UiT The Arctic University of Norway

Faculty of Science and Technology, Department of Physics and Technology

# Detecting EV Charging From Hourly Smart Meter Data

Per Harald Barkost

Master's thesis in physics - FYS-3900 - May 2020

# Abstract

Detecting electrical vehicle (EV) charging from smart meter data (EV detection) is a highly relevant problem for the distribution system operators (DSOs), especially with the expected growth of EVs world wide. There are several reasons why DSOs may want to detect EV charging. In the present day the main motivation is to reduce the total load on the grid in high demand periods. This can be achieved by giving incentives to EV owners to charge their EVs in low demand periods. In the future, it is also anticipated that EVs can act as an energy reservoir, which can be a further motivation for EV detection.

In this thesis, we explore two problems of EV detection. First, can we detect customers that charge an EV at home (EV load profiling)? Second, can we detect when an EV is charging (EV event detection)? To solve these problems, we analyze smart meter data provided by Eidsiva (a DSO from Norway).

For the problem of load profiling, we propose, a feature-based Gaussian mixture modeling of weekly load profiles. The results are promising, showing that some EV owners have unique power consumption patterns.

For the problem of event detection, we propose a modified version of UTime for EV event detection. UTime is a fully convolutional feed-forward neural network, initially proposed for sleep stage segmentation. The modified UTime is compared with previously proposed convolutional architectures for the problem of EV detection. Results show that UTime for EV detection outperforms the previous models on a generated labeled dataset.

In order to solve the problem of EV detection, a labeled data set with ground truth is crucial. Unfortunately, this is lacking in this thesis. We resolve this issue by proposing a method of generating a labeled data set by combining two data sources. Even though the method show promise and models seem to generalize for an unlabeled dataset, more verification is needed to state conclusively that our proposed method is efficient.

# Acknowledgement

I would like to give my sincere gratitude to my supervisors, Stian, Huamin, and Christoffer; without your support and guidance, this thesis would not have been achievable. Further, I would direct my appreciation to Eidsiva for providing me the necessary data. And at last, thank you, Kjersti and Ninja, for enduring in stressful times.

To everybody contributing; I appreciate your support.

- PHB

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

In this thesis, the goal is to detect EV charging at residential homes by analyzing smart meter data. Before describing the concrete problem at hand, we aim to give the reader insight into some of the motivation of why this is a relevant problem to solve, and an introduction in the different fields of smart meter analytics.

## 1.1 EVs an outlook, and its impact on the grid

In 2018 the global fleet of personal EVs (battery electrical vehicles (BEV) and plug-in hybrid electric vehicles (PHEV)) were 5.1 million units. This is a 64% increase from the previous year. There are many benefits to transition into a larger EV park, and some of them are[10]:

- The electrical motors in EVs are three to five times more efficient than conventional internal combustion engines.
- Reducing the reliance on importing fossil fuel for the road transport sector.
- Reducing air pollution since EVs have zero tailpipe emissions.
- Reducing greenhouse gas emission if electricity production is not greenhouse gas-intensive.
- EVs are quieter, reducing the noise pollution from the transport sector.

As well as practical and environmental benefits, there has been an increased focus on policy approaches to promote the deployment of EVs. An example of this is the electrical vehicle initiative (EVI) established under the Clean Energy Ministerial in 2009. EVI is dedicated to accelerating the deployment of electric vehicles worldwide. EVI tries to achieve this by enabling a platform where governmental policymakers from member countries can address and discuss challenges that come with electrical mobility[10].

A result of EVI is the 30@30 campaign launched at the Clean Energy Ministerial meeting in 2017. The goal with the campaign is that the member countries would have an EV market share 30% (excluding two and three-wheelers) by the year 2030. In addition to multi-governmental policies such as EVI, there is an increasing amount of countries that introduce policies to incentives electrification of mobility[10].

Transitioning to a larger EV fleet has benefits, but it also comes with challenges. As [10] points out, some of the challenges with increasing the EV fleet are:

- Pollution and work conditions when ramping up mining of raw materials to make batteries and motors.
- Managing the availability of raw materials when production increases.
- Recycling and reusing of batteries and motors.
- Greenhouse gas emission from EV is dependent on how the electricity is produced.
- Electrical power demand from EVs and its impact on the electrical grid becomes more relevant.

Since the electrical power demand from EVs becomes a more relevant issue. A challenge is to make sure the power delivery systems can handle this effectively, to ensure system stability. In the 30@30 estimate, most of the EV power demand comes from light-duty vehicles (<4500kg), and about 60% charging is done by slow charges that allow for flexibility in power management. Such flexibility could be;

- Controlled EV charging by reducing the load of charging in peak demand periods (and increasing it in low demand periods),
- Use EV batteries as an energy reservoir that can provide energy to either a home (vehicle-to-home) or the electrical distribution system (vehicle-to-grid)[10][11].

### 1.1.1 The EV charging situation in Norway

Norway is at the forefront of electrifying its car-park. Norway currently has the largest EV market in the world, with about 42% market share of yearly car sales in 2019[12]. Governmental policies and incentives are the principal drivers of this rapid growth. Figure 1.1 shows the growth of EVs in Norway since 2010. The number of EVs is expected to increase further since Norway's transport aims to only sell zero-emission light-duty vehicles by the year 2025. If Norway follows this projection, the number of EVs in Norway will be 1.5 million in 2030[13].

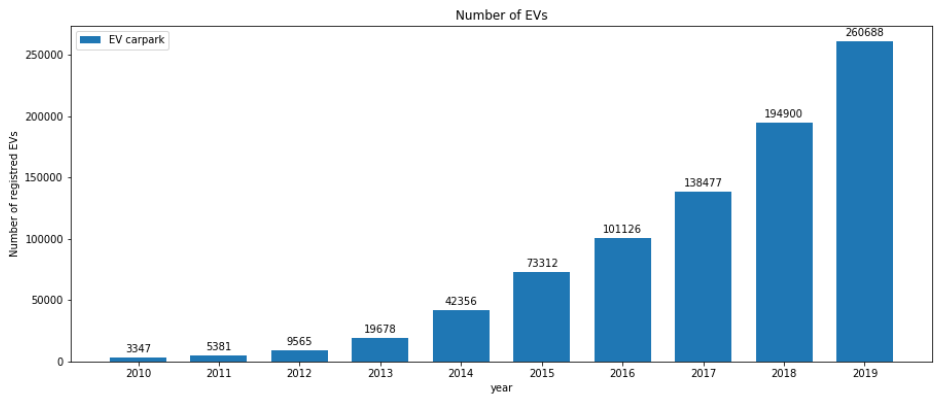


Figure 1.1: The evolution of registered EVs in Norway, from 2010 to the end of 2019.

A report by The Norwegian Water Resources and Energy Directorate investigated how the expected growth of EVs may impact the Norwegian electrical grid[13]. In the report, they estimate that 1.5 million EVs by 2030 will increase the average total electrical energy consumption by 3%. They conclude that the most of the grid infrastructure can handle this average increase. However, they raise concerns about the impact of simultaneous EV charging, in periods where the demand is already high. The added higher load if many customers charge their EV at once may result in overload and negatively impact transformers and cables in the low voltage distribution system. This concern is especially relevant in the winter season and in rural and recreational home areas where the distribution network is not built for high loads[11]. To tackle this issue, The Norwegian Water Resources and Energy Directorate suggest the use of smart meter analytics to reduce the impact of EV on the electrical grid[13].

How and when people charge their EVs is important for its impact on the grid. To investigate this further, we refer to a yearly survey from the Norwegian EV Association, where they asked how and when people charged their car. According to the survey, most people charge their EVs at home($\sim$ 90%). Table 1.1 summaries different charging options in Norway and what type of chargers people used according to the survey[5]. The survey showed that more than 50% of EV owners had installed an EV charger that is capable of drawing more power than a standard type-c wall outlet. The power column in table 1.1 indicates the maximum power available to the different type of charges. The actual power drawn is also dependent on the car model and the surrounding temperature.

Table 1.1: Type of EV charging in Norway and the percentage of how people charge their EV at home according to [5] (a survey from 2018).

| Category | Voltage/Current | Power | % charged at home |
|---|---|---|---|
| Standard type-c plug | 230V/10A | 2.3 kW | 50% |
| Slow EV charges | 230V/16A | 3.6kW | 24% |
| Semi fast EV chargers | 230V/32A | 7.4kW | 19% |
| Fast EV chargers | 400V/32A /tri-phase | 22kW | 3.5% |
| Ultra Fast EV chargers | 500V/100A | <50kW | |
| | | | 4.4% (other) |

Regarding the time of the day when people charge their EV at home, the research institution SINTEF summarized a survey from 2017 that shows that most homes reports that they charge their EV at home during the evening and night[14].

As mentioned smart meters could be the solution, to reducing the total load on the electrical grid. In the next section we aim to describe these meters, and what opportunities they enable for the DSOs.

## 1.2 Smart meters

A smart meter measures the power consumption at a household with a relatively high resolution previously not possible when electrical customers had to manually report their energy consumption for each billing period. Smart meters are essential part of the advanced metering infrastructure (AMI), where the data collected from smart meters is returned to the DSO. Some of the benefits for the customers from installing a smart meter is an automatic and more precise reading of energy consumption for billing purposes. Also making it easier to change electricity suppliers, as well as better detection of faults in the power delivery system such as ground faults.

For the DSOs, there are also significant benefits: Monitoring every single household power demand allows for higher insight into individual customers ' behaviors and their impact on the electrical grid. In the later years, several countries have done a massive roll-out of smart meters. This has spiked the interest in smart meter data analytics[15].

### 1.2.1 Smart meters in Norway and privacy concerns

Norway has decided that by 2019 all residential homes will have installed a standardized smart meter that registers both active and reactive power with a sampling rate of maximum one sample every 60 min. The smart meters should further allow for a 15 min sampling rate[16]. Currently, the Norwegian smart meters send out hourly measurements to the DSO, as well as information about short outs, ground fault, and reduced voltage quality[17]. The data collected is subject to the Norwegian Personal Data Act, meaning that measurements from smart meters is personal information and can only be stored for three years[18]. This regulation is not unique for Norway.

In general there are privacy concerns regarding smart meter data, making it difficult for energy providers to publish data to the public. This is an limitation of smart meter analytics as an open research field, since privacy of the customers are a priority.

## 1.3 Smart meter analytics

In this section, we attempt to categorize and briefly describe different fields of smart meter analytics, inspired by a review of smart meter analytics[15]. The research fields can be divided into four main categories; **Load monitoring**, **load analysis**, **load forecasting** and **load management**.

### 1.3.1 Load monitoring

Load monitoring can be divided into two main fields; Intrusive load monitoring (ILM) and none intrusive load monitoring (NILM).

**Intrusive load monitoring (ILM)** monitor's power demand at an appliance level, which means that a power meter is attached to each appliance in a household. Except for the inconvenience regarding installations, there are also privacy concerns having appliance level knowledge in a household[19].

**Non-intrusive load monitoring (NILM)** monitors the aggregated (sum of all appliances) power or current signal of out a household and in contrast to ILM, NILM does not require intruding into a household to install the sensors. Therefore the name non-intrusive. NILM is also cheaper and more convenient to implement since it only requires the installation of a single device at the main circuit board of a household. A smart meter is a type NILM device since it monitors the aggregated power signal of a household.

