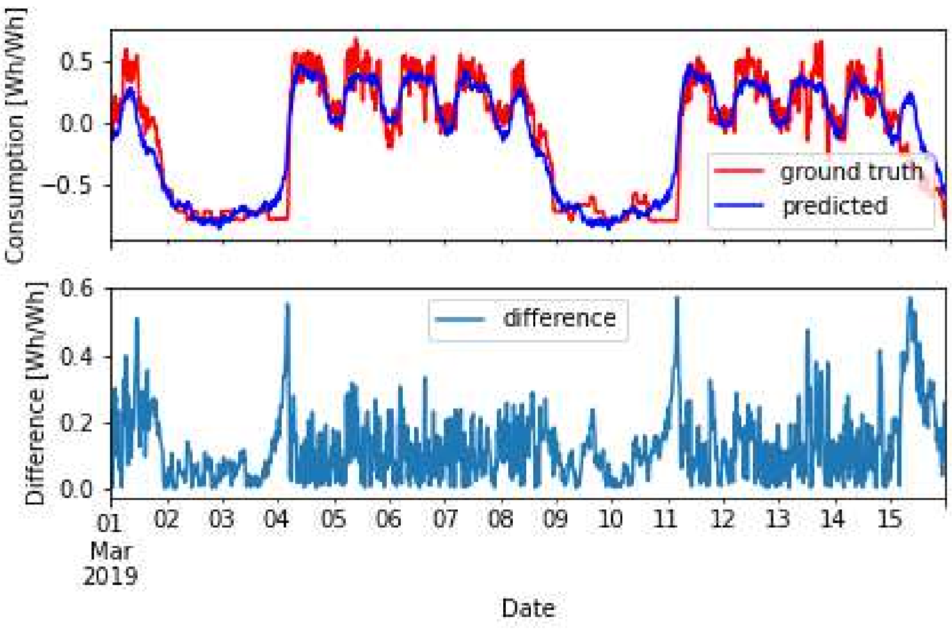

Figure 57: CNN Prediction for Dataset 2, Zoom 1 (01.03.2019 until 16.03.2019)

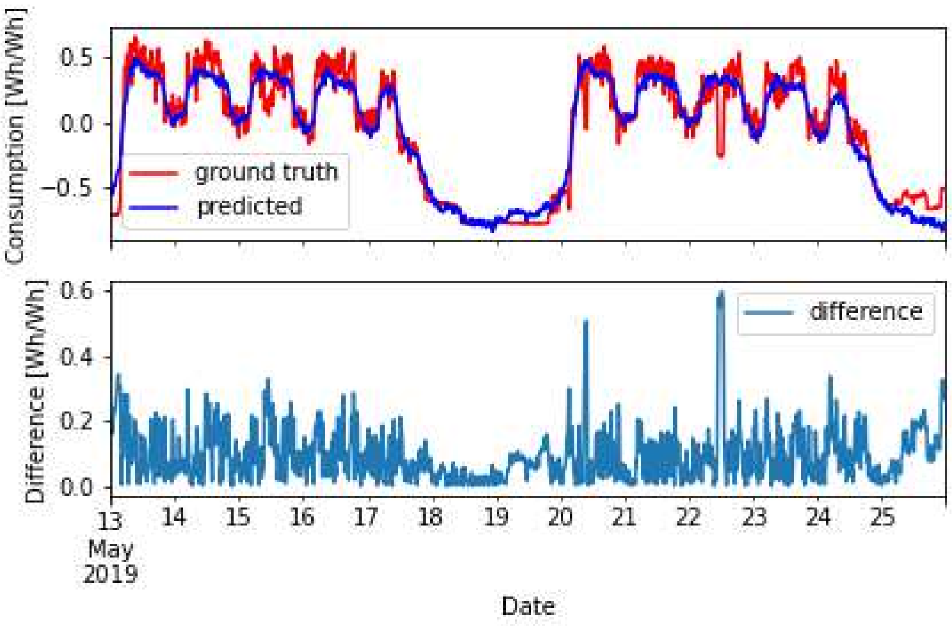

Figure 58: CNN Prediction for Dataset 2, Zoom 2 (Whit Monday 2018)

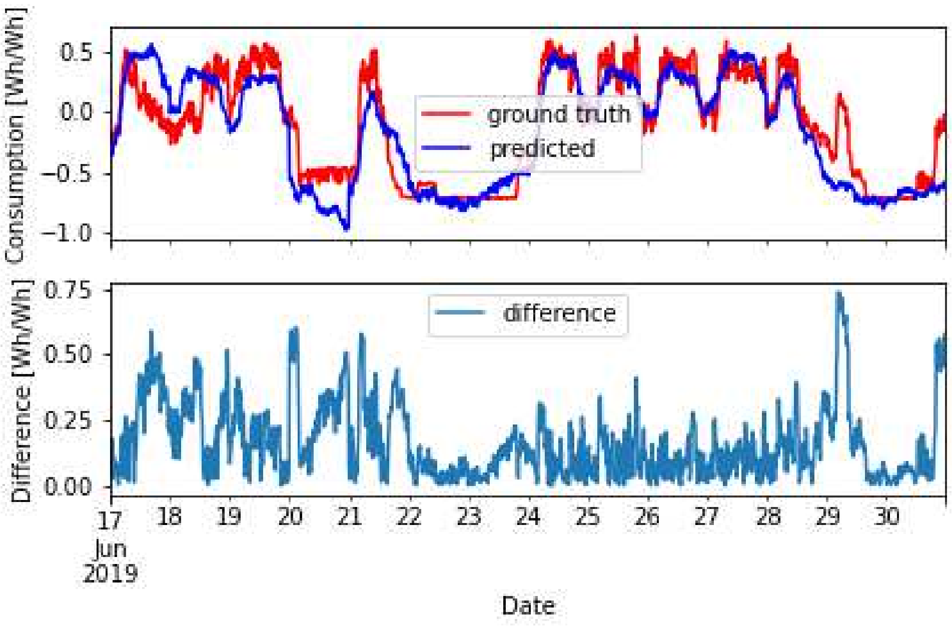

Figure 59: CNN Prediction for Dataset 2, Zoom 3 (Corpus Christi 2019)

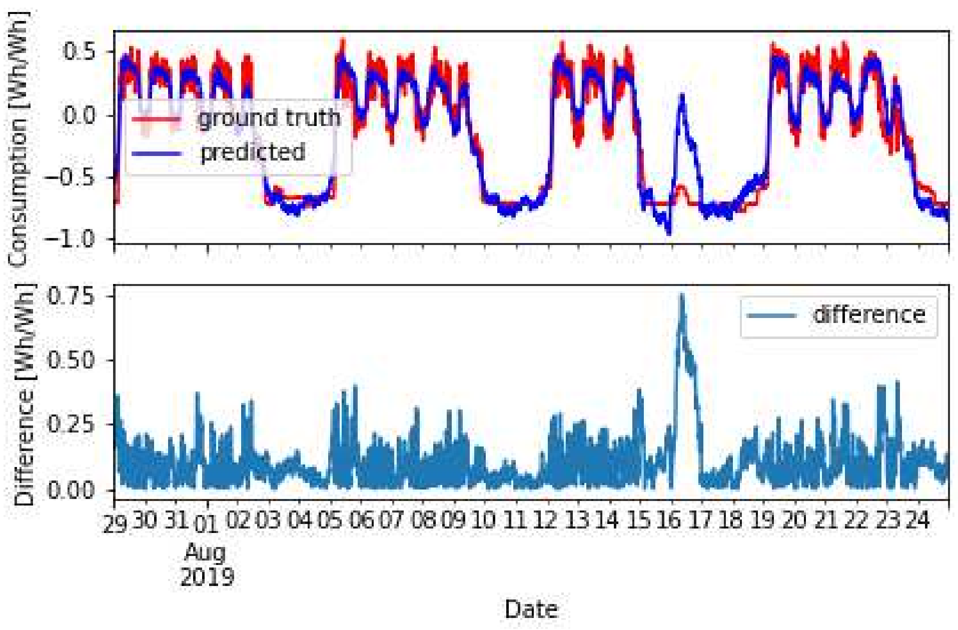

Figure 60: CNN Prediction for Dataset 2, Zoom 4 (29.07.2019 until 25.08.2019)

#### 4.3.2.4 Long-Short Term Memory

Table 9: Dataset 2 LSTM Error Statistics

| | MAE [Wh/Wh] | MSE [Wh/Wh] | RMSE [Wh/Wh] | CAE [Wh/Wh] | RMSE [%] | CAE [%] |
|---|---|---|---|---|---|---|
| Full | 0.103 | 0.022 | 0.148 | 2671 | 27 | 28 |
| Zoom 1 | 0.117 | 0.029 | 0.17 | 168 | 36 | 35 |
| Zoom 2 | 0.105 | 0.019 | 0.138 | 132 | 30 | 28 |
| Zoom 3 | 0.13 | 0.037 | 0.191 | 174 | 12 | 17 |
| Zoom 4 | 0.105 | 0.027 | 0.165 | 273 | 18 | 22 |

For Dataset 2, a total of 30 LSTM models have been trained and tested, and 22 models performed better than the Baseline, with an overall mean RMSE of $0.188$ and a mean cumulative absolute error of $3217$. The selected model uses one hidden layer, with 73 neurons each, facilitated a tanh activation function and used three days as input data. The model achieved an RMSE of $0.148$ and a cumulative absolute error of $2671$ on the test data, improving the prediction by $27\%$ and $28\%$, respectively.

*Figure 61* shows the LSTM prediction and ground truth for Dataset 2, indicating that the majority of absolute errors are peaking between $0.25$ and $0.5$.

*Figure 62*, *Figure 63*, *Figure 64* and *Figure 65* depict prediction, ground truth and error for Zoom 1, Zoom 2, Zoom 3 and Zoom 4.

Comparing Zoom 1 to Zoom 4 indicates, that the selected LSTM model is generally under-estimating the load demand for Mondays and that the model has learned to condense the demand pattern to a regression function, generally ignoring intra-days patterns. The LSTM model has the same difficulties to predict low-load Fridays (March 15th in Zoom 1) correctly, as the ANN and CNN model, supporting the assumption that signals are missing to correctly predict Fridays load demand. Additionally, the LSTM model is equally bad in predicting the bridge day (August 16th) of Zoom 4.

#### 4.3.2.5 Summary

Dataset 2 was provided by a paper manufacturing plant. The demand pattern is repetitive, with daily and weekly variations clearly identifiable. The ANN and CNN model are performing equally well, reducing the error by $25\%$ (RMSE) or $22\%$ (cumulative absolute error). The best results are achieved by the LSTM mode, improving prediction performance by $27\%$ (RMSE) or $28\%$ (CAE) compared to the Baseline. Besides intra-day variation, the most significant error is caused by unpredictable load patterns on Fridays and unexpected load reduction on bridge days. When implementing LSTM for Company B, it is suggested to provide internal planing information as algorithm input to eliminate prediction inaccuracies for Fridays, bridge days and intra-day deviations.