There are two typical use cases of NILM systems: First is to identify energy consumption of a single appliance from an aggregated signal[19]. Second, is a simpler task of event detection, which mean to determine whether a appliance is switched on or of.

Further, the field of NILM can be divided into two main approaches, supervised and unsupervised learning. One of the main challenges with supersized learning is that it requires a labeled dataset that is generally not available for smart meter data. Therefore unsupervised learning is the most attractive approach for development into business application since it does not require a ground truth. However, unsupervised methods are not easy to fully realized because of the need for verification that the implemented methods works[20]. In between supervised and unsupervised, we have a "self-learning" approach referred to as semi-supervised learning, which has also shown good practical results[21][20].

### 1.3.2 Load analysis

Customer's energy consumption and behaviors according to the weekday, time of day, season, etc. is varying. Having a better understanding and categorization of different consumption behaviors can be very important when doing further load analysis, such as forecasting and load management[15]. The categorization of consumer behaviors is often referred to as load profiling. Another important aspect of load analysis is bad-data and anomaly detection, since outliers may affect the performance of forecasting and clustering algorithms. Methods for bad data and outlier detection can also be used to detect energy theft[15].

### 1.3.3 Forcasting analysis

Load forecasting has been popular in the electrical power industry to anticipate future energy demands and pricing. Most forecasting research has been done on higher voltage signals from a region since the smoother nature of the signal is an easier task to forecast[15]. However, using additional information from smart meter data has shown it can improve the forecasting methods[15].

### 1.3.4 Load management

Load management, is balancing of the electrical supply not by adjusting the power station output, but rather controlling the power consumption. As [15] points out, there are three main ways smart meter data can contribute to better load management:

- Give the electrical provider a better understanding of customer's sociodemographic status. This can further be used to provide personalized services or anticipate customers load profiles, and energy demand.
- Target consumers with specific demand and response marketing.
- Implement demand and response programs. Such as adjusting the pricing according to the demand and incentivize customers with demand and response pricing to maximize profit or reducing the total load on the electrical grid at certain periods of the day[15].

For the task of detecting EV charging from smart meter data. The problem naturally falls under the field of load monitoring, and as discussed the main motivation is load management. Before providing a literature review of previous research into EV detection, we will present our contributions and structure of the thesis in the remainder of this chapter.

## 1.4 Contributions

We propose several new contributions to the problem of EV detection:

- A modified version of UTime for the problem of EV event detection.
- To train our supervised models, we propose generating a labeled dataset from two data sources.
- A method of EV event detection of long smart-meter sequences to reduce the number of missing values.
- Performing Gaussian mixture modeling of load profiles to capture EV owners in separate clusters.
- We propose detrending smart meter series before clustering, to remove seasonal variations.

## 1.5 Structure of the thesis

The remainder of the thesis we present according to chapter:

**Chapter 2:** Overview of the previous work regarding the problem of EV detection, and define the problems we aim to solve in this thesis.

**Chapter 3:** Present the the relevant theory for the chosen methods. This chapter is divided into two parts: Clustering and Supervised learning.

**Chapter 4:** Present the proposed methods, models, and implementations.

**Chapter 5:** Provide information about the different data sources, prepossessing, and describe how the labeled data set is generated.

**Chapter 6:** Present the clustering results, with the aim to capture distinct clusters with EV owners from load profiles.

**Chapter 7:** Experimental results and comparison of the proposed model for the task of EV event detection.

**Chapter 8:** Comparing the clustering results from Chapter 6 with the prediction of event detection in Chapter 7.

**Chapter 9:** Conclusion and further work.

# Chapter 2

# Literature review and problem definition

In the Introduction, we gave a brief overview of the different fields of smart meter analytics and explained the motivation for discovering EV charging from smart meter data. Now we will dive deeper into the problem of detecting EV charging. This chapter is divided into two main parts:

- Fist we present a literature review, of the different papers regarding EV detection (to the author knowledge).
- Second, we define the problem definition relevant for this thesis.

## 2.1 Detecting EV charging; A literature review.

Detecting EV charging from smart meter data can be viewed as a part of the NILM category of smart meter analytics. Meaning that from an aggregated power signal, we aim to either detect if an EV charge event is present, or desegregate the power signal from EV charging. Table 2.1 gives an overview of papers in the field of EV detection from smart meter data. As Table 2.1 shows, most of the papers are using the Pecan Street dataset[22].

Pecan Street has records of desegregated (appliance level) electrical consumption at a one-minute sampling rate from nearly 1000 volunteer homes in Texas, California, and Colorado in the US. In Pecan Street, some of the households charge their EV at home, and therefore it has become a popular dataset into the research of EV detection.

Table 2.1: Overview of articles (to the authors knowledge) addressing the problem of EV detection

| Title | Key words | Dataset | Sampling rate |
|---|---|---|---|
| "Automated Detection of Electric Vehicles in Hourly Smart Meter Data." | Supervised, RNN, CNN, Autoencoder, Cross-correlation filtering,stacked model | Pecan Street dataport Eidsiva | 1hr |
| "Training-free non-intrusive load monitoring of electric vehicle charging with low sampling rate." | Unsupervised, (a) Tresholding, (b) Filtering, (c) Removing noise, (d) Energy desegregation | Pecan Street dataport | 1min |
| "Extracting and Defining Flexibility of Residential Electrical Vehicle Charging Loads" | Independent component analysis (ICA) | Pecan Street dataport | 1min |
| "An improved non-intrusive load monitoring method for recognition of electric vehicle battery charging load" | Pattern recognition, Cross correlation filtering | Simulated data | - |
| "Electric vehicle charging load filtering by power signature analysis" | Unsupervised, Filtering | Pecan Street dataport | 1min |
| "Unsupervised non intrusive extraction of electrical vehicle charging load patterns" | Unsupervised, ICA | Pecan Street dataport | 1min |
| "A data-drivenapproach to identify households with plug-in electrical vehicles (pevs)" | Mining algorithms, Load profile analysis, Clustering (random forest, k-nn) | Smart meter data from Michigan (US) | Resampled to 1hr |
| "Analyzing household charging patterns of plug-in electric vehicles (pevs): A data mining approach" | Load profile analysis, feature extraction, Mining algorithms | Smart meter data from Michigan (US) | Resampled to 1hr |
| "Robust identification of ev charging profiles" | Denoising autoencoder, convolutional neural network | Pecan Street dataport | 1 min |

When detecting EV from smart meter data there are three main problems we may attempt to solve;

1. **Load profiling:** Determine whether a customer owns an EV?

2. **Event detection:** When is the EV charging?

3. **Load desegregation:** How much power is drawn from EV charging?

As for NILM, the problem of EV detection there are supervised and unsupervised load desegregation and event detection methods. The semi supervised category is not included, since to the authors knowledgde this has not yet been explored for the task of EV detection. In addition, we include data-driven approaches that is related to load analytics, which falls outside the NILM category of smart meter analytics.

In the remainder of this section, we describe the methods in Table 2.1 categorized according to whether its methods are supervised, unsupervised, or data-driven.

### 2.1.1 Unsupervised load desegregation

As Table 2.1 summarizes most of the research in the field of unsupervised methods has been done for with a sampling rate of 1 min (1/60 Hz).

Paper [23] proposes sliding window of cross correlation filtering and pattern matching in order to detect sections where EV charging is present. The validation is done on a synthetic generated dataset. Paper [6] explores a similar model as [23] and validates it by using Pecan street. From paper [6] the cross-correlation filtering is worse than the other proposed supervised models.

The two papers [24] and [25] have a very similar filtering technique. Where they both assume EV charge events draw more than 3kW of power and has a square waveform. The algorithm in [25] is described in a five-step procedure:

1. Thresholding the aggregated signal by setting values of the input signal under a certain threshold $T_{low}$ to zero. After thresholding, the signal can be divided into segments where the thresholded signal has non zero values.

2. Remove segments with a short duration compared to the surrounding segments.

3. Remove residual noise.

4. Classify the remaining segments into three categories by analyzing a cumulative counting function that counts the number of sample points above a certain value.

5. Desegregate the power drawn from EV

The major difference between [24] and [25] is that [24] removes baseline noise before the first step. Both papers use Pecan street with a 1 minute sampling-rate as validation and use a hidden Markov model as a baseline model for comparison. They both point out the major task when filtering is to distinguish EV power signals from other high power appliances such as; air conditioner, washing machine, dryer, and water heater.

The two papers [26] and [27] follows the same approach of independent component analysis (ICA)[28]. The difference is that [27], in addition to extracting electrical vehicle charging loads, also suggests a flexibility index for an aggregated EV load demand (when several households charge their vehicle at the same time).

ICA is a statistical model that assumes that the observed signal $\vec{x} = [x_1, x_2, ..., x_m]$ comes from a mixing of independent components $\vec{s} = [s_1, s_2, ..., s_m]$. The general form of ICA can be expressed as the linear relation

$$\vec{x} = \mathbf{A}\vec{s} \tag{2.1}$$

where $\mathbf{A}$ is an unknown mixing matrix of size $m \times n$. Since $\vec{x}$ is the only observed value, the problem becomes to estimate $\vec{s}$ and $\mathbf{A}$[28].

When estimating the assumption that components $s_i$ are independent and drawn from a none Gaussian distribution is made[26]. For the concrete problem of ICA for extracting EV loads from aggregated power signal, there are two mixing components; load from EV and the rest of the aggregated power signal. Further [26] and [27] simplify the problem by assuming known amplitudes for EV charging signals, meaning that one of the distributions can be assumed to be known.

The process of ICA for EV load extraction is in [26] [27] described in four main phases;

1. **Initialization**

2. **Iterative process**

    - Application of ICA.
    - Extracting the EV load vector.
    - Remove false positives
    - Estimation of EV load amplitude.

3. **Improve estimation of the extracted EV loads**

4. **Extract gradual increase and gradual decrease in the extracted EV loads**

[26] validates the their ICA method for both event detection and load desegregation with Pecan Street for different sampling rates (from 1 to 5 minutes) with declining results for higher sampling rates. However, it shows overall better performance than [25] for 1 minute sampling rate.

### 2.1.2 Supervised methods

Inspired new development of supervised learning methods in NILM [6] and [2] both utilizes artificial neural networks (also referred to as deep learning) for the task of EV detection.

Based on the workings from a master thesis[3], the article [6] proposes two neural networks: A convolutional neural network (CNN) and a recurrent neural network (RNN) as baseline model they uses a cross correlation filtering technique similar to [23]. Since the three model are able to detect unique charging instances a stacked model is proposed, that combines all three for better prediction capabilities. Table 2.2 shows the reported results. The CNN and RNN have similar over all performance however the CNN are less accurate for its most confident predictions, and the best performing model is the stacked one[6]. The RNN and CNN are trained on Pecan street resampled to 1 hour. In addition to the labels from Pecan street, synthetic square waveform charge events is added with probability of 50% at random where no charge event is present. Further [6] investigates how the stacked model preforms on unlabeled smart meter data from Norway. The predicted result is roughly in line the EVs registered in the region. As [6] points out there are three main concerns when training a model on US smart meter data and for application in Norway;

- There are differences in the electrical consumption between US and Norway. Such as the use of AC in the summertime in US and electrical heating in Norway at during the winter period.
- Type of EV used in the countries may differ. Resulting in different charging patterns for the countries.
- Percentage of EV charge events are different in the two datasets.

The supervised paper, [2], aims to desegregate EV loads by using CNN for feature extraction and dense denoising autoencoder for reconstructing the EV load signal. Before training the model; filtering and smoothing is performed on the

Table 2.2: Reported results from [6]. The RNN is a long-short term memory (LSTM) network.

| Model | Best F1 score | Average precision |
|---|---|---|
| Cross correlation filter | 0.45 | 0.40 |
| CNN | 0.67 | 0.58 |
| RNN (LSTM) | 0.67 | 0.68 |
| Stack | 0.70 | 0.71 |

input signal as well as normalization on both the input signal and labels (ground truth of EV load). The result shows that the proposed model can effectively detect start times and EV charging periods as well as generalize to other out of sample houses[2]. Pecan Street for 1 minute sampling rate is used, and the presented result is from training with only one house and validated on a different house. As future work they suggest;

- Increasing the amount of training data.
- Handling certain dips in the signal (maybe due to missing data).
- Validate model performance outside of Pecan Street. How will the model perform outside of Pecan street?

### 2.1.3 Data-mining and load analytics

The last two papers [7] [8] analyzes hourly-weekly load profiles to classify consumers as EV or no EV owners. Since smart meter data has a significant seasonal variation, three load profiles are extracted according to the time of year; winter, summer, and combining spring and fall into one. The load profiles are further processed by applying a Hampel filter to remove outliers and normalized to ensure that all feature dimensions have equal importance.

The two papers differ in the type of features that are extracted from the load profiles. [8] extract skewness and kurtosis features, while [7] extracts feature by using energy envelope and delta thresholding. Both papers compare different supervised classification algorithm: k-NN (k nearest neighbors), RFA (random forest algorithm), CART (classification and regression trees), and CHAID (chi-square automatic interaction detector). A summary of the classification accuracy is reported in Table 2.3.

The results show that RFA is the best performing classifier, and kurtosis features gave the best overall classification result. Since kurtosis and skewness have a large, peek at the weekend, only weekdays profiles are used when classifying in [8]. These proposed methods are supervised, and shows that proposed feature spaces have underlying patterns unique to people charging an EV at home.

An unsupervised approach to discovering such patterns is clustering, and have been popular method for analyzing load profiles with the aim to capture customers with similar consumption patterns[29][15][30]. Therefore, clustering is believed to be a promising approach, to discover these underlying patterns for customers that charges an EV at home.

Table 2.3: Summary of clustering results from the training set reported in paper [7] and [8].

| | Accuracy Skewness (%) | | | Accuracy Kurtosis (%) | | | Accuracy Energy Envelope and delta thresholding (%) | | |
|---|---|---|---|---|---|---|---|---|---|
| | no EV | with EV | Overal | no EV | with EV | Overal | no EV | with EV | Overal |
| k-nn | 84.23 | 88.11 | 86.8 | 87.22 | 91.23 | 89.28 | 77.52 | 81.05 | 79.28 |
| RFA | 89.23 | 95.53 | 90.25 | 93.52 | 97.65 | 94.59 | 82.87 | 87.18 | 84.95 |
| CART | 85.42 | 88.25 | 86.68 | 90.33 | 93.24 | 91.86 | 80.23 | 83.33 | 81.76 |
| CHAID | 86.65 | 90.51 | 87.25 | 89.64 | 94.81 | 91.51 | 79.23 | 84.12 | 81.42 |

## 2.2 Problem definitions

Most research into EV detection (see Table 2.1) has been done with a 1 minute sampling rate. This relatively high sampling rate, is seldom available for the DSOs due to privacy concerns and storage capacity. Some research has utilized smart meter data from energy companies, and they have had a sampling rate of 1 hour.

Summarizing the papers for an hourly sampling rate, deep learning methods is suggested for event detection, and load profiling and classification algorithms is suggested for the problem of load profiling[6][7][8]. Where all previous methods have been supervised. Inspired by previous work, we aim to explore two problems of EV detection:

1. The problem of EV load profiling: "Do a customer charge an EV at home?"

2. The problem of EV event detection: "When is an EV charging?"

The motivation for exploring both problems is due to the limitations of the available datasets, we lack ground truths (see Chapter 5).

In the remaining sections of this chapter, we aim to provide a clear definition of these two problems.

### 2.2.1 EV load profiling

The nature of smart-meter data is noisy and often inconsistent due to missing values or varying sampling rates. One solution to tackle these inconsistencies is to derive what is referred to as load profiles, which means to transform raw smart meter data into hourly usage[7]. For this thesis, we will use what we refer to as **weekly-hourly load profiles**, which summarises the consumption at each hour of the week starting on Monday at 00:00.

By extracting load profiles, each customer has a feature vector with a fixed length that can be used for further analysis. One of the benefits of extracting load profiles is that we are able to capture the general trend of customers consumption behaviors, giving us a more smooth series. Further, we aim to use these extracted load profiles to cluster whether a customer charges an EV at home.

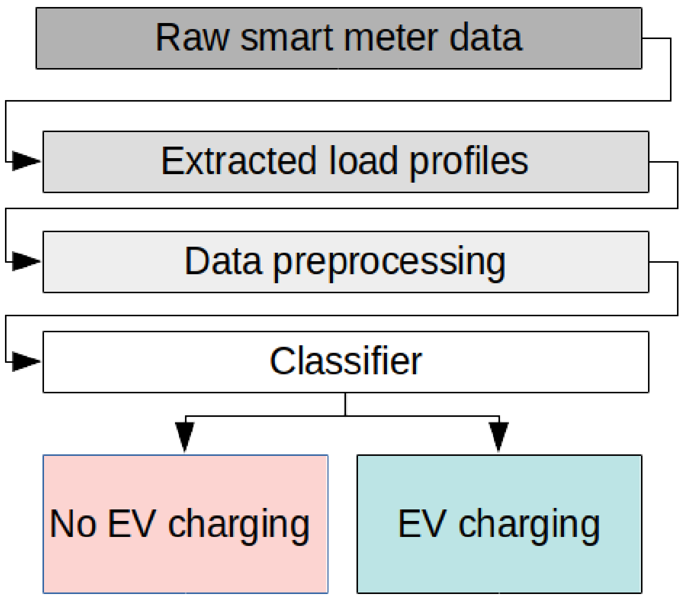


Figure 2.1: Overview of the problem of EV load profiling.

To extract profiles with the aim to classify customers that charges an EV at home is what we define as **EV load profiling**. A overview of the process of EV load profiling is shown in Figure 2.1

#### Challenges

A significant concern when extracting load profiles is that the general trend of the data is highly seasonal. With the trend of higher electrical energy consumption during the winter periods. This trend should be considered when extracting load profiles. Since, at each hour of the week, the consumption becomes highly varying because of the seasonal trend.

In this thesis, we propose detrending the data before extracting load profiles. However, other options such as extracting load profiles during certain seasons is also an option[7][8].

### 2.2.2 EV event detection

If we have an aggregated smart meter sequence of length $N$ the discrete smart meter measurements (with unit kWh) can be written as a real vector

$$\boldsymbol{x} = [x_1, x_2, ..., x_N]. \tag{2.2}$$

For the problem of event detection the goal is to determine whether at each time point $x_i$ there is an EV charging or not. Figure 2.2 shows an example of a smart meter sequence where ground truth

$$\boldsymbol{y} = [y_1, y_2, ..., y_N] \tag{2.3}$$

of EV charge events is categorized with the value 1 when an EV is charging and 0 elsewhere. The positive labels in $\boldsymbol{y}$ can be refered to as EV activations.

The proposed models for event detection $f$ takes an observation $\boldsymbol{x}$ as input and returns predictions as an output $\hat{\boldsymbol{y}}$ and it can be formulated as

$$f(\mathbf{x}) \rightarrow \hat{\boldsymbol{y}} \tag{2.4}$$

where $\boldsymbol{x}, \hat{\boldsymbol{y}} \in \mathbb{R}^N$. Meaning for each sample point there is a prediction, this model type is often refereed to as a sequence to sequence model.

#### Challenges

The main challenge with EV event detection, is to miss classify other high power appliances as EV charging. In residential homes, there is a wide variety of EV chargers available. These can be further categories according to what maximum power they can draw as shown in Table 1.1.

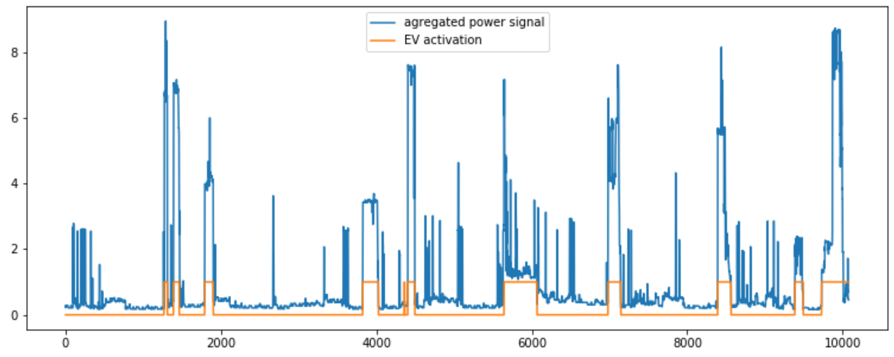


Figure 2.2: Example of EV event detection from smart meter data. The sampling rate is 1 minute (1/60 Hz), and the series has a duration for one weeks. This example series is generated from ACN+UKDALE (see Chapter 5).

We would expect it is a more difficult task to detect EV charge events with lower power consumption since these signatures may overlap with other appliances. Table 2.4 shows some common appliances that may have a similar maximum power output as EV charging. The table shows that detecting charge events with lower power consumption is a more difficult task.

Table 2.4: Typical appliances and their power rating. Source [9]

| Appliance | Average power rating (Watts) |
|---|---|
| Immersion heater | 3000 |
| Kettle | 3000 |
| Tumble Dryer | 2000-3000 |
| Oven | 2000-2200 |
| Hairdryer | 2000 |
| Oil-filled radiator | 1500-2500 |
| Washing machine | 1200-3000 |
| Dishwasher | 1050–1500 |

## 2.3 Summary

In this chapter we have provided an overview of the different papers addressing the problem of EV detection. From the previous research, we saw that for hourly smart meter data there are two main problem that is attractive to solve. First the problem of EV load profiling, and second the problem of EV event detection. In the next chapter we aim to present the relevant theory for our proposed solution to these two problems.

# Chapter 3

# Theory

We aim explore two problems of EV detection. The first problem is the problem of EV load profiling, and second the problem of EV event detection with supervised deep learning. Therefore this theory chapter is divided into two parts:

In the first section, we present the relevant theory for **time series clustering** that we use to explore the problem of EV load profiling. We have chosen an clustering approach since we aim to discover underlying patterns for customers that owns an EV.

In the second section, we describe the relevant theory for our approach of EV event detection, which is **supervised learning** by using deep learning frameworks. This is the main focus of this thesis. The reasoning of this approach is because this is a similar approach to previous work, and that currently deep learning is current state-of-the in the field of EV event detection from hourly smart meter data.

## 3.1 Clustering

Previously work of EV load profiling has used supervised algorithms for classifying customers as EV owners. However, for this thesis, the data provided has no ground truth where we, with certainty, can say a customer charges their EV at home. The information we have is whether they own an EV or not.