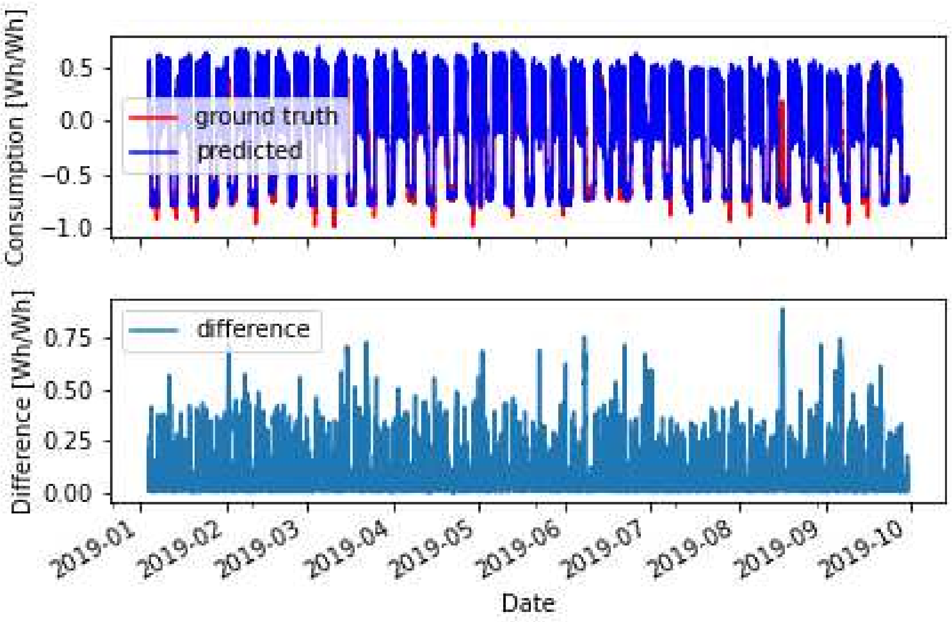

Figure 61: LSTM Prediction for Dataset 2

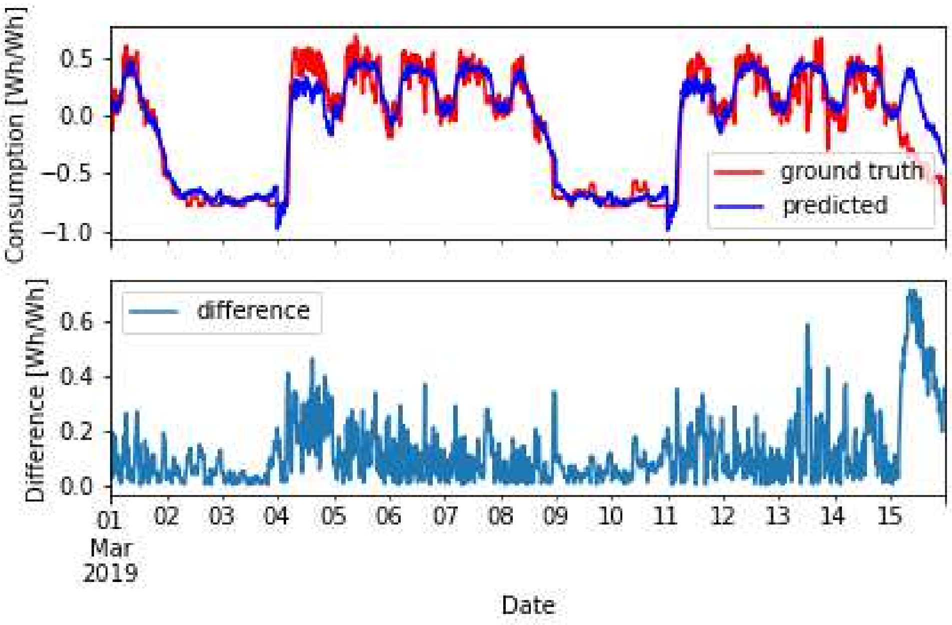

Figure 62: LSTM Prediction for Dataset 2, Zoom 1 (01.03.2019 until 16.03.2019)

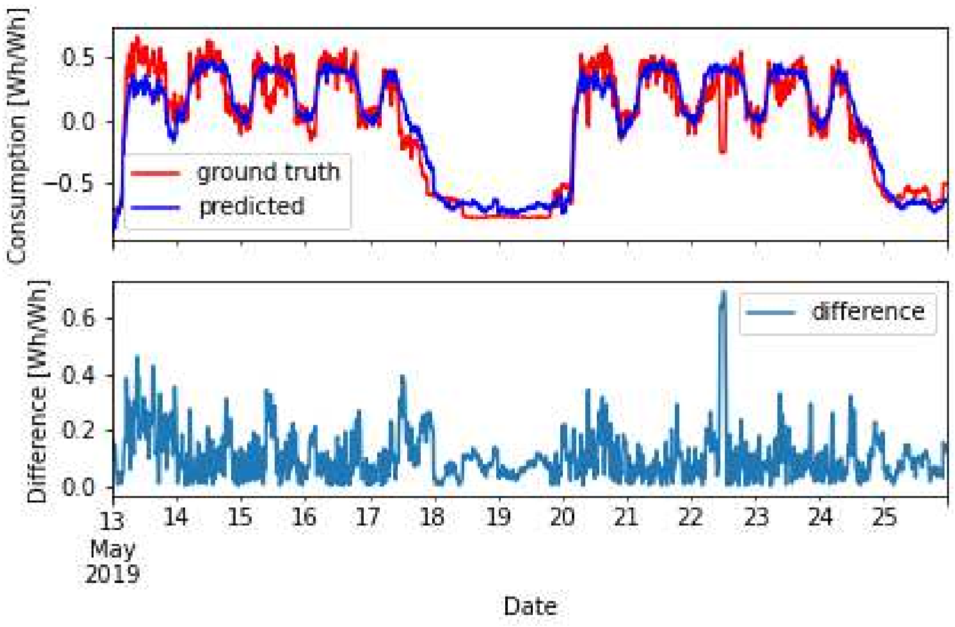

Figure 63: LSTM Prediction for Dataset 2, Zoom 2 (Whit Monday 2018)

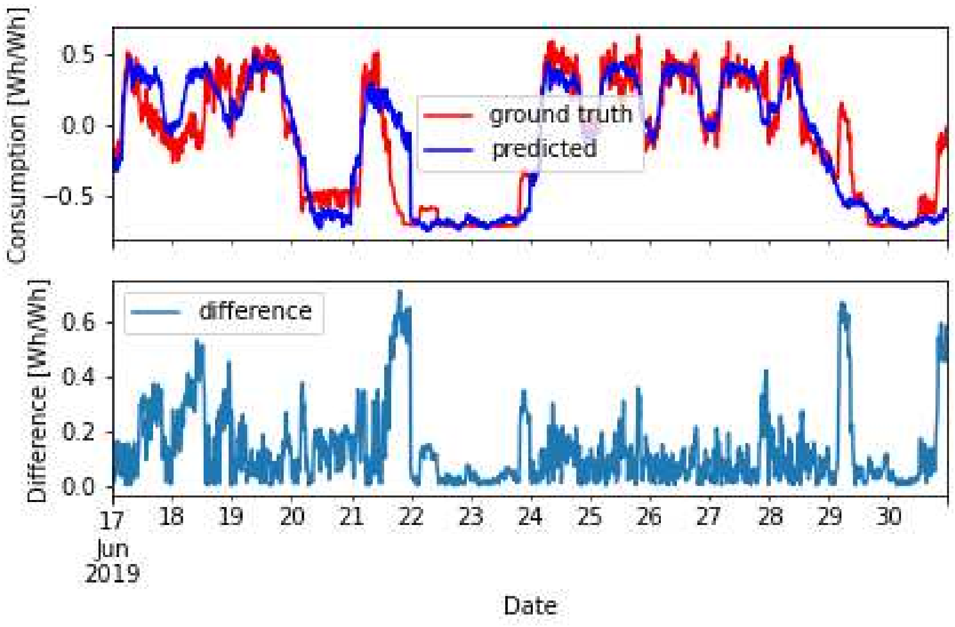

Figure 64: LSTM Prediction for Dataset 2, Zoom 3 (Corpus Christi 2019)

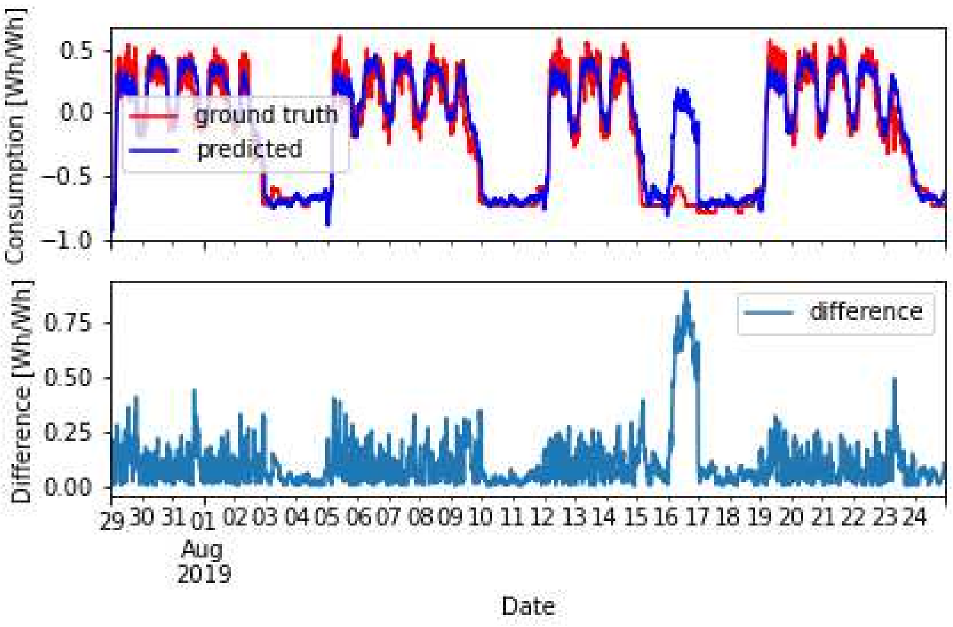

Figure 65: LSTM Prediction for Dataset 2, Zoom 4 (29.07.2019 until 25.08.2019)

### 4.3.3 Processing *Dataset 3*

#### 4.3.3.1 Baseline

Table 10: Dataset 3 Baseline Error Statistics

| | MAE [Wh/Wh] | MSE [Wh/Wh] | RMSE [Wh/Wh] | CAE [Wh/Wh] |
|---|---|---|---|---|
| Full | 0.203 | 0.095 | 0.309 | 7128 |
| Zoom 1 | 0.156 | 0.037 | 0.193 | 224 |
| Zoom 2 | 0.210 | 0.086 | 0.293 | 262 |
| Zoom 3 | 0.233 | 0.118 | 0.343 | 313 |
| Zoom 4 | 0.162 | 0.064 | 0.253 | 420 |

*Figure 66* shows the Baseline prediction for Dataset 3 using the Baseline algorithm introduced in *Section 3.2* for the entire test set, indicating a repetitive load demand pattern, with three maintenance or holiday periods. Overall, the Baseline achieved an RMSE of $0.309$ and a cumulative absolute error of $7128$.

*Figure 67*, *Figure 68*, *Figure 69* and *Figure 70* depict prediction, ground truth and error for Zoom 1, Zoom 2, Zoom 3 and Zoom 4.

Zoom 1 to Zoom 4 reinforce the assumption of highly repetitive demand patterns, suggesting limited improvement potential. In Zoom 2, the baseline algorithm is generating error by replacing Whit Monday 2018 with the mean consumption of a workday. The drop of Corpus Christi 2019 (June 20th) is accounted for by using the mean consumption of off-days as forecast. However, the bridge day after Corpus Christi 2019 (June 21st) has been a normal workday in 2018, resulting in a false prediction.

Zoom 4 shows a two week holiday/maintenance period of Company C. The data of the previous year is accurately predicting the consumption for the first week. For the second week, the Baseline Algorithm is generating an error by replacing Assumption Day 2018 (August 14th in *Figure 70*) with the workday mean and Assumption Day 2019 (August 15th in *Figure 70*) with the off-day mean, both forecasting higher loads than necessary for the holiday period. *Table 10* lists the performance statistics for the Baseline of Dataset 3.

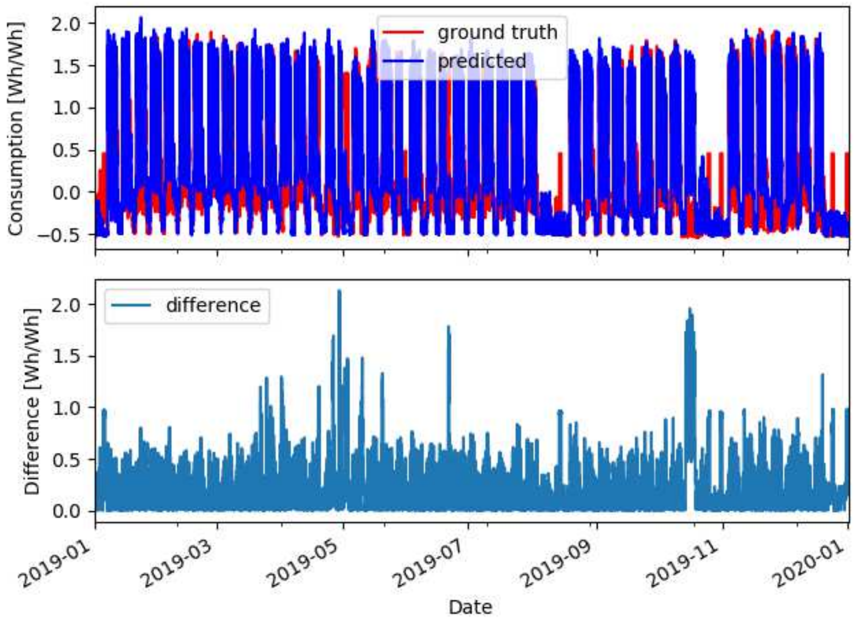

Figure 66: Baseline Prediction for Dataset 3

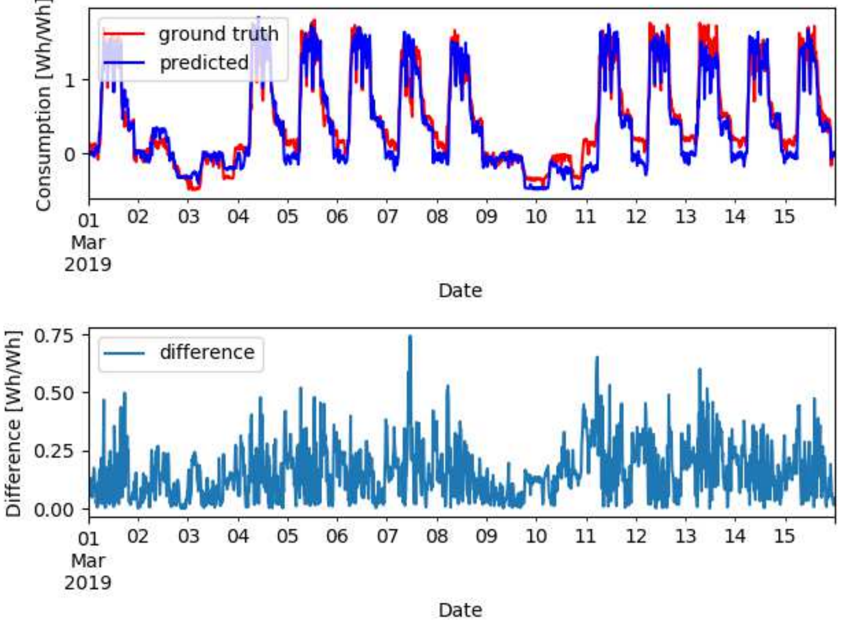

Figure 67: Baseline Prediction for Dataset 3, Zoom 1 (01.03.2019 until 16.03.2019)

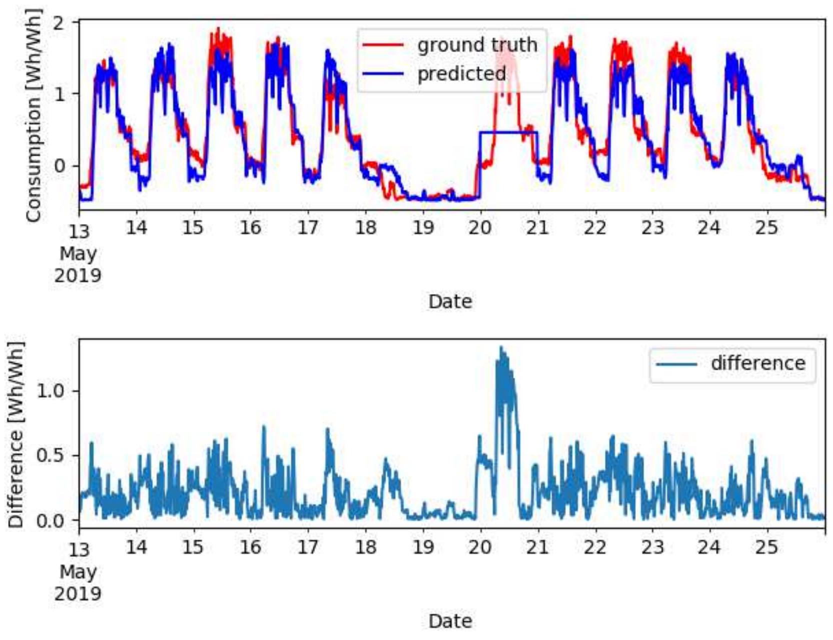

Figure 68: Baseline Prediction for Dataset 3, Zoom 2 (Whit Monday 2018)

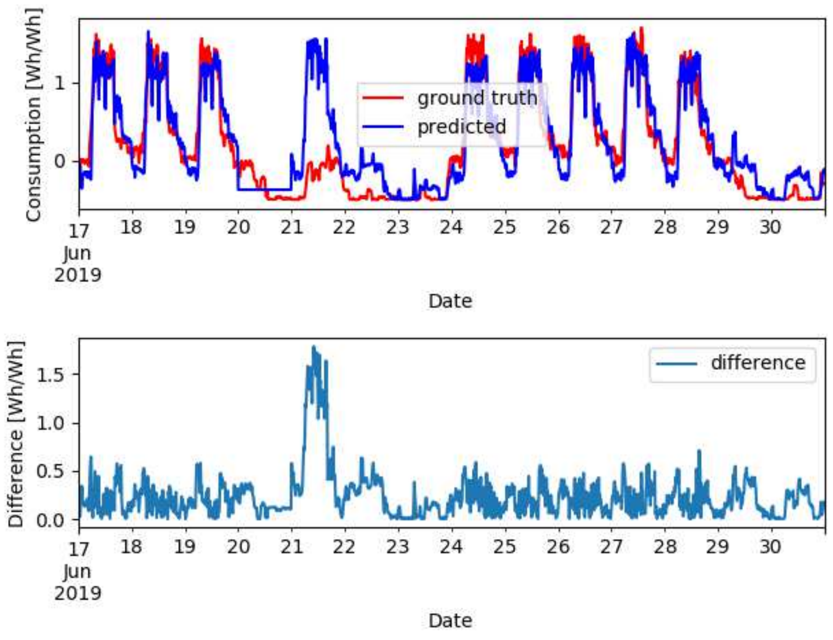

Figure 69: Baseline Prediction for Dataset 3, Zoom 3 (Corpus Christi 2019)

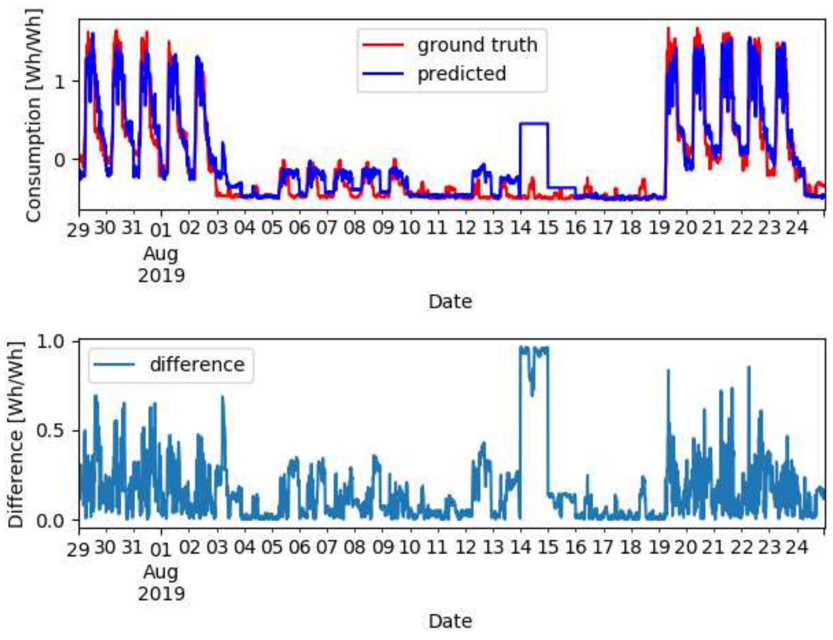

Figure 70: Baseline Prediction for Dataset 3, Zoom 4 (29.07.2019 until 25.08.2019)

#### 4.3.3.2 Artificial Neural Network

Table 11: Dataset 3 ANN Error Statistics

| | MAE [Wh/Wh] | MSE [Wh/Wh] | RMSE [Wh/Wh] | CAE [Wh/Wh] | RMSE [%] | CAE [%] |
|---|---|---|---|---|---|---|
| Full | 0.21 | 0.098 | 0.313 | 8168 | -1 | -15 |
| Zoom 1 | 0.219 | 0.094 | 0.306 | 315 | -58 | -40 |
| Zoom 2 | 0.21 | 0.088 | 0.296 | 262 | -1 | 0 |
| Zoom 3 | 0.22 | 0.102 | 0.319 | 296 | 7 | 5 |
| Zoom 4 | 0.186 | 0.111 | 0.333 | 482 | -32 | -15 |

For Dataset 3, a total of 60 models have been trained, of which none has performed better than Baseline. The selected model has performed best among the trained models, achieving an RMSE of $0.313$ and a cumulative absolute error of $8168$, performing $1\%$ and $15\%$ worse then Baseline.

*Figure 71* shows the overall ground truth, predicted load demand and the absolute errors. *Figure 72*, *Figure 73*, *Figure 74*, and *Figure 75* show prediction, ground truth and absolute errors for Zoom 1, Zoom2, Zoom 3 and Zoom 4.

The model has learned a regressive function to predict the future load demand, but as the demand patterns of 2018 and 2019 are highly recurring, the learned regression curve is performing significantly worse than the Baseline, achieving an RMSE of $0.306$ and a cumulative absolute error of $315$, deteriorating the performance by $58\%$ and $40\%$ compared to Baseline.

Only for Zoom 3, the ANN model is able to outperform the Baseline by $7\%$ for the RMSE and $5\%$ for the cumulative absolute error, as it is good in predicting the reduced load demand for the bridge day. The ANN model has learned to quickly identify holiday/maintenance periods completely adjusting to the drop in load demand after an adoption period of 24h, as shown in Zoom 4. However, while adapting to the holiday/maintenance mode quickly, the model is not anticipating the return no normal demand behaviour requiring another adaptation period of 24h. The performance metrics for the whole testset, Zoom 1, Zoom 2, Zoom 3 and Zoom 4 are listed in *Table 11*.