Due to these weak labels, we have chosen an approach of unsupervised clustering to capture some EV owners in distinct clusters. To the author's knowledge, this has not yet been attempted before this thesis. The motivation for including this approach is to further validate our results of EV event detection by comparing the clustering results with the results from EV event detection.

Our approach of clustering smart meter data is feature based, and the applied clustering algorithm is Gaussian mixture modeling. These approaches will be further explained in this section.

### 3.1.1 Time series clustering

The problem of **Time series clustering** can be formulated as follows: Given an data set containing $N$ time series $D = \{T_1, T_2, ..., T_N\}$ the aim to partition $D$ into $C = \{C_1, C_2, ..., C_K\}$ clusters. Where the grouping is done by a pre-defined similarity measure. By definition an observation can not be assign to several clusters. Mathematically this can be written as $D = \cup_{i=1}^{K} C_i$ and $C_i \cap C_j = \varnothing$ when $i \neq j$[31].

For this project we assume that the time series is continuous real values represented as real vector with length $l_i$ for time series $i$, meaning the time series might have varying lengths with only one value for each time stamp. In other words the time series are a single channel one dimensional temporal signals with varying lengths.

Time series data is nature chronological, meaning we have observations in sequence as a function of time. A result of this sequentiality is that time series often has a high dimensional and are large in data size. High dimensions are often an issue when clustering because of the computational cost when applying conventional clustering algorithms. Another issue is the potential varying lengths of the time series in $D$ makes defining a similarity measure difficult[31]. There are several approaches suggested to address these issues, and they should be chosen according to the problem we aim to solve.

**Feature-based clustering** methods is when raw time series is transformed into a feature vector in a lower dimension such that conventional clustering algorithms may be applied. Figure 3.1 summarises the steps of feature based clustering. This type of approach has been popular for clustering smart meter data with the aim to cluster customers with similar underlying patterns in energy consumption[29][15][30]. Further, similar feature-based methods have proved to work well in a supervised manner for classifying customers with EV charging[7][8].

The feature extraction mapping $h$ is done for each time series $T_i$ in the dataset $D$, such that each time series transformed into and feature vector $x_i$ with same length $d$

$$T_i \in \mathbb{R}^{l_i} \xrightarrow{h} x_i \in \mathbb{R}^d. \tag{3.1}$$

Where the $x_i$'s if further inputted in the proposed clustering algorithm.

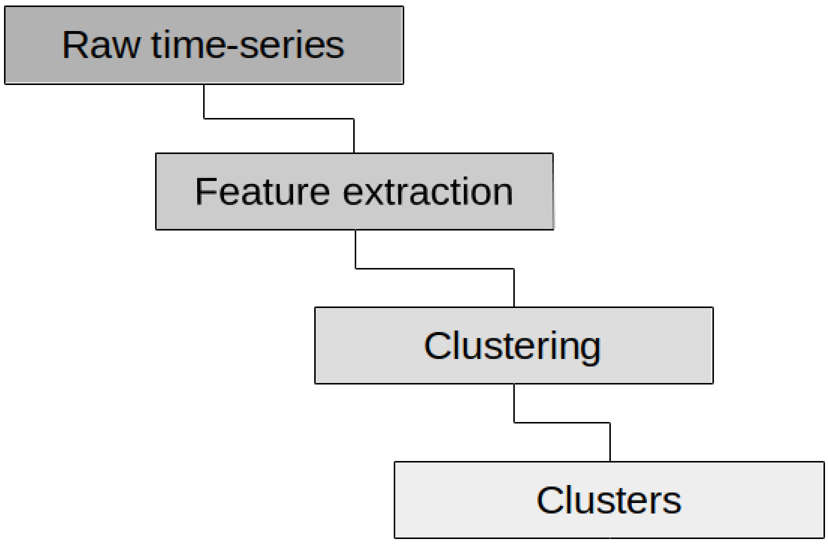


Figure 3.1: Step-by-step overview of feature based clustering which has been an popular method for clustering smart meter data.

### 3.1.2 Gaussian mixture modelling (GMM)

Mixture modeling is a tool for density estimation where we assume that observations is drawn from a mixture of probability distributions. For a Gaussian mixture model we assume that the data is drawn $K$ normal distributions, it can be formulated as

$$p(x;\theta) = \sum_{K=1}^{K} \pi_k f(x;\mu_k,\Sigma_k) \tag{3.2}$$

where $K$ is the number of mixing components and $\pi_k$ is referred to as the mixing proportion with the constrain $\sum_{k=1}^{K} \pi_k = 1$ and $0 \leq \pi_k \leq 1$. The function $f$ is the probability density function (pdf) of a multivariate normal distribution, with mean vectors $\mu_k$ and covariance matrices $\Sigma_k$ for each component k [32]. $f$ can be defined as

$$f(x;\mu_k,\Sigma_k) = f(x;\theta_k) = \frac{\exp[-\frac{1}{2}(x-\mu_k)^T \Sigma_k^{-1}(x-\mu_k)]}{\sqrt{(2\pi)^d |\Sigma_k|}} \tag{3.3}$$

where $x$ is real $d$ dimensional observation vector and $|\Sigma_k| = \det(\Sigma_k)$ (the determinate of $\Sigma_k$) and $\Sigma_k$ is assumed to be positive semi definite[33].

A mixture model also provide a confidence score $\hat{c}_{(i,k)}$ that observation $x_i$ belongs to the distribution $k$

$$\hat{c}_{(i,k)} = \frac{\pi_k f(x_i; \mu_k, \Sigma_k)}{\sum_{k=1}^{K} \pi_k f(x_i; \mu_k, \Sigma_k)}. \tag{3.4}$$

Further hard cluster assignment can be further be derived by choosing the most likely component which can be derived from Bayes theorem[32].

### Likelihood functions

The parameter's of Gaussian mixture model $\theta_k = \{\alpha_k, \mu_k, \Sigma_k\}$ for $k = 1, 2, .., K$ needs to be optimized. The aim is to maximize the incomplete likely hood

$$L(\theta|X) = \prod_{i=1}^{N} p(x_i; \theta). \tag{3.5}$$

where there are $X = \{x_1, x_2, ..., x_N\}$ observations. However the inner sum in $p$ makes optimization of $L(\theta|X)$ difficult. Therefore we introduce a new variable latent variable $Z = \{z_1, z_2, ..., z_K\}$ that indicates which component $X$ is sampled from, meaning $z_{k,i} = 1$ if sample $x_i$ comes component $k$ and zero otherwise. The distribution of latent variable can be expressed in term of the mixing proportion

$$p(z_k = 1) = \alpha_k \tag{3.6}$$

and

$$p(Z) = \prod_{k=1}^{K} \alpha_k^{z_k} \tag{3.7}$$

By introducing $Z$ we can express the conditional distribution as

$$p(x|z; \theta) = \prod_{k=1}^{K} f(x_i; \theta_k)^{z_k} \tag{3.8}$$

and the complete likelihood as

$$L(\theta|X, Z) = \prod_{i=1}^{N} p(x_i|z_i; \theta) \cdot p(z_i) = \prod_{i=1}^{N} \prod_{k=1}^{K} [\pi_k \cdot f(x_i|\theta_k)]^{z_k} \tag{3.9}$$

Further we maximize the complete log-likelihood

$$l(\theta|X, Z) = \ln(L(\theta|X, Z)) = \sum_{n=1}^{N} \sum_{k=1}^{K} z_{k,n}(\ln[f(x_n|z_k; \theta)] + \ln[\pi_k]). \tag{3.10}$$

which is equivalent to maximizing the complete likelihood[32].

**Optimization by EM algorithm**

A popular choice of mixture model optimization is the so called Expectation Maximization (EM) algorithm. The EM algorithm can be described in four steps:

1. **Initialize** parameters $\theta^{(t)}$ when $t = 0$ .
2. **E-step:** Compute expectation of the complete log likelihood given current parameters $\theta^{(t)}$: $Q(\theta|\theta^{(t)}) = E[l(\theta|X, Z, \theta)|X, \theta^t]$
3. **M-step**: Update parameters according to the computed expectation values in the E-step. (Normally by solving $\frac{\partial Q(\theta|\theta^{(t)})}{\partial\theta} = 0$ with respect to the parameter's $\theta$).
4. Continue E-step followed by M-step until convergence.

For the case of a mixture model the procedure for deriving the $Q$ is

$$Q(\theta|\theta^{(t)}) = E[l(\theta|X, \theta)|X, \theta^t] \tag{3.11}$$

$$Q(\theta|\theta^{(t)}) = \sum_{n=1}^{N}\sum_{k=1}^{K} \hat{c}_{(i,k)}(\ln[f(x_n|z_k;\theta)] + \ln[\pi_k]) - \lambda(\sum_{k=1}^{K}\pi_k - 1). \tag{3.12}$$

Since $\pi_k$ is constrained ($\sum \pi_k = 1$) Lagrange multiplier of $\lambda(\sum_{k=1}^{K} \pi_k - 1)$ is added.

When maximizing the partial derivative of $Q$ is computed with respect to parameters $\pi_k$, $\mu_k$ and $\Sigma_k$ and solved with respect to the parameters when set to zero

$$\frac{\partial Q(\theta|\theta^{(t)})}{\partial\theta} = 0. \tag{3.13}$$

The resulting parameter update scheme for Gaussian mixture becomes

$$\pi_k = \frac{\sum_{n=1}^{N} \hat{c}_{(i,k)}}{N} \tag{3.14}$$

$$\mu_k = \frac{\sum_{n=1}^{N} \hat{c}_{(i,k)} x_n}{\sum_{n=1}^{N} \hat{c}_{(i,k)}} \tag{3.15}$$

$$\Sigma_k = \frac{\sum_{n=1}^{N} \hat{c}_{(i,k)}(x_n - \mu_k)(x_n - \mu_k)^T}{\sum_{n=1}^{N} \hat{c}_{(i,k)}} \tag{3.16}$$

The EM algorithm is fast and monotone but does not guarantee convergence to a global optimum. Therefore when optimizing, the algorithm should be run for multiple initialization to ensure convergence the possible optimum[33].

The number of mixture components also needs to be pre-defined, which is not a straight forward task to determine, further discussion about this will be presented in the next section.

### 3.1.3 Model selection

If we do not have any ground truths, the suitable number of cluster $K_{opt}$ is typically a data-driven approach to derive. A traditional approach is to examine within-cluster dissimilarity $D_K$. We can obtain $D_K$ for a range for number of components $K \in \{1, 2, ..., K_{\max}\}$, resulting in $\{D_1, D_2, ..., D_{K_{\max}}\}$ dissimilarity scoring. These dissimilarity measures will be often be decreasing when we increase the number of components $K$, as the components fill the feature space.

Therefore, choosing the lowest dissimilarity may not best in practice. If we assume the data comes from a discrete number of natural underlying distributions $K_{\text{Nat}}$. We would expect the dissimilarity scoring $D_K$ for cluster $K > K_{Nat}$ to be less rapid when we further segment the natural underlying distributions. This can be used to derive a sufficient amount of components[32].

In this thesis, the derivation of the number of sufficient clusters is done in an exploratory way to capture clusters with a high concentration of EV owners. This clustering is highly biased and the reasoning and discussion about this will be presented in Chapter 6.