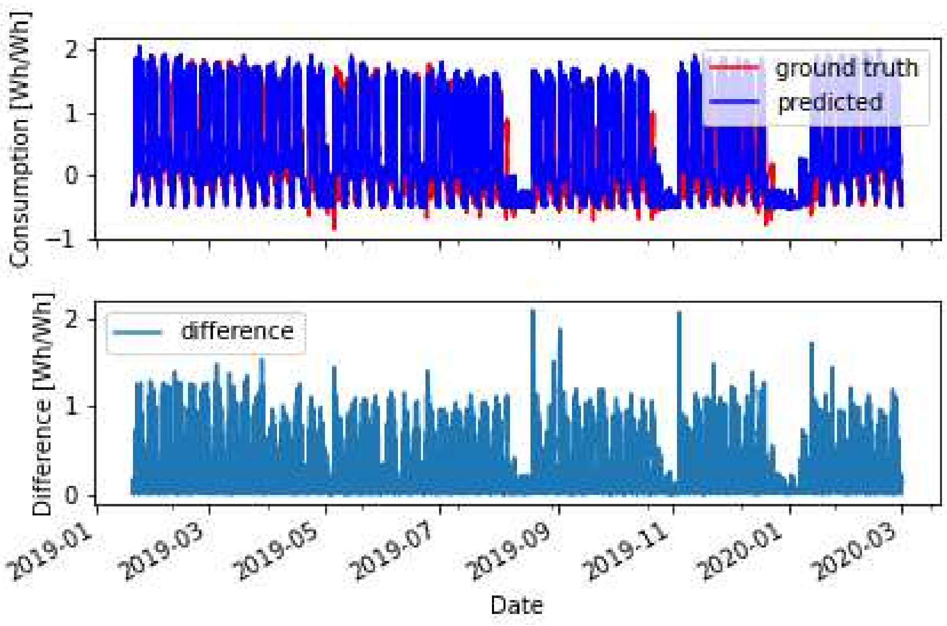

Figure 71: ANN Prediction for Dataset 3

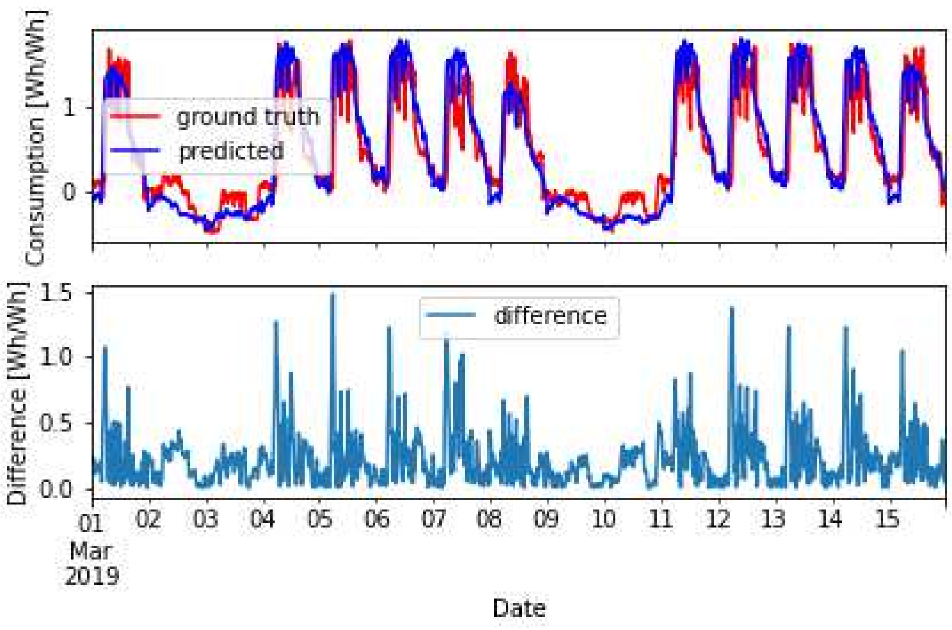

Figure 72: ANN Prediction for Dataset 3, Zoom 1 (01.03.2019 until 16.03.2019)

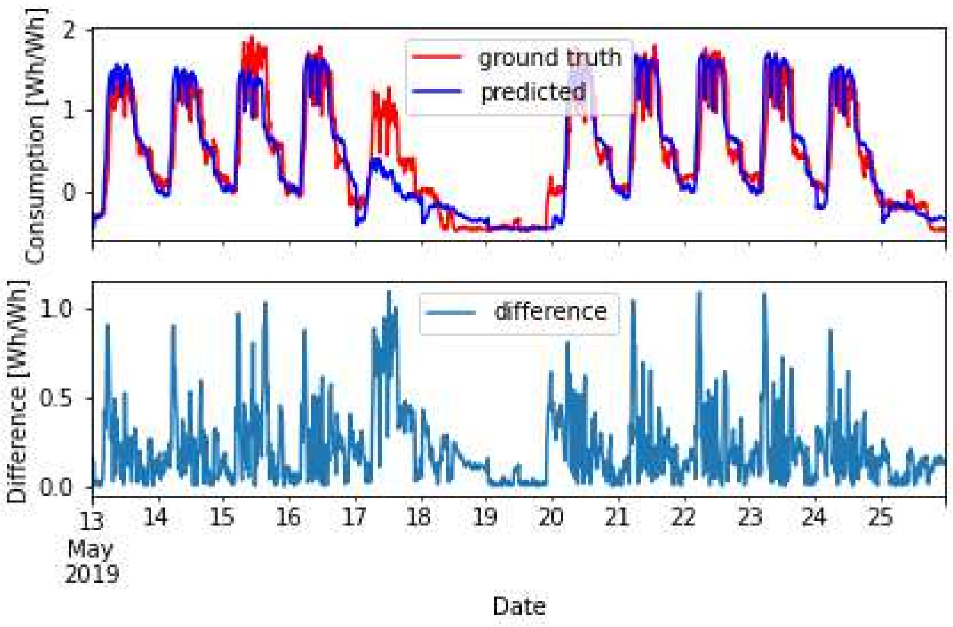

Figure 73: ANN Prediction for Dataset 3, Zoom 2 (Whit Monday 2018)

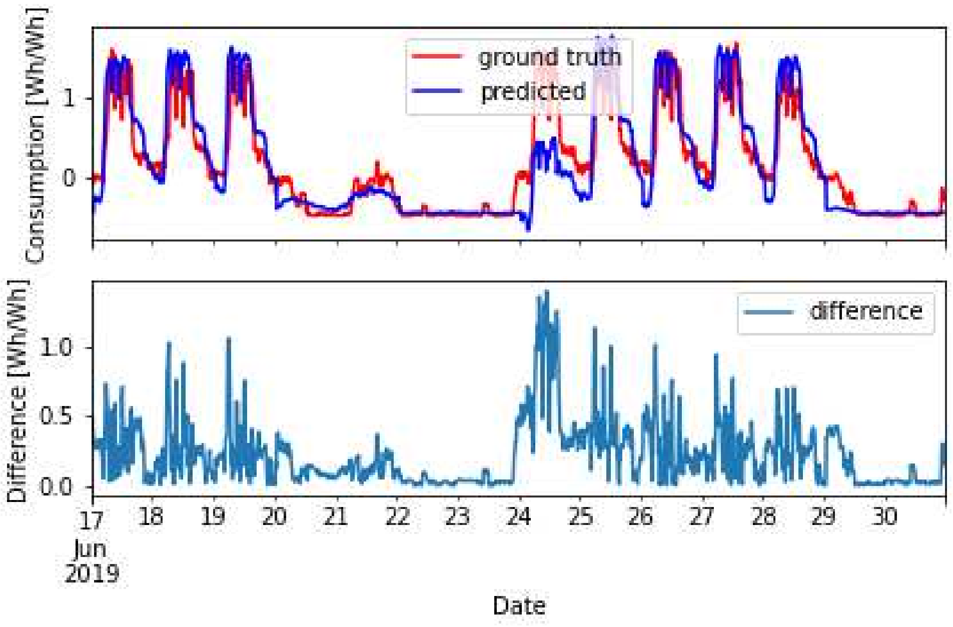

Figure 74: ANN Prediction for Dataset 3, Zoom 3 (Corpus Christi 2019)

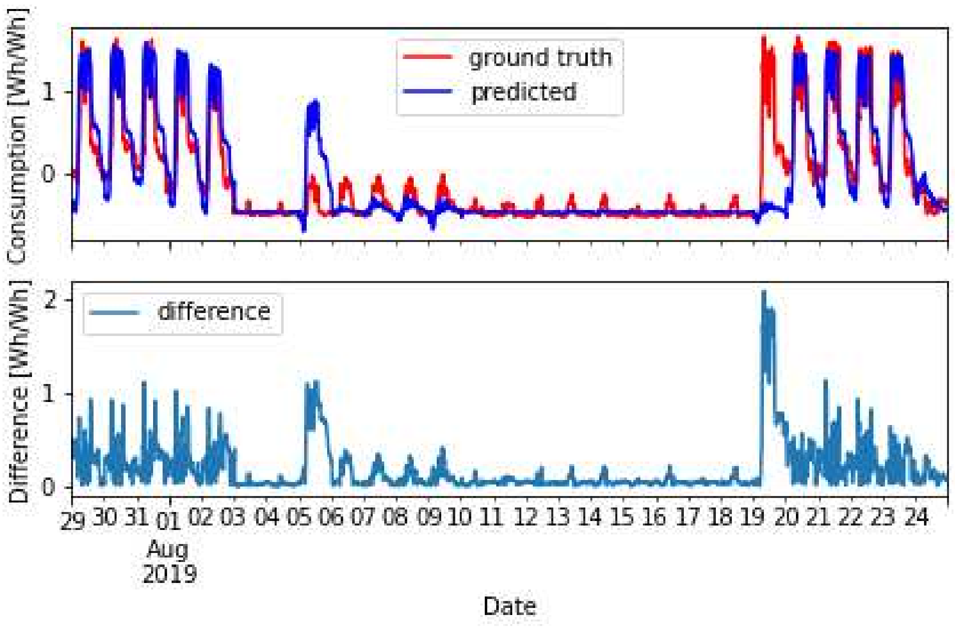

Figure 75: ANN Prediction for Dataset 3, Zoom 4 (29.07.2019 until 25.08.2019)

### 4.3.3.3 Convolution Neural Network

Table 12: Dataset 3 CNN Error Statistics

| | MAE [Wh/Wh] | MSE [Wh/Wh] | RMSE [Wh/Wh] | CAE [Wh/Wh] | RMSE [%] | CAE [%] |
|---|---|---|---|---|---|---|
| Full | 0.161 | 0.055 | 0.234 | 6516 | 24 | 9 |
| Zoom 1 | 0.173 | 0.053 | 0.231 | 249 | -20 | -11 |
| Zoom 2 | 0.149 | 0.039 | 0.198 | 187 | 32 | 29 |
| Zoom 3 | 0.13 | 0.033 | 0.18 | 174 | 48 | 44 |
| Zoom 4 | 0.192 | 0.09 | 0.299 | 499 | -18 | -19 |

A total of 35 CNN models have been trained and tested on Dataset 3, achieving a mean RMSE of $0.567$ and a mean cumulative absolute error of $18404$. 12 models performed better than the Baseline. The selected CNN model was built using one hidden layer, 65 neurons, a kernel size of 3 and a pooling size of 4, utilizing tanh as activation function and four days as input data. The selected CNN model achieved an RMSE of $0.234$ and a cumulative absolute error of $6516$ on the whole test set, improving by $24\%$ and $9\%$ compared to baseline.

*Figure 76* shows the overall ground truth, predicted load demand and the absolute errors. *Figure 77*, *Figure 78*, *Figure 79*, and *Figure 80* show prediction, ground truth and absolute errors for Zoom 1, Zoom2, Zoom 3 and Zoom 4.

Comparing the weekday predictions for Zoom 1, Zoom 2 and Zoom 4, it is obvious that the model has learned different patterns, for different load conditions (higher load demand predicted for Mondays in Zoom 1 and equal to lower load demand predicted for Mondays in Zoom 2 and Zoom 4). The model is accurately predicting Corpus Christi 2019 (June 20th) and is also performing well in predicting the bridge day after Corpus Christi 2019 (June 21st). In Zoom 1, the model seems to have issues predicting the intra-day demand accurately and in Zoom 4, the model had difficulties reducing the predicted load to zero for the holiday/maintenance period.