## 3.2 Supervised learning

Based on previous research of EV detection from hourly smart meter data, we have chosen an approach of supervised learning. Meaning we aim to utilize the ground truths to improve the predictive models.

Unfortunately, the widely popular Pecan Street is not available for this project. Therefore, we suggest a new approach by synthesizing a labeled dataset. A similar approach has been attempted earlier by adding square waveforms as charge events[3][6]. However, we try to improve upon this idea by adding real EV charge events from commercial charging stations with the hope this will generate more realistic charging events than square waveforms.

In the remainder of this chapter describe of supervised learning, as well as the main components of the proposed deep learning models, will be presented.

### 3.2.1 Defining supervised learning

For supervised deep learning, we want to approximate a function $f(\boldsymbol{x}_i)$ that takes $\boldsymbol{x}_i$ as input and outputs predictions $\hat{\boldsymbol{y}}_i$. Further, we aim to improve the predictive power of $f$ by comparing its outputs $\hat{\boldsymbol{y}}_i$ with true labels $\boldsymbol{y}_i$. This process of improving models by comparing the outputs with the ground truths is known as "learning by example"[32].

In this thesis, the signal from a smart-meter is the input vector $\boldsymbol{x}$, and whether an EV is charging or not the elements of $\boldsymbol{y}$ is either 1 or 0. The predictive output $\hat{\boldsymbol{y}}$ is a has elements that represent a confidence score between 0 and 1, which indicates the confidence the label is 1 (a positive prediction).

The final $y_{prediction}$ prediction of whether a measuring point $x_i$ is an positive event can be written as

$$y_{prediction} = \begin{cases} 1, & \text{if } \rho < \hat{y}_i. \\ 0, & \text{otherwise.} \end{cases} \tag{3.17}$$

where $0 < \rho < 1$ is a certain threshold, usually equal to 0.5 by default. This problem is a binary classification/segmentation problem.

### 3.2.2 Validation of unbalanced data

The problem at hand is a binary classification/segmentation problem where we want to predict if an EV charge event is either present or not. We may also assume that the events we want to segment are few and far between, meaning that there are a lot more none EV charge events than charging events. This assumptions leads to an imbalanced dataset.

For imbalanced data, an accuracy score, defined by the ratio number of correct predictions divided by the total number of predictions, can be misleading since we may have a high accuracy score with zero positive predictions.

An example of this if we have time-series of one day (24 hours) with 3 hours of EV charging. The accuracy of a model that could not detect any charge event would still be $\frac{21}{24} = 0.875$ a seemingly high accuracy score but a with poor performance. This example demonstrates that other metrics are important when measuring model performance on imbalanced data.

**The metrics:** precision, recall, and f1 score have been popular choices when evaluating the performance of EV event detection[6][26]. Receiver operating characteristic (ROC) will also be presented since its a popular method for comparing classifiers. However, please note that it has its weaknesses when evaluating on an unbalanced data[34].

Description of precision, recall, f1 score, and ROC is is presented in Chapter 7.

### 3.2.3 Overfitting and generalization

The goal is to have a model that generalizes well, meaning that it shows good predictive power on unseen data. A simple way to evaluate generalization is to divide the dataset into train, validation, and test set.

The model should be optimized on the train set and evaluated and tuned with respect to its performance on the validation set. After training and tuning, the last evaluation is performed on the unseen test set. It is essential with no "peeking" into the test set before the final training is done to account for bias when tuning with respect to the validation set. Figure 3.2 shows a scenario of over-fitting, where the model parameters are iterative being updated. Overfitting occurs at the moment the test(/validation) error starts to increase while the training error continues to decrease. The black vertical line shows where the overfitting begins.

### 3.2.4 Supervised deep learning

Based on the new development of deep learning for time series segmentation and that previously proposed supervised methods also uses artificial neural networks

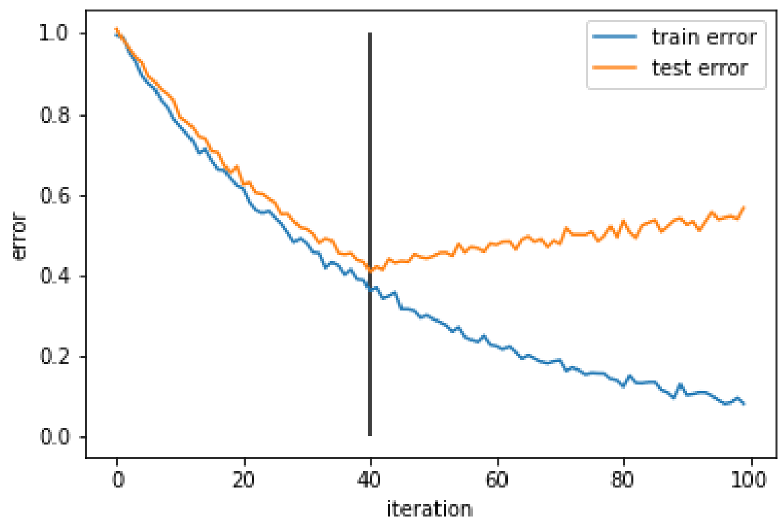


Figure 3.2: The figure shows how over fitting may look like in an supervised training scenario. The black vertical indicates where the over-fitting begins.

(ANN), we aim to use deep learning methods for the task of EV event detection.

Deep learning is an umbrella term for different ANN architectures. ANN is inspired by the receptors in the human brain and has proven to work well for a variety of different classification tasks. Even though ANN first was introduced in the 80's ANN has gotten serious attention in the later years. Some of the reasons for this are improvements in computer hardware, more massive datasets, and newly proposed groundbreaking network architectures.

In this section, we aim to give a general overview of the workings of supervised deep learning with an emphasis on how trainable parameters inside an ANN is optimized.

Initially we will describe any deep learning model as a differentiable nonlinear statistical model

$$\mathbf{x} \rightarrow f(\mathbf{x}; \theta) \rightarrow \hat{\boldsymbol{y}} \tag{3.18}$$

that performs non linear feature mapping of input features $\mathbf{x}$ into a prediction space where $\theta = \{\theta_w, \theta_b\}$ is the trainable parameters of the model that is divided into bias terms $\theta_b$ and weights $\theta_w$. This simple explanation is sufficient to give overview of how deep learning models learns by stochastic gradient descent based methods. Later we will provide a more detailed insight into the implemented model components.

Before explaining gradient descent-based methods, we need to define a loss function that we wish to minimize. The optimization of a loss function is analogous to maximum likelihood estimation. However, we wish to minimize the loss function rather than maximizing the likelihood.

### 3.2.5 Lossfunctions

A supervised loss function can is defined as a differentiable error function $L$ that evaluates the predictive output $\hat{\boldsymbol{y}}$ of model $f$ with the true labels $\mathbf{y}$. The choice of a loss function is task dependent and requires consideration. For our problem which is a binary classification problem where we assume imbalance there are several options that is worth considering and some of them are:

#### Mean square error loss

For a binary classifier the mean square error loss can be defined as

$$L_{MSE} = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2 \tag{3.19}$$

#### Binary cross entropy

Binary cross entropy loss be defined as

$$L_{BCE} = \frac{1}{N}\sum_{i=1}^{N} -w_i[y_i \log \hat{y}_i + (1 - y_i)\log(1 - \hat{y}_i)] \tag{3.20}$$

where $w_i$ is a manual rescaling weight given to each element in order to emphasise certain predictions.

#### Dice-loss

The Dice similarity coefficient (DSC) also known as F1 score measures the amount of agreement between two regions. When applied to binary classification data and using confusion metrics of true positives (TP), true negatives (TN), false positives (FP), false negatives (FN) the DSC can be defined as

$$\text{DSC} = \frac{2\text{TP}}{2\text{TP} + \text{FP} + FN}. \tag{3.21}$$

DSC is not differentiable and can not be used as a loss function to compute gradients.

For a binary classification the differentiable Dice loss $L_{DSC}$ which we wish to minimize can be defined as

$$L_{DSC} = 1 - 2\frac{\sum_{i=1}^{N} \hat{y_i} y_i}{\sum_{i=1}^{N} \hat{y_i} + y_i}[35]. \tag{3.22}$$

Earlier research into EV event detection, the mean square error loss has been used for load desegregation, and equally-weighted binary cross-entropy loss has been used for event detection[3][2].

In this thesis, we propose the Dice loss as which is a differentiable version of F1 score. It would be interesting to explore the performance of Dice loss since F1 score is our primary performance metric when evaluating.

As previously mentioned, when training, we wish to optimize the parameters of our statistical model by minimizing the loss function, and this is done by what is called gradient descent-based methods.

### 3.2.6 Optimization

Deep learning is usually done by gradient descent-based methods. Even though gradient descent-based methods do not guarantees converge to global minima. It has been proven empirically that gradient-based methods work well for deep learning models. In this section, optimization methods by gradient descent are described. Further details about computing gradients by backpropagation will be provided in the sections about the different model components.

#### Gradient descent

If we have a predictive function $f(x;\theta) = \hat{y}$ that aims to approximate a true output vector $y$ from the input feature vector $x$. $\theta = [\theta_1, \theta_2, ..., \theta_n]$ represent the tuneable parameters of function $f$. The error of the output can be computed by a loss function $L$.

Gradient descent aims to minimize $L$ by iterative update the parameters $\theta$ such that it minimizes the loss function $L$. The updating procedure for a parameter $\theta_i$ can be written as

$$\theta_i \leftarrow \theta_i - \lambda \frac{\partial L}{\partial \theta_i}. \tag{3.23}$$

Where $\lambda$ determines how much the parameter is beeing "moved" for each iteration, the "direction" is determined by computing the partial derivative $\frac{\partial L}{\partial \theta_i}$[36][37].

Equation 3.23 shows the standard gradient descent methods, several improvements to this optimization methods has been proposed such as gradient descent with momentum, which have the update scheme:

$$\theta_i \leftarrow \theta_i + \lambda v \tag{3.24}$$

where

$$v \leftarrow \rho v - \lambda \frac{\partial L}{\partial \theta_i} \tag{3.25}$$

For gradient descent with momentum, the parameter is updated by adding a velocity term $v$. $v$ is computed from the previous gradients, and $\rho$ is referred to as the momentum parameter. The $\rho$ parameter dictates how much the velocity from the previous iteration is going to contribute to the current update[38].

For gradient descent with momentum, all parameters have the same updating scheme. ADAM optimizer allows for individual parameter updates by introducing first and second-order momentum[39].

### ADAM optimizer

ADAM optimizer computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients[39][37]. The optimization algorithm is updates the parameters as follows:

1. Set parameters $\beta_1$, $\beta_2$ and $\epsilon$. $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 1 \times 10^{-8}$ is good default settings[39].

2. Initialize first and second order moment:

    - $m = 0$ (first order momentum)
    - $v = 0$ (second order momentum)

3. for t in (1:$N_{iterations}$ ):

    (a) Update the first and second order momentum:

    - $m \leftarrow (\beta_1 m + (1 - \beta_1)\frac{\partial L}{\partial \theta_i})/(1 - \beta_1^t)$ (first order momentum)
    - $v \leftarrow (\beta_2 v + (1 - \beta_2)(\frac{\partial L}{\partial \theta_i})^2)/(1 - \beta_2^t)$ (second order momentum)

    (b) Update the parameters by:

    - $\theta_i \leftarrow \theta_i - \lambda \frac{m}{\sqrt{v}+\epsilon}$

### Stochastic learning

For deep learning tasks, the dataset is usually huge, making it very expensive to compute predictions and gradients for the complete training set. One method to overcome this is so-called stochastic learning, which means to divide the training set into mini-batches and for each mini-batch update the parameters[36][37].