The CNN model is performing worse than the Baseline for Zoom 1 and Zoom 4, scoring RMSEs of $0.053$ and $0.09$ and cumulative absolute errors of $249$ and $499$.

*Table 12* lists the performance metrics for the selected CNN model.

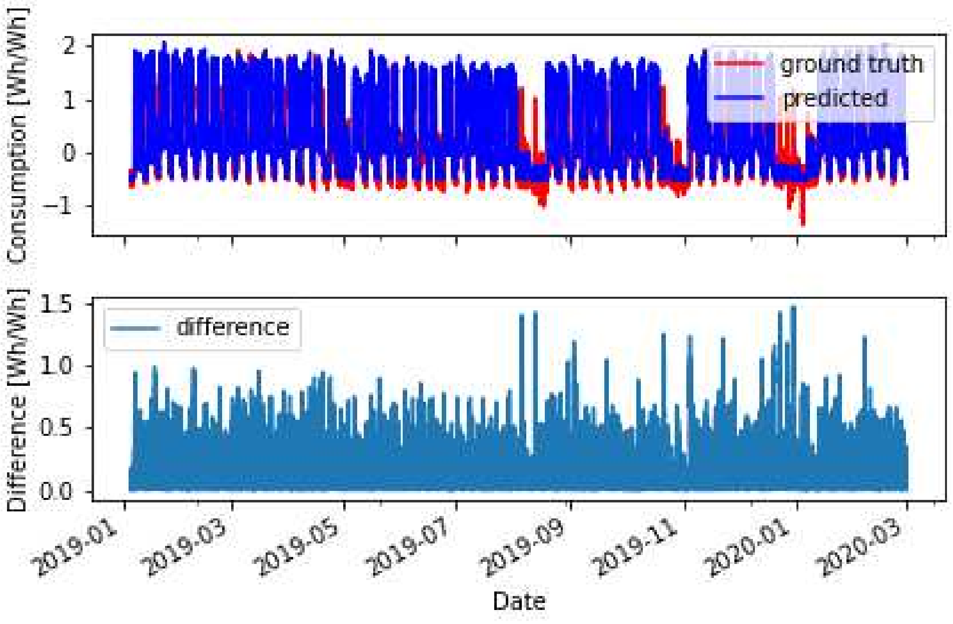

Figure 76: CNN Prediction for Dataset 3

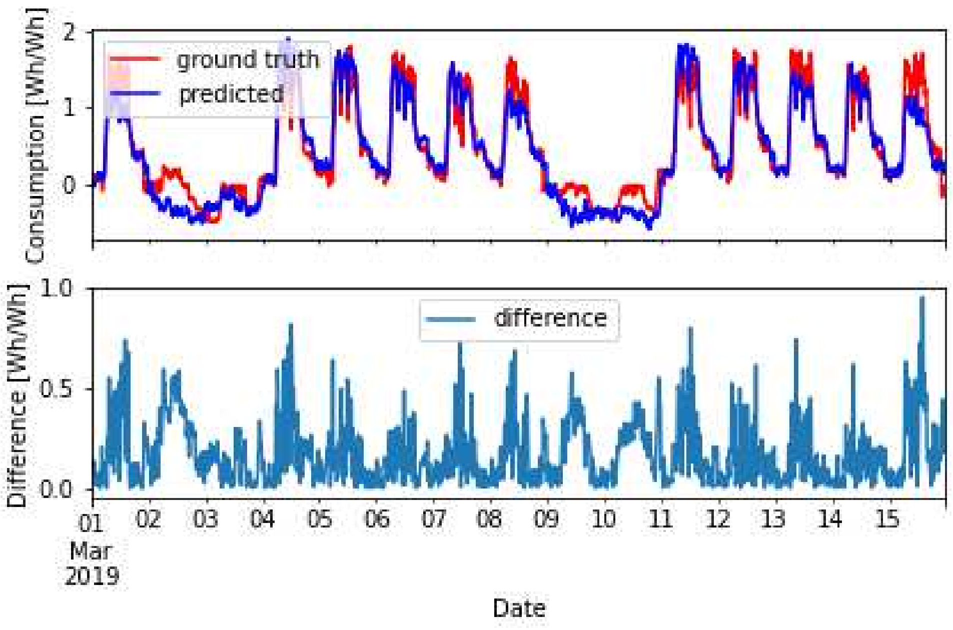

Figure 77: CNN Prediction for Dataset 3, Zoom 1 (01.03.2019 until 16.03.2019)

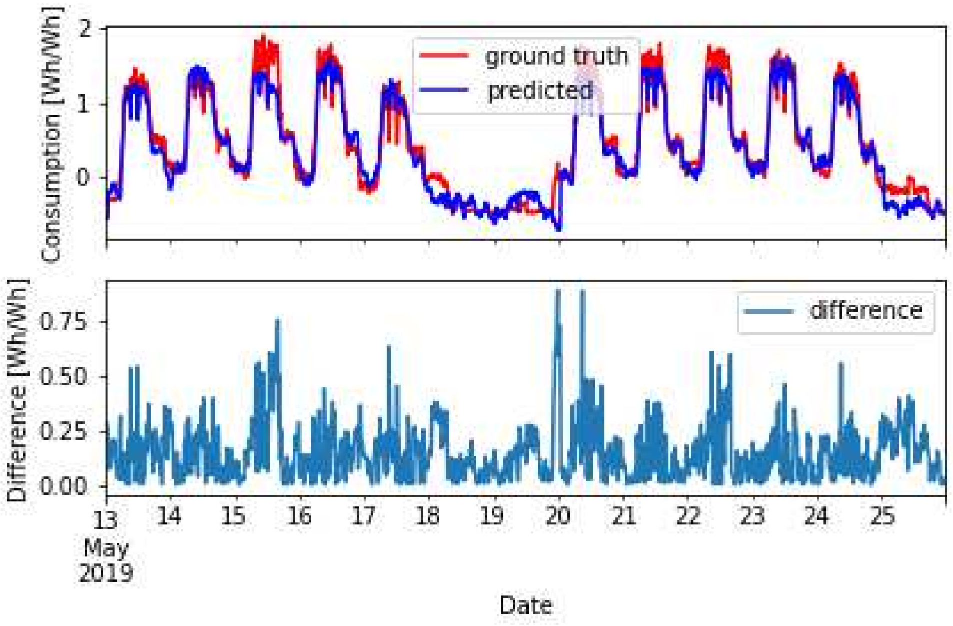

Figure 78: CNN Prediction for Dataset 3, Zoom 2 (Whit Monday 2018)

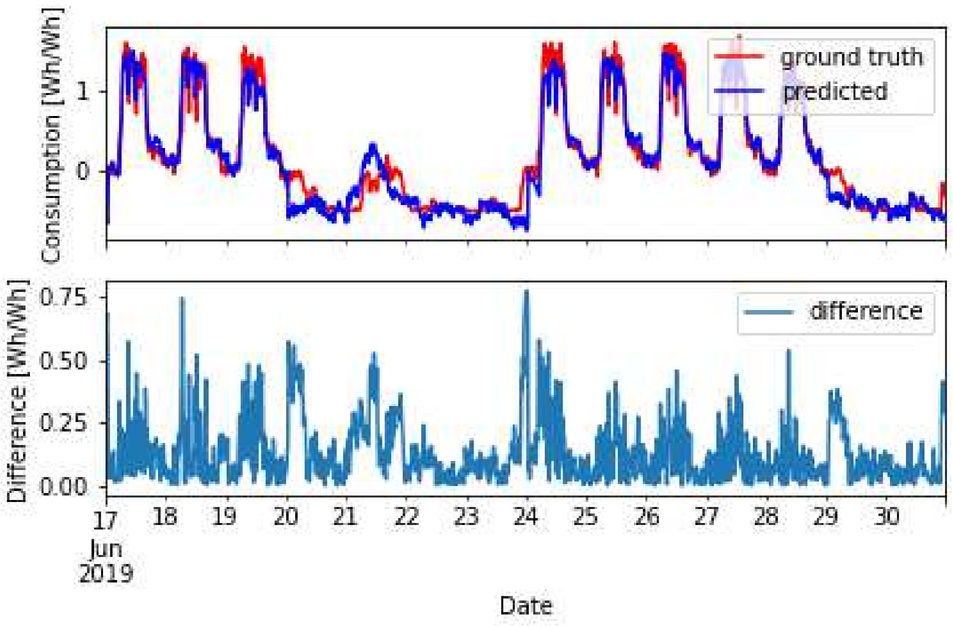

Figure 79: CNN Prediction for Dataset 3, Zoom 3 (Corpus Christi 2019)

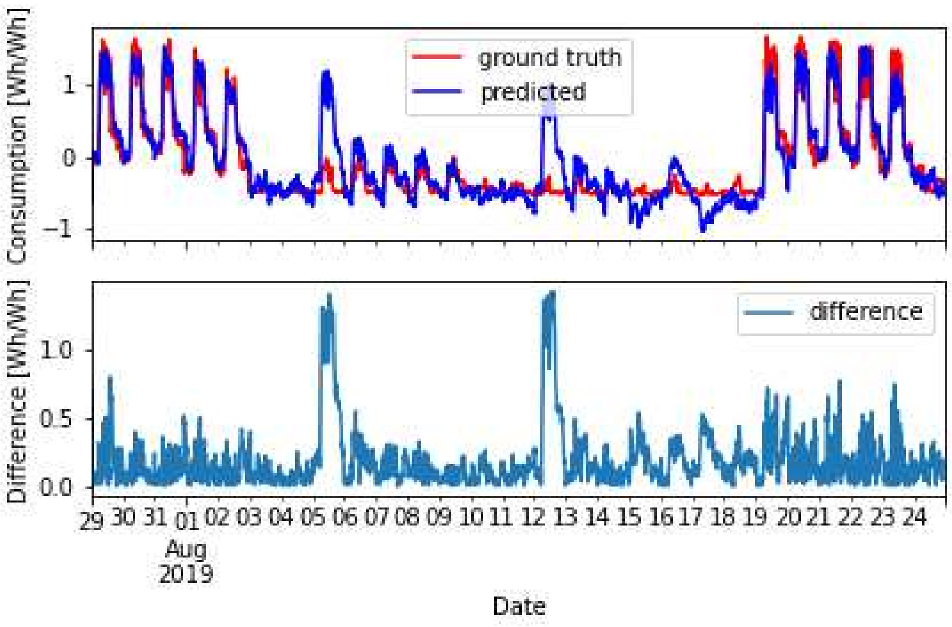

Figure 80: CNN Prediction for Dataset 3, Zoom 4 (29.07.2019 until 25.08.2019)

#### 4.3.3.4 Long-Short Term Memory

Table 13: Dataset 3 LSTM Error Statistics

| | MAE [Wh/Wh] | MSE [Wh/Wh] | RMSE [Wh/Wh] | CAE [Wh/Wh] | RMSE [%] | CAE [%] |
|---|---|---|---|---|---|---|
| Full | 0.155 | 0.053 | 0.23 | 6272 | 26 | 12 |
| Zoom 1 | 0.132 | 0.029 | 0.17 | 190 | 12 | 15 |
| Zoom 2 | 0.149 | 0.04 | 0.2 | 186 | 32 | 29 |
| Zoom 3 | 0.125 | 0.029 | 0.171 | 168 | 50 | 46 |
| Zoom 4 | 0.156 | 0.08 | 0.282 | 404 | -12 | 4 |

A total of 29 LSTM models have been trained and tested on Dataset 3, with 27 performing better than the Baseline. Together, the models achieved a mean RMSE of $0.273$ and a mean cumulative absolute error of $7217$. The selected model uses a relu activation function, three days as input data and five hidden layers, with 41 neurons each.