With stochastic learning, we must determine the size of the mini-batches. A good practice is to choose that each batch approximates the whole training set while also being computationally efficient. In stochastic learning, usually, validation is done after the entire training set has passed trough the model, also known as an epoch.

### Regularization

To avoid over fitting adding a regularisation penalty $L_{reg}$ to the loss function can significantly improve generalization of a model

$$L_{total} = L_{task} + \alpha L_{reg} \tag{3.26}$$

since the regularization term $L_{reg}$ is a function with respect to the parameter weights $\theta_w$ that penalties weights with high values, forcing parameters to have a lower values. $\alpha$ is the regularization penalty coefficient that need to be tuned, if chosen to large the models are prone to under fit. Common choices of regularisation penalties the $L1$ and $L2$ norm defined as

$$L_{L1} = ||\theta_w||_1 = \sum_{j=1}^{N_w} |\theta_{w_i}| \tag{3.27}$$

$$L_{L2} = ||\theta_w||_2 = \sum_{j=1}^{N_w} \theta_{w_i}^2 \tag{3.28}$$

In this section, we have given a brief introduction to how deep learning models are optimized and that adding a regularization penalty can give better generalization. In the next sections, we will provide descriptions of the implemented deep learning components and how gradients of parameters are being computed by backpropagation.

## 3.3 Deep learning model components

So far we have given an description, of supervised learning and how supervised deep learning frameworks are being trained by gradient descent. In this section we aim to describe the two types of model components used which are:

- Dense neural networks
- 1 dimensional Convolution neural networks (1D CNN)

### 3.3.1 Dense Neural networks classifier

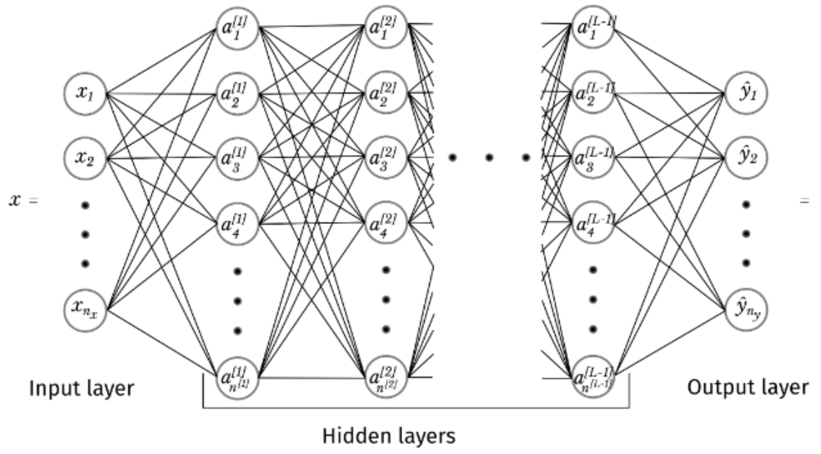


Figure 3.3: A dense neural network.

A dense neural network can be represented as a network diagram as shown in Figure 3.3. Figure 3.3 shows a sketch of a dense neural network classifier with L layers and an $n^{[l]}$ nodes in layer l. It takes a input as a flatten feature vector $\vec{x}$ then passed forward to nodes in the hidden layers. At each node there is a function that can be defined as

$$a^{[l]}_k = g(\sum_{j=1}^{n^{[l-1]}} w^l_{jk} a^{[l-1]}_j + b^{[l]}_k) \tag{3.29}$$

where $g$ is an non linear activation function, $w^l_{jk}$ is a weight (shown as an edge in Figure 3.3), $a^{[l-1]}_j$ is activation's from the previous layer, and a bias term $b^{[l]}_k$. This function can be vectorised for a entire hidden layer l as a linear combination $z^{[l]}$ inside an activation function $g$

$$z^{[l]} = (W^{[l]})^T a^{[l-1]} + b^{[l]} \tag{3.30}$$

$$a^{[l]} = g(z^{[l]}) \tag{3.31}$$

where $W^{[l]}$ is a weight matrix for layer $l$, $b^{[l]}$ is a bias vector, and $a^{[l-1]}$ is activation vector from previous layer. In the final layer $L$ the activation function are normally removed or adapted to fit the desired output. At the final output layer the computations can be written as

$$z^{[L]} = (W^{[L]})^T a^{[L-1]} + b^{[L]} \tag{3.32}$$

$$a^{[L]} = s(z^{[L]}) = \hat{y} \tag{3.33}$$

where $s$ is the modified activation function. $z^{[L]}$ are often referred to as logits, and the output values $\hat{y}$ is predicted output confidence scores. For a binary classification case function $s$ is often chosen to be the sigmoid function defined as

$$s(x) = \frac{1}{1+e^{-x}}. \tag{3.34}$$

**Back propagation**

As mentioned the parameters in a neural network needs to be optimized by gradient descent based methods. In this section we aim to derive how gradients of weights and bias term of dense neural network is calculated by back propagation.

To update the parameters we need to calculate the partial derivative of the loss function with respect to the weight and bias terms, meaning we aim to compute

$$\frac{\partial L}{\partial w_{jk}^{[l]}} \quad \text{and} \quad \frac{\partial L}{\partial b_k^{[l]}} \tag{3.35}$$

for all layers and nodes.

Since a neural network is a series of linear combinations followed by none linear activation the gradients in a dense layer can be computed by using the chain rule

$$\frac{\partial L}{\partial w_{jk}^{[l]}} = a_j^{[l-1]} \frac{\partial L}{\partial z_k^{[l]}} \quad \text{and} \quad \frac{\partial L}{\partial b_k^{[l]}} = \frac{\partial L}{\partial z_k^{[l]}} \tag{3.36}$$

where

$$\frac{\partial L}{\partial z_k^{[l]}} = g'(z_k^{[l]}) \sum_{i=1}^{n^{[l+1]}} \frac{\partial L}{\partial z_j^{[l+1]}} w_{kj}^{[l+1]} \qquad \text{for} \quad l = 1,..,L-1 \tag{3.37}$$

and for the final layer L we need to calculate

$$\frac{\partial L}{\partial z_k^{[L]}} = \sum_{j=1}^{N_{out}} \frac{\partial L}{\partial a_j^{[L]}} \cdot \frac{\partial a_j^{[L]}}{\partial z_k^{[L]}} \tag{3.38}$$

From the above equation you may see that the output layer needs specific handling according to the chosen loss function however the layers from $L-1$ to 1 are only dependent of the layer in front and the activation function. This allow us to back propagate meaning updating the weights starting at the output layer, moving backwards to the input layer[40].

## Vectorization

Since dense layer are mainly a linear combination followed by a nonlinear activation ANN can be vectorized with several inputs at once. Meaning that we can derive forward pass and propagation of a input matrix $X = [x_1, x_2, ..., x_m]$ with shape $n_x \times m$.

In the forward pass the linear combination $Z^{[l]}$ and activation $A^{[l]}$ at layer $l$ can be written as follows

$$Z^{[l]} = W^{[l]T} A^{[l-1]} + B^{[l]} \qquad \text{and} \qquad A^{[l]} = g(Z^{[l]}) \tag{3.39}$$

where $Z^{[l]}$, $A^{[l]}$, $B^{[l]}$ is of shape $n^{[l]} \times m$ and $W^{[l]}$ is of shape $n^{[l-1]} \times n^l$. As an example the gradients when back propagating can be expressed as

$$\nabla_{W^{[l]}} C = \frac{1}{m} A^{[l-1]} \mathbf{J}_{z^{[l]}}(C)^T \tag{3.40}$$

$$\nabla_{b^{[l]}} C = \frac{1}{m} \mathbf{J}_{z^{[l]}}(C) \mathbf{1}(m) \tag{3.41}$$

$$\mathbf{J}_{z^{[l]}}(C) = g'(Z^{[l]}) \circ (W^{[l+1]} \mathbf{J}_{z^{[l+1]}}(C)) \tag{3.42}$$

$$\mathbf{J}_{z^{[L]}}(C) = \hat{Y} - Y \tag{3.43}$$

where $\mathbf{J}_{z^{[L]}}(C)$ is the Jacobian of vectorized cross entropy loss with respect to $z^{[l]}$, $\mathbf{1}(m)$ is an $m$ dimensional column vector with value ones and $\circ$ represent the operation of element vise multiplication (Hadamard product)[41]. The vectorized cross entropy loss $C$ can be defined as

$$C = -\frac{1}{m} \mathbf{1}(n_y)^T (\hat{Y} \circ \log Y) \mathbf{1}(m) \tag{3.44}$$

The purpose of showing this example is to show that dense neural networks allow for an array of input vectors. Further, the forward pass and backpropagation can be significantly be sped up by parallel high-performance computing.

### 3.3.2 Activation functions

The activation functions $g$ has traditionally been the sigmoid or tanh function but has been less popular later years since its gradients vanish for large and small input values. This is an issue since parameters require gradients to be updated by gradient descent.

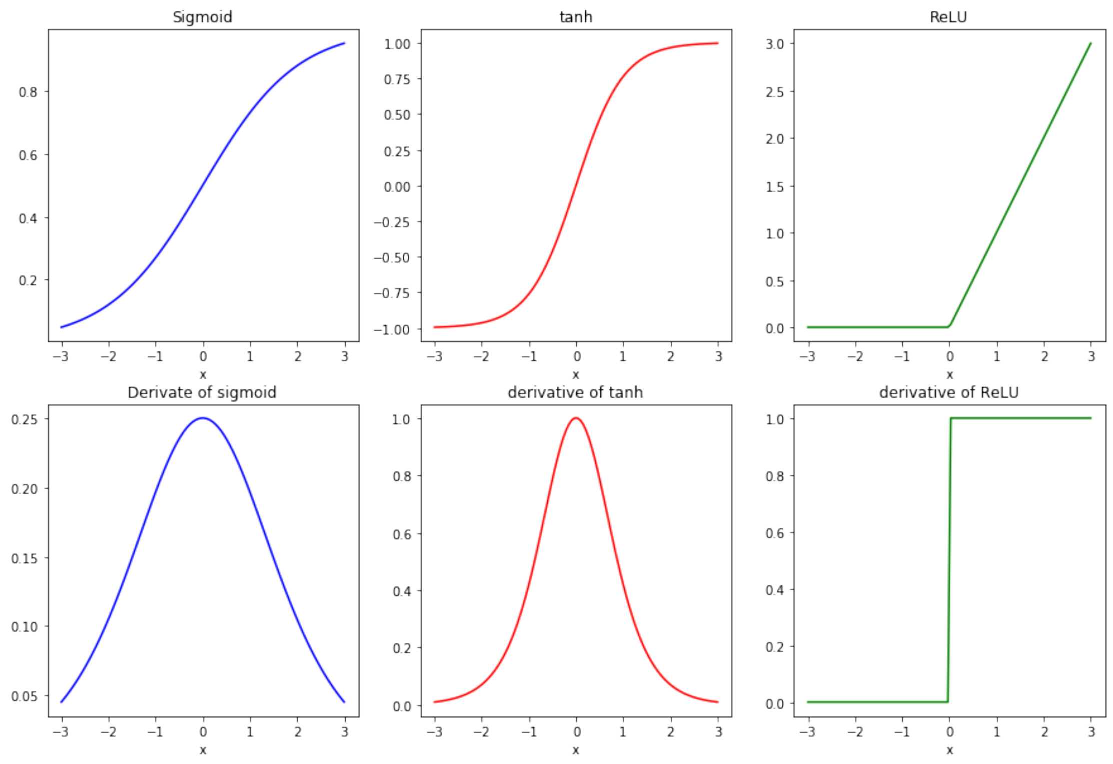


Figure 3.4: Plots of popular activation function and its derivative.

The choice of activation functions is still an area of research and in the later years rectified linear unit (ReLU) function

$$\text{ReLU}(z) = max\{0, z\} \tag{3.45}$$

has become a more popular choice[42]. Even though the ReLU function is not differentiable at $x = 0$, it is seldom an issue in real-life applications. Figure 3.4 shows the different activation functions and their derivatives.

ReLU has been shown in practice to yield better results. Another issue with ReLu is that the gradient is zero for negative input values, meaning the problem of vanishing gradients is not completely resolved. As an alternative to ReLU, the leaky ReLU tackles this issue further to some degree[42].

### 3.3.3 1D convolution neural networks (CNN)

Even though Convolutional neural networks (CNN) have existed for over 30 years, the later years CNN has gotten a lot of traction because of their groundbreaking performance on image data[43][40]. One of the breakthroughs in convolution neural networks in modern days was AlexNet proposed by Krizhevsky in 2012 that out preformed traditional machine learning methods[44]. Another aspect of why CNN has become increasingly popular is because we are now able to train massive data set on much faster computer hardware. Which solves one of the early drawbacks with neural networks; that optimization of millions of parameters is computation expensive. Especially graphical processor units (GPUs) and parallelization have significantly increased the speed of optimization when training on large datasets; this is also a major reason for why CNN today has become so popular.

Most of the research into CNN has done for large datasets with multichannel two dimensional (2D) images. However, recent research into one dimensional (1D) CNN has gotten more attention. As [40] points out, one of the major drawbacks when training a deep CNN (many hidden layers) is that CNN requires a massive data set to achieve reasonable generalization. For many 1D dimensional applications, this may not be viable since the size of a labeled dataset is generally smaller, and therefore deep 1D CNN can result in over fitting and poor generalization.

#### 1D convolution layer

The traditional architecture of a CNN consists of stacked convolutional layers with max pooling in between. After the final convolutional layer, the output is flattened and passed through a dense (fully connected) layers. Figure 3.5 show an example of such architecture, and its details will be explained in this section.

The computations trough a 1d convolutional filter $d$ at layer $l$ can be written as

$$a^{[l]}_{j,d} = g(x^{[l]}_{j,d}) = g((y^{[l-1]} * K^{[l]}_{d})_{j,d} + b^{[l]}_{d}) \tag{3.46}$$

Where $g$ is the activation function $y^{[l-1]} \in \mathbb{R}^{H_{in} \times C}$ is the one dimensional input

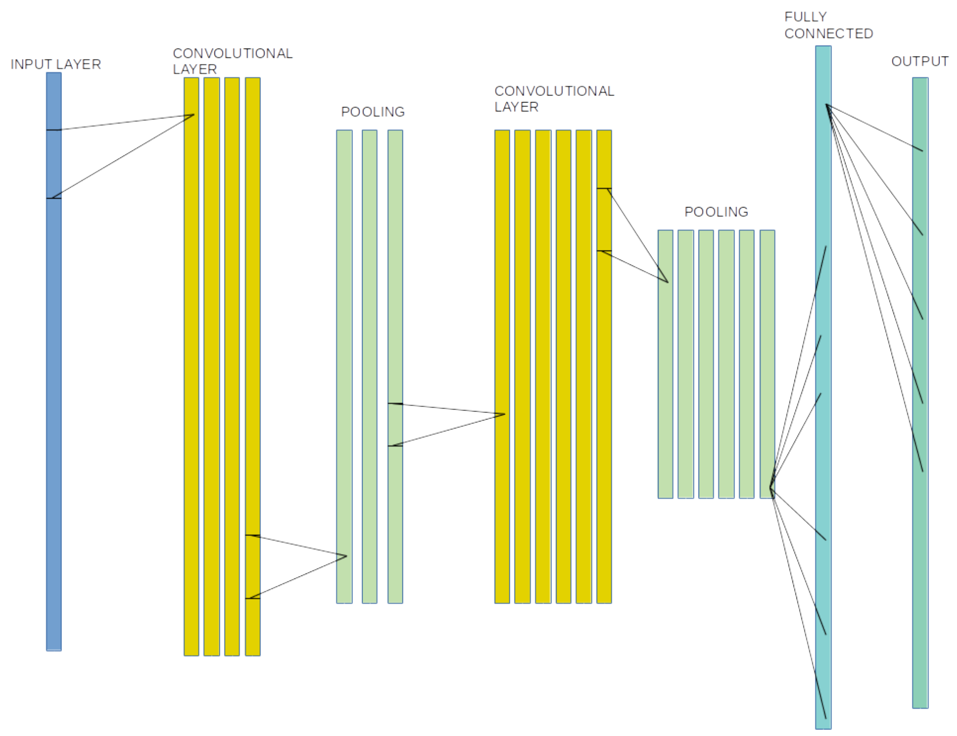


Figure 3.5: Figure of a traditional 1D CNN. With one channel 1D data as input with two convolutional layers followed by a dense (fully connected ) layer.

with length $H$ and $C$ channels, the filter kernel $K^{[l]} \in \mathbb{R}^{k \times C \times D}$ with kernel size of k, and $D$ filters has element wise weights $w^{[l]}_{m',c,d}$ and bias term $b \in \mathbb{R}^{D}$ at layer l where $m' \in \{1,..,k\}$, $c \in \{1,..,C\}$ and $d \in \{1,..,D\}$. The convolutional operation $x^{[l]}_{j,d}$ of $j$th element of $d$ filter can be can be written as

$$x^{[l]}_{j,d} = (y^{[l-1]} * K^{[l]})_{j,d} + b^{[l]}_{d} \tag{3.47}$$

where

$$(y^{[l-1]} * K^{[l]})_{j,d} = \sum_{c=1}^{C} \sum_{m=1}^{k} w^{[l]}_{m,c,d} \cdot y^{[l-1]}_{i+m,c} \tag{3.48}$$

and $x^{[l]}_{j,d} \in \mathbb{R}^{H_{out} \times D}$, with the output length is $H_{out} = H_{in} - k + 1$. In Figure 3.5 and 3.6 there is a down sampling pooling layer after convolutional layer this common approach in CNNs to reduce the spatial size, and parameters in the network. Mathematically the down sampling can be expressed with the down sampling operation $D_{\downarrow}$ resulting the final input into the next convolutional layer to be

$$y^{[l]}_{j,d} = D_{\downarrow}(a^{[l]}_{j,d}). \tag{3.49}$$

In Figure 3.5, the last layers are fully connected/dense layers in practice; the output from the last pooling layer is flattened as an input to a dense model.

Figure 3.6 shows an example of a convolutional layer with one filter kernel with a kernel size of 3, resulting in one output channel. The figure also points out additional aspects like padding, stride, and dilation.

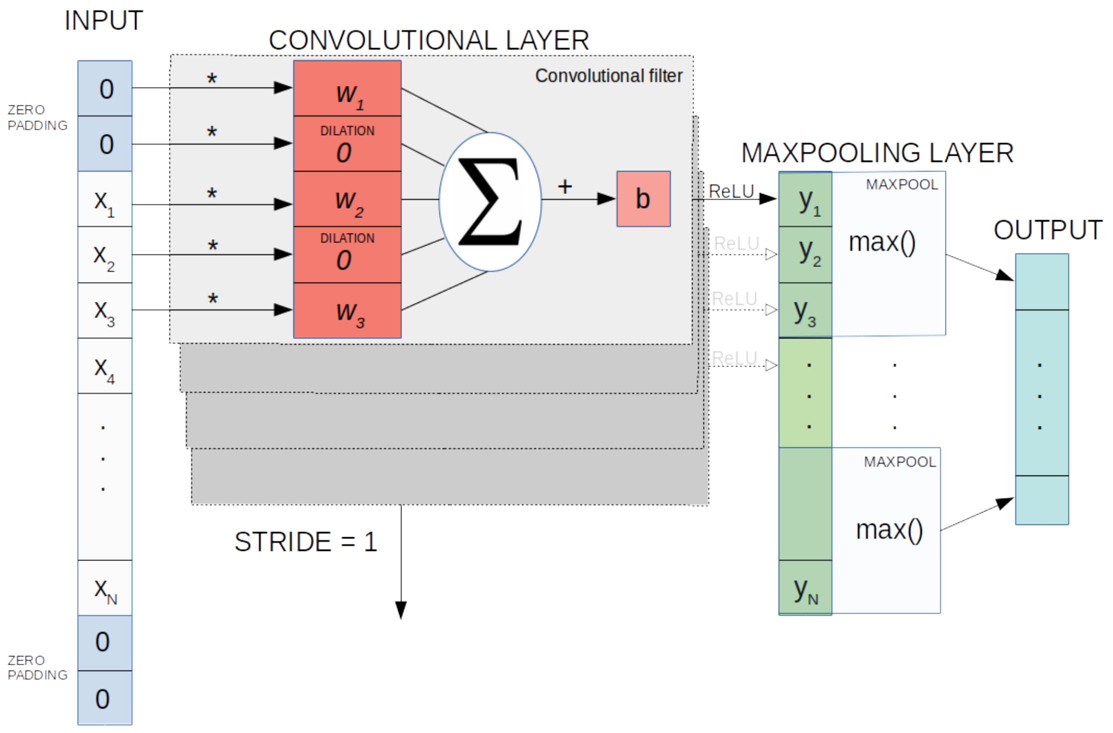


Figure 3.6: Example of a 1D convolutional layer followed by a ReLu activation and maxpooling. The convolutional layer has a kernelsize size of 3 and one filter, dilation and stride is both 1. The two zero padding keeps the output the same size as the input. The maxpooling layer is a down sampling layer where the maximum value in the window is returned. In the figure the pooling window is 3.

**Stride:** Stride is the step length for the convolutional operation. In Figure 3.6, the stride is 1, meaning it the kernel moves one step at the time. Stride is a hyperparameter that heavily determine the output size.

**Padding:** When preforming convolutional operation the output may have lower dimension than the input size. A solution to this, is to add padding to the edges of the input vector. A common choice is to add zero padding (as in Figure 3.6). The output size $H_{out}$ for stride $S$, filter size $F$, input size $H_{in}$ and padding

$P$ can be calculated by the formula

$$N_{out} = \frac{N_{in} - F + 2P}{S} + 1$$

if $S = 1$ we can achieve same $H_{out} = H_{in}$ by choosing the padding to be

$$P = (F - 1)/2 \tag{3.50}$$

assuming that the results is divisible.

**Kernel size:** At each convolutional layer, several filters can be applied. The output then gets an added depth for each filter added. In implementations of CNN this the number of filters at each convolutional layer is often referred to as output channels. The size of a filter is referred to as kernel size. For a 1D CNN the kernel size is the length of a vector $w$ that the 1D input is convolved with.