Overall, the model achieved an RMSE of $0.23$ and a cumulative absolute error of $6272$, improving by $26\%$ and $12\%$. Unlike to the CNN model, the LSTM model predicts very similar patterns for normal work weeks, predicting relatively constant demand levels across working days. However, the model is still outperforming the Baseline for Zoom 1, Zoom 2 and Zoom 3. Similar to the ANN and CNN model, the LSTM model has difficulties predicting if a Monday is the beginning of a holiday/maintenance period, but is able to adapt to the drop in energy demand within 24h. Having those issues, the LSTM model is achieving an RMSE of $0.282$ and a cumulative absolute error of $404$, performing by $12\%$ worse than the Baseline for the RMSE. This drop in prediction performance can be improved by providing company internal planning information to the model input data.

*Table 13* lists the performance metrics for the selected LSTM model.

#### 4.3.3.5 Summary

Dataset 3 was provided by a utility company and the true origin of the dataset is unknown. The demand pattern is highly recurring, with little volatility and the company is scheduling fixed holiday/maintenance periods. On Dataset 3, the CNN and LSTM models are performing better than baseline, with the LSTM outperforming the CNN model by $2\%$ for the RMSE and $3\%$ for the cumulative absolute error. Overall it is possible to reduce the prediction error by $26\%$ for the RMSE and $12\%$ for the cumulative absolute error, using LSTM.

All algorithms had difficulties predicting holiday/maintenance periods. Providing internal planning information as algorithm input could significantly strengthen the prediction performance.

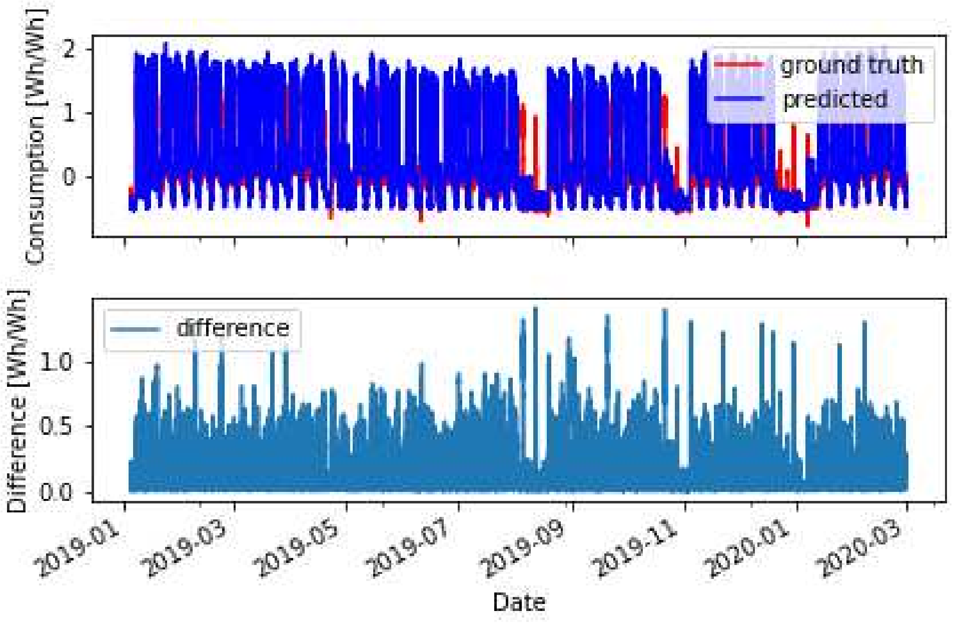

Figure 81: LSTM Prediction for Dataset 3

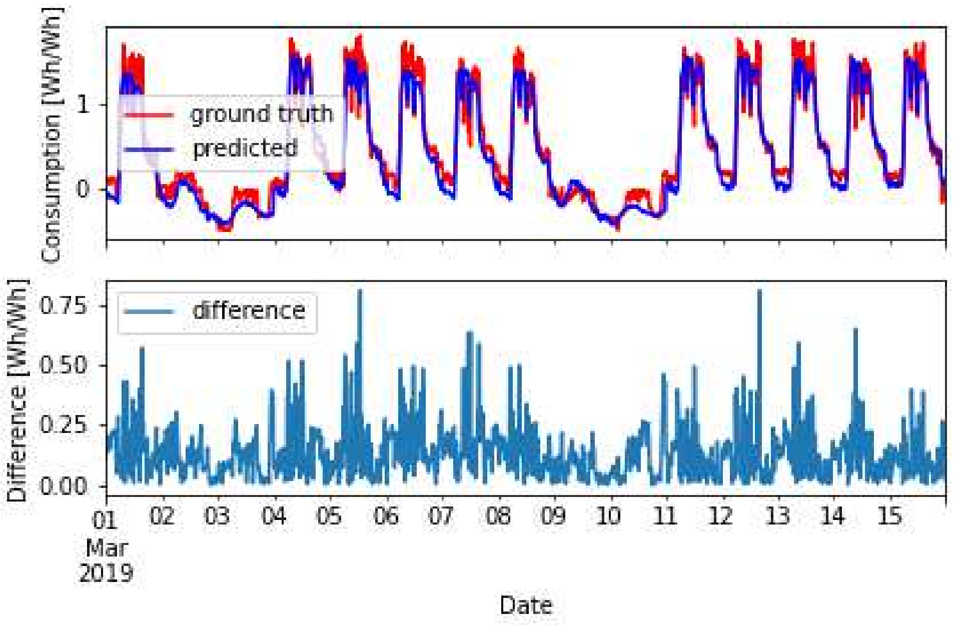

Figure 82: LSTM Prediction for Dataset 3, Zoom 1 (01.03.2019 until 16.03.2019)

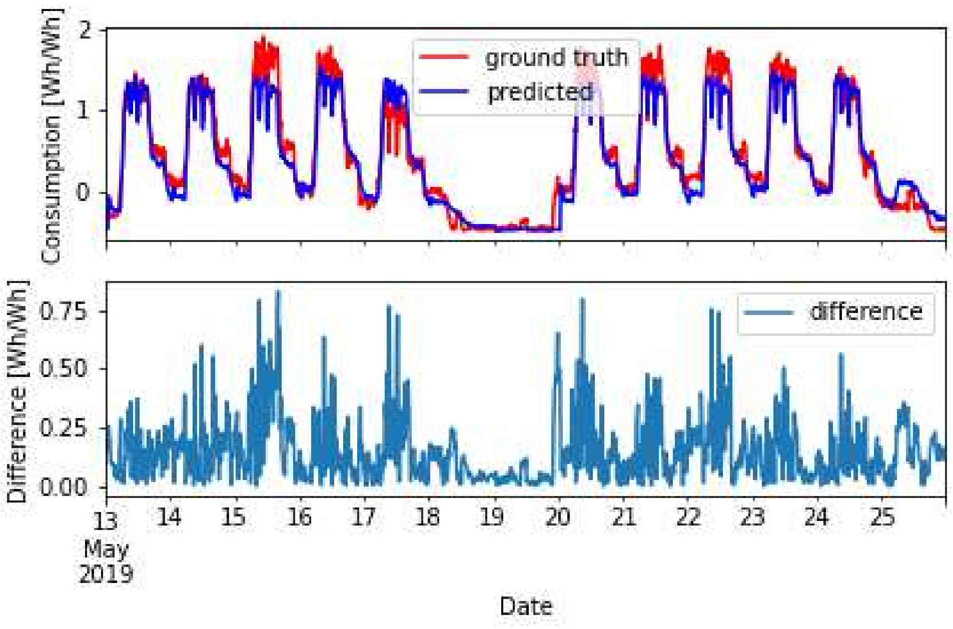

Figure 83: LSTM Prediction for Dataset 3, Zoom 2 (Whit Monday 2018)

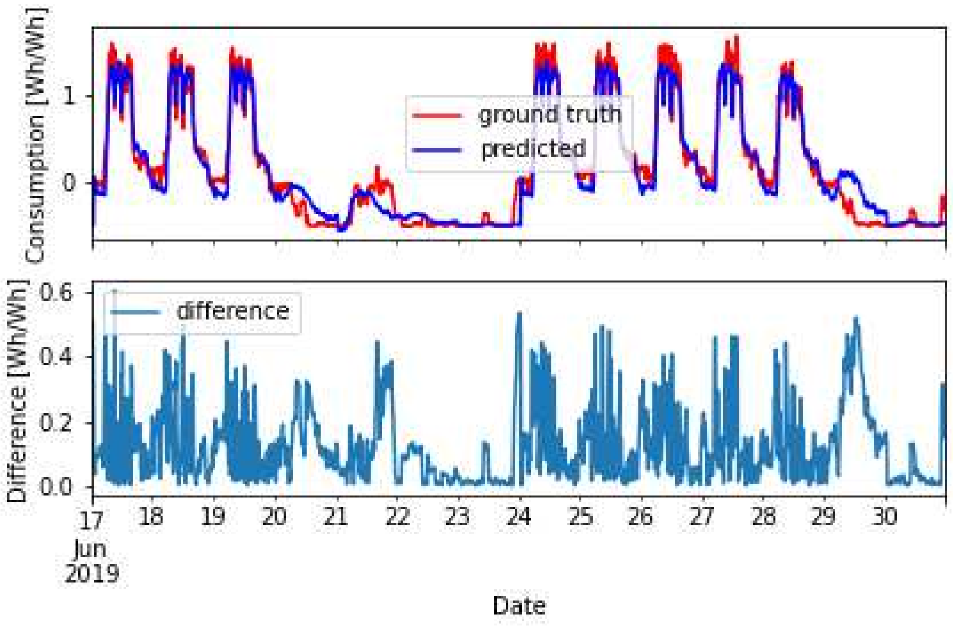

Figure 84: LSTM Prediction for Dataset 3, Zoom 3 (Corpus Christi 2019)

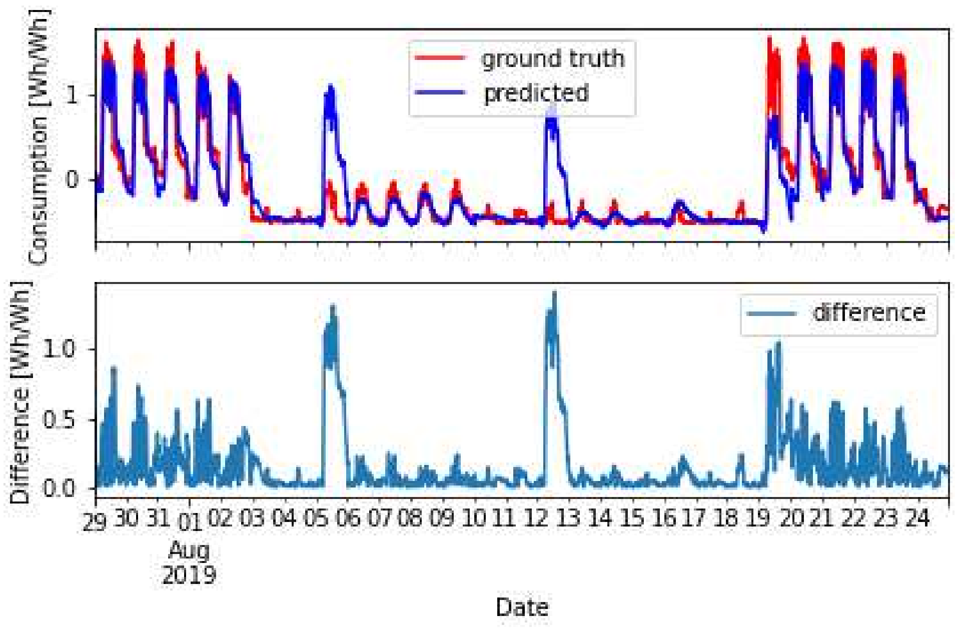

Figure 85: LSTM Prediction for Dataset 3, Zoom 4 (29.07.2019 until 25.08.2019)

# 5 Conclusion

In the context of this master thesis, three different deep learning algorithms were tested for their ability to predict industrial electricity demand one day in advance with a consumption accuracy of 15 minutes. Due to their widespread use, Artificial Neural Networks (ANN), Convolution Neural Networks (CNN) and Long-Short Term Memory Networks (LSTM) were selected for the experiments. The baseline algorithm deployed is the method used by Company B and Company C in practice, which assumes last year's consumption as a prediction for future consumption. The experiments were conducted on three industrial data sets, each of which has different consumption characteristics and covers at least a period of one year and nine months up to a period of seven years. Before the baseline algorithms or deep learning algorithms could be applied to the data, the datasets were first manually inspected to understand their properties and then unified into a coherent data structure (pre-processing). Subsequently, a separate baseline was calculated for each data source in order to be able to decide on the performance of the predictions during the deep learning experiments. To conduct the experiments efficiently, all Deep Learning specific activities were combined in a Deep Learning Pipeline, in which the power consumption data was extended by date and holiday information, the dataset was divided into a training and test set and the Deep Learning models were trained, evaluated and optimized. Normalized data were used for the training process and, to ensure comparability between the datasets, standardized data was used for the documentation within the scope of this writing.

To find the best hyperparameter configuration for each dataset and model, more than 200 models were trained and evaluated. For every dataset, LSTM models gave better results than the baseline, ANN and CNN models. The ANN and CNN models alternate in 2nd and 3rd place in terms of prediction accuracy. The level of improvement in prediction accuracy varies, depending on the consumption volatility of the dataset studied, between approximately 20% and 45% per model, compared to the baseline. When demand volatility is high, the consumption data from the previous year is a poor approximation of actual demand and this leads to a high divergence between predicted and actual demand. Dataset 1 has the highest demand volatility, and the selected LSTM model achieved a 45% RMSE improvement. The ANN model was not significantly worse, with a 42% RMSE improvement. The CNN model was "only" able to achieve an RMSE improvement of 39%. Applied to Dataset 2, the best LSTM model achieved an RMSE improvement of 27% while both the ANN model and the CNN model achieved an RMSE improvement of 25%. Dataset 3 has the highest percentage of recurring consumption patterns, so the baseline is already a passable approximation to the actual consumption. With the LSTM model selected for Dataset 3, an RMSE improvement of 26% was achieved. In this case, the CNN model selected for Dataset 3 came second, with an RMSE improvement of 24%, and none of the trained ANN models achieved better results than the baseline. Comparing the prediction improvement of the LSTM models with the second placed models, the LSTM models achieve an improvement of only 2-3%. Since LSTM models are much more expensive to train than ANN or CNN models, due to the complexity of the models, it may be preferable to use ANN or CNN models despite their poorer performance compared to LSTM models in applications requiring frequent re-training, for example when new consumption data is to be incorporated into the model quickly. If frequent re-training is not necessary, LSTM models should be chosen over ANN or CNN models.

On all datasets the algorithms had to cope with changes in demand patterns, due to bridge days, changes in production processes or order situation and unpredictable holiday/maintenance periods. Overall, the algorithms managed to adapt their forecasts quickly to a changing demand situation.

As a next step, the partner companies will be informed about the achieved results. Furthermore, the Deep Learning Pipeline allows to easily test more complex model architectures, further generalization methods or changes of the used input data.

# 6 References

Afram, Abdul et al. (2018). "Development of an Accurate Gray-Box Model of Ubiquitous Residential HVAC System for Precise Performance Prediction during Summer and Winter Seasons". In: *Energy and Buildings* 171, pp. 168–182.

Agency, European Environment (2020). *Final Energy Consumption by Sector and Fuel in Europe*. URL: `https://www.eea.europa.eu/data-and-maps/indicators/final-energy-consumption-by-sector-10/assessment` (visited on 06/26/2020).

Aladag, Cagdas Hakan et al. (2012). "A New Time Invariant Fuzzy Time Series Forecasting Method Based on Particle Swarm Optimization". In: *Applied Soft Computing* 12.10, pp. 3291–3299.

Amjady, N. (2001). "Short-Term Hourly Load Forecasting Using Time-Series Modeling with Peak Load Estimation Capability". In: *IEEE Transactions on Power Systems* 16.3, pp. 498–505.

Anbazhagan, S. and Narayanan Kumarappan (2012). "Day-Ahead Deregulated Electricity Market Price Forecasting Using Recurrent Neural Network". In: *IEEE Systems Journal* 7.4, pp. 866–872.

APG (2020). *Electricity Market*. URL: `https://www.apg.at/en/markt/strommarkt` (visited on 06/27/2020).

Bergstra, James S. et al. (2011). "Algorithms for Hyper-Parameter Optimization". In: *Advances in Neural Information Processing Systems*, pp. 2546–2554.

Bergstra, James, Dan Yamins, and David D. Cox (2013). "Hyperopt: A Python Library for Optimizing the Hyperparameters of Machine Learning Algorithms". In: *Proceedings of the 12th Python in Science Conference*. Vol. 13. Citeseer, p. 20.

Bergstra, James et al. (2015). "Hyperopt: A Python Library for Model Selection and Hyperparameter Optimization". In: *Computational Science & Discovery* 8, p. 014008.

Berthou, Thomas et al. (2014). "Development and Validation of a Gray Box Model to Predict Thermal Behavior of Occupied Office Buildings". In: *Energy and Buildings* 74, pp. 91–100.

Botchkarev, Alexei (2019). "Performance Metrics (Error Measures) in Machine Learning Regression, Forecasting and Prognostics: Properties and Typology". In: *Interdisciplinary Journal of Information, Knowledge, and Management* 14, pp. 45–79.

Bouktif, Salah et al. (2018). "Optimal Deep Learning LSTM Model for Electric Load Forecasting Using Feature Selection and Genetic Algorithm: Comparison with Machine Learning Approaches †". In: *Energies* 11.7 (7), p. 1636.

Brownlee, Jason (2014). *How To Get Baseline Results And Why They Matter*. URL: `https://machinelearningmastery.com/how-to-get-baseline-results-and-why-they-matter/` (visited on 07/13/2020).

— (2017). *How to Convert a Time Series to a Supervised Learning Problem in Python*. URL: `https://machinelearningmastery.com/convert-time-series-supervised-learning-problem-python/` (visited on 09/24/2020).

— (2018a). *A Gentle Introduction to Early Stopping to Avoid Overtraining Neural Networks*. URL: `https://machinelearningmastery.com/early-stopping-to-avoid-overtraining-neural-network-models/` (visited on 09/27/2020).

Brownlee, Jason (2018b). *How to Avoid Overfitting in Deep Learning Neural Networks*. URL: `https://machinelearningmastery.com/introduction-to-regularization-to-reduce-overfitting-and-improve-generalization-error/` (visited on 07/25/2020).

— (2018c). *How to Develop LSTM Models for Time Series Forecasting*. URL: `https://machinelearningmastery.com/how-to-develop-lstm-models-for-time-series-forecasting/` (visited on 09/25/2020).

— (2018d). *How to Improve Deep Learning Model Robustness by Adding Noise*. URL: `https://machinelearningmastery.com/how-to-improve-deep-learning-model-robustness-by-adding-noise/` (visited on 09/27/2020).

— (2018e). *How to Reduce Generalization Error With Activity Regularization in Keras*. URL: `https://machinelearningmastery.com/how-to-reduce-generalization-error-in-deep-neural-networks-with-activity-regularization-in-keras/` (visited on 09/27/2020).

— (2018f). *How to Reduce Overfitting With Dropout Regularization in Keras*. URL: `https://machinelearningmastery.com/how-to-reduce-overfitting-with-dropout-regularization-in-keras/` (visited on 09/27/2020).

— (2018g). *Use Weight Regularization to Reduce Overfitting of Deep Learning Models*. URL: `https://machinelearningmastery.com/weight-regularization-to-reduce-overfitting-of-deep-learning-models/` (visited on 09/27/2020).

— (2020). *How to Use StandardScaler and MinMaxScaler Transforms in Python*. URL: `https://machinelearningmastery.com/standardscaler-and-minmaxscaler-transforms-in-python/` (visited on 07/13/2020).

Bunge, Mario (1963). "A General Black Box Theory". In: *Philosophy of Science* 30.4, pp. 346–358. JSTOR: 186066.

Che, Zhen-Guo, Tzu-An Chiang, and Zhen-Hua Che (2011). "Feed-Forward Neural Networks Training: A Comparison between Genetic Algorithm and Back-Propagation Learning Algorithm". In: *International journal of innovative computing, information and control* 7.10, pp. 5839–5850.

Chen, Yibo and Hongwei Tan (2017). "Short-Term Prediction of Electric Demand in Building Sector via Hybrid Support Vector Regression". In: *Applied Energy* 204, pp. 1363–1374.

Connor, J.T., R.D. Martin, and L.E. Atlas (1994). "Recurrent Neural Networks and Robust Time Series Prediction". In: *IEEE Transactions on Neural Networks* 5.2, pp. 240–254.

Core, TensorFlow (2020). *tf.keras.callbacks.Callback | TensorFlow Core v2.3.0*. URL: `https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/Callback?hl=de` (visited on 09/25/2020).

Dale, Charles (1981). "Practical Experiences with Modeling and Forecasting Time Series: G.M. Jenkins, GJP Ltd". In: *Journal of Policy Modeling* 3.3, pp. 411–412.

Ding, Xiao et al. (2015). "Deep Learning for Event-Driven Stock Prediction". In: *Twenty-Fourth International Joint Conference on Artificial Intelligence*. Twenty-Fourth International Joint Conference on Artificial Intelligence.

Eseye, Abinet Tesfaye et al. (2019). "Machine Learning Based Integrated Feature Selection Approach for Improved Electricity Demand Forecasting in Decentralized Energy Systems". In: *IEEE Access* 7, pp. 91463–91475.

Ferrarini, Luca et al. (2019). “Energy Consumption Models for Residential Buildings: A Case Study”. In: *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. 2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), pp. 673–678.

Filonov, Pavel, Andrey Lavrentyev, and Artem Vorontsov (2016). *Multivariate Industrial Time Series with Cyber-Attack Simulation: Fault Detection Using an Lstm-Based Predictive Data Model*. arXiv: 1612.06676.

Fu, Rui, Zuo Zhang, and Li Li (2016). “Using LSTM and GRU Neural Network Methods for Traffic Flow Prediction”. In: *2016 31st Youth Academic Annual Conference of Chinese Association of Automation (YAC)*. IEEE, pp. 324–328.

Guresen, Erkam, Gulgun Kayakutlu, and Tugrul U. Daim (2011). “Using Artificial Neural Network Models in Stock Market Index Prediction”. In: *Expert Systems with Applications* 38.8, pp. 10389–10397.

Hagan, Martin T. and Suzanne M. Behr (1987). “The Time Series Approach to Short Term Load Forecasting”. In: *IEEE Transactions on Power Systems* 2.3, pp. 785–791.

Han, Zhongyang et al. (2016). “A Two-Stage Method for Predicting and Scheduling Energy in an Oxygen/Nitrogen System of the Steel Industry”. In: *Control Engineering Practice* 52, pp. 35–45.

Han, Zhongyang et al. (2019). “A Review of Deep Learning Models for Time Series Prediction”. In: *IEEE Sensors Journal*, pp. 1–1.

Hecht-nielsen, ROBERT (1992). “III.3 - Theory of the Backpropagation Neural Network**Based on “Nonindent” by Robert Hecht-Nielsen, Which Appeared in Proceedings of the International Joint Conference on Neural Networks 1, 593–611, June 1989. © 1989 IEEE.” In: *Neural Networks for Perception*. Ed. by Harry Wechsler. Academic Press, pp. 65–93. ISBN: 978-0-12-741252-8.

Heng, E.T.H., D. Srinivasan, and A.C. Liew (1998). “Short Term Load Forecasting Using Genetic Algorithm and Neural Networks”. In: *Proceedings of EMPD ’98. 1998 International Conference on Energy Management and Power Delivery (Cat. No.98EX137)*. Proceedings of EMPD ’98. 1998 International Conference on Energy Management and Power Delivery (Cat. No.98EX137). Vol. 2, 576–581 vol.2.

Hirasawa, K. et al. (1996). “Forward Propagation Universal Learning Network”. In: *Proceedings of International Conference on Neural Networks (ICNN’96)*. Proceedings of International Conference on Neural Networks (ICNN’96). Vol. 1, 353–358 vol.1.

Hochreiter, Sepp and Jürgen Schmidhuber (1997). “Long Short-Term Memory”. In: *Neural Computation* 9.8, pp. 1735–1780.

Hoermann, Stefan, Martin Bach, and Klaus Dietmayer (2018). “Dynamic Occupancy Grid Prediction for Urban Autonomous Driving: A Deep Learning Approach with Fully Automatic Labeling”. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. 2018 IEEE International Conference on Robotics and Automation (ICRA), pp. 2056–2063.

Holzinger, Andreas et al. (2017). *A Glass-Box Interactive Machine Learning Approach for Solving NP-Hard Problems with the Human-in-the-Loop*. arXiv: 1708.01104 [cs, stat]. URL: http://arxiv.org/abs/1708.01104 (visited on 06/29/2020).

Hong, Wei-Chiang (2009). "Chaotic Particle Swarm Optimization Algorithm in a Support Vector Regression Electric Load Forecasting Model". In: *Energy Conversion and Management* 50.1, pp. 105–117.

Hubel, D. H. and T. N. Wiesel (1959). "Receptive Fields of Single Neurones in the Cat's Striate Cortex". In: *The Journal of Physiology* 148.3, pp. 574–591. pmid: 14403679.

Kaushik, Shruti et al. (2020). "AI in Healthcare: Time-Series Forecasting Using Statistical, Neural, and Ensemble Architectures". In: *Frontiers in Big Data* 3, p. 4.

Khashei, Mehdi and Mehdi Bijari (2011). "A Novel Hybridization of Artificial Neural Networks and ARIMA Models for Time Series Forecasting". In: *Applied Soft Computing* 11.2, pp. 2664–2675.

Kourti, Theodora (2003). "Multivariate Dynamic Data Modeling for Analysis and Statistical Process Control of Batch Processes, Start-Ups and Grade Transitions". In: *Journal of Chemometrics* 17.1, pp. 93–109.

LeCun, Yann and Yoshua Bengio (1995). "Convolutional Networks for Images, Speech, and Time Series". In: *The handbook of brain theory and neural networks* 3361.10, p. 1995.

Lee, Ching-Hung, Feng-Yu Chang, and Chih-Min Lin (2014). "An Efficient Interval Type-2 Fuzzy CMAC for Chaos Time-Series Prediction and Synchronization". In: *IEEE Transactions on Cybernetics* 44.3, pp. 329–341.

Lee, K.Y., Y.T. Cha, and J.H. Park (1992). "Short-Term Load Forecasting Using an Artificial Neural Network". In: *IEEE Transactions on Power Systems* 7.1, pp. 124–132.

Li, Xiaoqing et al. (2016). "Energy Consumption Test and Analysis of Large Public Buildings Based on Gray Box Model". In: *Procedia Engineering*. The 8th International Cold Climate HVAC Conference 146, pp. 47–52.

Liao, Gwo-Ching and Ta-Peng Tsao (2006). "Application of a Fuzzy Neural Network Combined with a Chaos Genetic Algorithm and Simulated Annealing to Short-Term Load Forecasting". In: *IEEE Transactions on Evolutionary Computation* 10.3, pp. 330–340.

Liu, Bin and Geoffrey I. Webb (2010). "Generative and Discriminative Learning". In: *Encyclopedia of Machine Learning*. Ed. by Claude Sammut and Geoffrey I. Webb. Boston, MA: Springer US, pp. 454–455. ISBN: 978-0-387-30164-8.

Liu, Hui, Hong-qi Tian, and Yan-fei Li (2012). "Comparison of Two New ARIMA-ANN and ARIMA-Kalman Hybrid Methods for Wind Speed Prediction". In: *Applied Energy* 98, pp. 415–424.

Masum, Shamsul et al. (2019). "Investigation of Machine Learning Techniques in Forecasting of Blood Pressure Time Series Data". In: *Artificial Intelligence XXXVI*. Ed. by Max Bramer and Miltos Petridis. Lecture Notes in Computer Science. Cham: Springer International Publishing, pp. 269–282. ISBN: 978-3-030-34885-4.

Mellit, Adel and Alessandro Massi Pavan (2010). "A 24-h Forecast of Solar Irradiance Using Artificial Neural Network: Application for Performance Prediction of a Grid-Connected PV Plant at Trieste, Italy". In: *Solar Energy* 84.5, pp. 807–821.

Miranian, Arash and Majid Abdollahzade (2013). "Developing a Local Least-Squares Support Vector Machines-Based Neuro-Fuzzy Model for Nonlinear and Chaotic

Time Series Prediction". In: *IEEE Transactions on Neural Networks and Learning Systems* 24.2, pp. 207–218.

Mirikitani, Derrick T. and Nikolay Nikolaev (2010). "Recursive Bayesian Recurrent Neural Networks for Time-Series Modeling". In: *IEEE Transactions on Neural Networks* 21.2, pp. 262–274.

Moghram, I. and S. Rahman (1989). "Analysis and Evaluation of Five Short-Term Load Forecasting Techniques". In: *IEEE Transactions on Power Systems* 4.4, pp. 1484–1491.

Mordjaoui, Mourad et al. (2017). "Electric Load Forecasting by Using Dynamic Neural Network". In: *International Journal of Hydrogen Energy*. Special Issue on The 4th European Conference on Renewable Energy Systems (ECRES 2016), 28-31 August 2016, Istanbul, Turkey 42.28, pp. 17655–17663.

Ng, Andrew Y. and Michael I. Jordan (2002). "On Discriminative vs. Generative Classifiers: A Comparison of Logistic Regression and Naive Bayes". In: *Advances in Neural Information Processing Systems*, pp. 841–848.

Pankratz, Alan (2009). *Forecasting with Univariate Box-Jenkins Models: Concepts and Cases*. Vol. 224. John Wiley & Sons.

Parlos, Alexander G., Omar T. Rais, and Amir F. Atiya (2000). "Multi-Step-Ahead Prediction Using Dynamic Recurrent Neural Networks". In: *Neural networks* 13.7, pp. 765–786.

Paulescu, Marius et al. (2017). "Structured, Physically Inspired (Gray Box) Models versus Black Box Modeling for Forecasting the Output Power of Photovoltaic Plants". In: *Energy* 121, pp. 792–802.

Ramasso, Emmanuel, Michele Rombaut, and Noureddine Zerhouni (2012). "Joint Prediction of Continuous and Discrete States in Time-Series Based on Belief Functions". In: *IEEE transactions on cybernetics* 43.1, pp. 37–50.

Rudin, Cynthia (2019). "Stop Explaining Black Box Machine Learning Models for High Stakes Decisions and Use Interpretable Models Instead". In: *Nature Machine Intelligence* 1.5 (5), pp. 206–215.

scikit (2020). *Sklearn.Preprocessing.MinMaxScaler — Scikit-Learn 0.23.2 Documentation*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html (visited on 09/24/2020).

scikit, scalers (2020). *Compare the Effect of Different Scalers on Data with Outliers — Scikit-Learn 0.23.1 Documentation*. URL: https://scikit-learn.org/stable/auto_examples/preprocessing/plot_all_scaling.html (visited on 07/22/2020).

Shih, Han and Suchithra Rajendran (2019). *Comparison of Time Series Methods and Machine Learning Algorithms for Forecasting Taiwan Blood Services Foundation's Blood Supply*. URL: https://www.hindawi.com/journals/jhe/2019/6123745/ (visited on 06/27/2020).

stackoverflow (2020[a]). *Python - Scikit-Learn: How to Scale Back the 'y' Predicted Result*. URL: https://stackoverflow.com/questions/38058774/scikit-learn-how-to-scale-back-the-y-predicted-result (visited on 09/25/2020).

stackoverflow (2020[b]). *Python - Split Dataframe into Two on the Basis of Date*. URL: `https://stackoverflow.com/questions/37532098/split-dataframe-into-two-on-the-basis-of-date` (visited on 09/24/2020).

— (2020[c]). *Stackoverflow - Outlier Removel in Python*. URL: `https://stackoverflow.com/questions/59678724/replace-given-columns-outliers-with-mean-of-before-and-after-rows-values-in-pa` (visited on 09/24/2020).

Suganthi, L. and Anand A. Samuel (2012). "Energy Models for Demand Forecasting—A Review". In: *Renewable and Sustainable Energy Reviews* 16.2, pp. 1223–1240.

Taieb, Souhaib Ben et al. (2012). "A Review and Comparison of Strategies for Multi-Step Ahead Time Series Forecasting Based on the NN5 Forecasting Competition". In: *Expert systems with applications* 39.8, pp. 7067–7083.

Taylor, J W (2003). "Short-Term Electricity Demand Forecasting Using Double Seasonal Exponential Smoothing". In: *Journal of the Operational Research Society* 54.8, pp. 799–805.

Vaccaro, Alfredo et al. (2012). "Adaptive Local Learning Techniques for Multiple-Step-Ahead Wind Speed Forecasting". In: *Electric Power Systems Research* 83.1, pp. 129–135.

Valgaev, Oleg, Friederich Kupzog, and Hartmut Schmeck (2017). "Building Power Demand Forecasting Using K-Nearest Neighbours Model – Practical Application in Smart City Demo Aspern Project". In: *CIRED - Open Access Proceedings Journal* 2017.1, pp. 1601–1604.

Wang, Huai-zhi et al. (2017). "Deep Learning Based Ensemble Approach for Probabilistic Wind Power Forecasting". In: *Applied Energy* 188, pp. 56–70.

Wang, Sun-Chong (2003). "Artificial Neural Network". In: *Interdisciplinary Computing in Java Programming*. Ed. by Sun-Chong Wang. The Springer International Series in Engineering and Computer Science. Boston, MA: Springer US, pp. 81–100. ISBN: 978-1-4615-0377-4.

Wei, Yixuan et al. (2018). "A Review of Data-Driven Approaches for Prediction and Classification of Building Energy Consumption". In: *Renewable and Sustainable Energy Reviews* 82, pp. 1027–1047.

Werbos, P.J. (1990). "Backpropagation through Time: What It Does and How to Do It". In: *Proceedings of the IEEE* 78.10, pp. 1550–1560.

Winters, Peter R. (1960). "Forecasting Sales by Exponentially Weighted Moving Averages". In: *Management science* 6.3, pp. 324–342.

Witten, Ian H. and Eibe Frank (2002). "Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations". In: *Acm Sigmod Record* 31.1, pp. 76–77.

Wong, F. S. (1991). "Time Series Forecasting Using Backpropagation Neural Networks". In: *Neurocomputing* 2.4, pp. 147–159.

Young, Peter C. (2012). *Recursive Estimation and Time-Series Analysis: An Introduction*. Springer Science & Business Media.

Zhao, Jun et al. (2015). "Granular Model of Long-Term Prediction for Energy System in Steel Industry". In: *IEEE transactions on cybernetics* 46.2, pp. 388–400.

Zhu, Taiyu et al. (2020). "Dilated Recurrent Neural Networks for Glucose Forecasting in Type 1 Diabetes". In: *Journal of Healthcare Informatics Research*, pp. 1–17.

# A Appendix

## List of Figures

# List of Tables