**Dilation:** Dilation is skipping inside values in the kernel, allowing to have a kernel that covers a larger area without increasing the number of parameters. If choosing a proper stride, the layer may cover all inputs, as shown in Figure 3.8.

### Back propagating

Similar as for dense neural networks computation of gradients of parameters in convolutional layers is done by backpropgation. Taking the the CNN architecture proposed in Figure 3.5 as an example the first last dense layer is computed the same way as for dense neural networks (Section 3.3.1). Further we need to compute the gradients of parameters int the convolutional layers with respect to the loss function $L$

$$\frac{\partial L}{\partial w^{[l]}_{m',c,d}} \quad \text{and} \quad \frac{\partial L}{\partial b^{[l]}_{d}} \tag{3.51}$$

For simplicity in notation we assume that the input has one channel $C = 1$ and the filter kernel has $D = 1$ filters and the down sampler is linear such that $y^{[l]}_j = a^{[l]}_j$. By using the chain rule of the derivatives the derivatives can be written as

$$\frac{\partial L}{\partial w^{[l]}_{m'}} = \sum_{i=1}^{H_{out}} \frac{\partial L}{\partial x^{[l]}_i} \frac{\partial x^{[l]}_i}{\partial w^{[l]}_{m'}} = \sum_{i=1}^{H_{out}} \Delta^{[l]}_i \frac{\partial x^{[l]}_i}{\partial w^{[l]}_{m'}} \tag{3.52}$$

and

$$\frac{\partial L}{\partial b^{[l]}} = \sum_{i=1}^{H_{out}} \frac{\partial L}{\partial x^{[l]}_i} \frac{\partial x^{[l]}_i}{\partial b^{[l]}} = \sum_{i=1}^{H_{out}} \Delta^{[l]}_i \frac{\partial x^{[l]}_i}{\partial b^{[l]}} \tag{3.53}$$

where $\frac{\partial L}{\partial x_i^{[l]}} = \Delta_i^{[l]}$. Further computation of $\Delta_i^{[l]}$, $\frac{\partial x_i^{[l]}}{\partial w_{m'}^{[l]}}$ and $\frac{\partial x_i^{[l]}}{\partial b^{[l]}}$ is required. The result are as following

$$\Delta_{i'}^{[l]} = \sum_{m=1}^{k} \Delta_{i'-m}^{[l+1]} \frac{\partial x_{i'-m}^{[l+1]}}{\partial x_{i'}^{[l]}} \tag{3.54}$$

where we can compute $\frac{\partial x_{i'-m}^{[l+1]}}{\partial x_{i'}^{[l]}}$, $\frac{\partial x_i^{[l]}}{\partial w_{m'}^{[l]}}$ and $\frac{\partial x_i^{[l]}}{\partial b^{[l]}}$ can be differentiate equation 3.47 where we get a contribution of the derivatives where $m' = m$. The result becomes

$$\frac{\partial x_{i'-m}^{[l+1]}}{\partial x_{i'}^{[l]}} = w_m^{[l+1]} g'(x_{i'}^{[l]}) \tag{3.55}$$

$$\frac{\partial x_i^{[l]}}{\partial w_{m'}^{[l]}} = y_{m'}^{[l-1]} \tag{3.56}$$

$$\frac{\partial x_i^{[l]}}{\partial b^{[l]}} = 1 \tag{3.57}$$

Inserting these results into equation 3.54, 3.52 and 3.53 the result of the derivatives becomes

$$\frac{\partial L}{\partial w_{m'}^{[l]}} = \sum_{i=1}^{H_{out}} \Delta_i^{[l]} w_m^{[l+1]} g'(x_{i'}^{[l]}) \tag{3.58}$$

$$\frac{\partial x_i^{[l]}}{\partial b^{[l]}} = \sum_{i=1}^{H_{out}} \Delta_i^{[l]} \tag{3.59}$$

where

$$\Delta_{i'}^{[l]} = \frac{\partial L}{\partial x_i^{[l]}} = \sum_{m=1}^{k} \Delta_{i'-m}^{[l+1]} w_m^{[l+1]} g'(x_{i'}^{[l]}) \tag{3.60}$$

$\Delta_{i'}^{[l]}$ and $\frac{\partial L}{\partial w_{m'}^{[l]}}$ can be as a convolutional operation with zero padding and $\text{rot}_{180^\circ}$ flips the kernel

$$\Delta_{i'}^{[l]} = \frac{\partial L}{\partial x_i^{[l]}} = \Delta_i^{[l]} * \text{rot}_{180^\circ} w_m^{[l+1]}\} g'(x_{i'}^{[l]}) \tag{3.61}$$

$$\frac{\partial L}{\partial w_{m'}^{[l]}} = \text{rot}_{180^\circ}\{\Delta_i^{[l]}\} * y_{m'}^{[l-1]}. \tag{3.62}$$

If there is a pooling layer that down samples the convolutional output such that $y_j^{[l]} = D_{\downarrow}(a_j^{[l]}) \neq a_j^{[l]}$ then $y_j^{[l]}$ is up sampled by an inverse mapping as for the down sampler. In the case of maxpooling where the maximum value inside a max pooling window is returned, the index the returned maximum value is stored such that only the gradient from the maximum value is computed[45][4][46].

### 3.3.4 CNN for time series applications

In the realm of sequence modeling, the go-to architectures have been recurrent neural networks (RNN)[47]. However, in the later years, CNNs have shown to perform on par or even better than the state-of-the-art RNNs in various of different sequence modeling tasks, such as word translation, and audio synthesis[1][48][49][50].

**Sequence to sequence modelling** performers a feature mapping, for a given input sequence $x_1, x_2, ..., x_T$, to a target prediction output $y_1, y_2, ..., y_T$. The general constraint of sequence modelling is that they only can use previous observation and its current to perform predictions. Meaning that sequence models can not see into the "future" to perform predictions[1].

RNNs is a family of neural network modules that are dedicated to sequence modeling. The main idea of RNNs is that they carry through a hidden state vector that "remember" temporal dependencies. RNNs also allows for mapping of any input length to an output of the same length. Figure 3.7 show the general structure of RNN architectures.

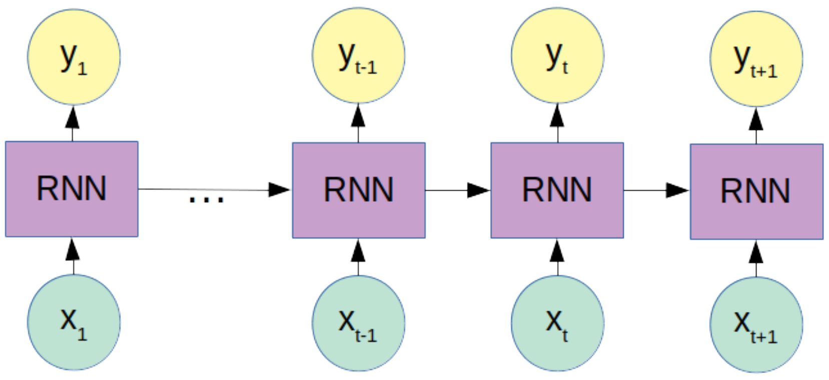


Figure 3.7: General structure of sequence to sequence RNN architecture. Where the hidden state is transferred to future predictions.

Compared to classical feed-forward networks, the training process for RNNs are generally more difficult. Further, RNNs are prone to the vanishing/exploding gradient problem[51][52], and often require fine-tuning of its hyper-parameters, and architecture adaption to the specific task at hand[1].

For CNN in time series applications, the standard is a sequence to sequence model that generates an output with the same lengths as the input. There are several important reasons for why CNN is a solution to sequence modeling, and some [1] points out some of them are;

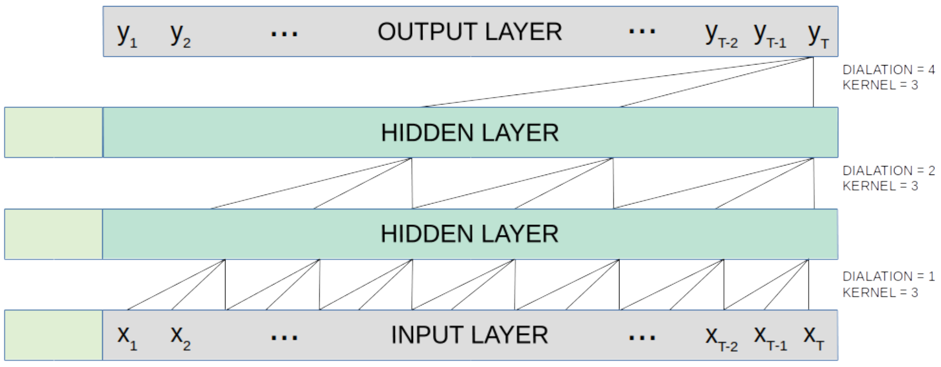


Figure 3.8: Overview of how a CNN can cover input features in its output by choosing proper dilation. This type overview is similar to the temporal CNN architecture proposed in [1].

- **Parallelism:** For CNN, the same operations and filters are applied for over the entire sequence, allowing for long time series sequences to be passed through in parallel.
- **Modification to the receptive field:** The receptive field of CNN is an important aspect form learning temporal dependencies of a CNN. Adjusting the dilation or filter size throughout the convolutions layers, are viable options to modify the "memory" of the model, and can be modified to the particular task at hand.
- **Stable gradients:** CNN are less prone to the problem of vanishing and exploding gradients while back-propagating[52].
- **Potential for lower memory requirements.** RNN requires storage of hidden states, and cell gates when back-propagating. CNN does not necessarily require this, and therefore CNN can be less memory bound, especially for long sequences during training.

However, there is also drawbacks and challenges compare to RNN and some of them are

- **Data storage during training:** During validation, RNN takes the current observation and the a hidden state vector as input to create a prediction. In most cases this require less storage than CNN, since CNN requires the same input length as it had during training.

- **Application in different problem domains:** Different problem domains might have different requirements to ”remembering” dependencies. This can be adjusted by modifying the receptive fields of the layers, but this needs Modification for the task at hand. See Figure 3.8 for a overview of how dilation can increase the receptive field, while still cover all input features. This restrictions might constrain CNN architectures in some applications, and should be considered.

- **Fixed input and output lengths:** Traditionally CNN requires a fixed input length and a fixed output. RNN resolves this issue, which is particularly relevant for many time series applications where the input length may be varying. Even though new convolution approaches somewhat resolve this, it is a relative task-dependent whether these convolutional approaches resolve this issue.

We have defined our problem as a time series segmentation problem, and in contrast to sequence models

- We are not constraint to only use current and previous observations to perform a prediction, meaning we able to ”peek” into the future.

- The target output is the same shape as the input. Reducing the need for RNNs that can handle varying input sizes.

## 3.4 Further deep learning details

In this section, we aim to list some further details about neural networks that have not yet been mentioned but are important to consider.

### 3.4.1 Data preparation

Deep learning methods are often very dependent on its input and a common method to ensure that input features with different scales have equal importance is by centering and normalization of the input data.

In the case of EV detection, we might argue that normalization is of less importance since the aim is to detect loads above a certain threshold. By normalizing, some of this information might get lost, which becomes less obvious when data is normalized. The master thesis by B. Fesche explored this, and their findings said that **not** normalizing the input features gave overall better results[3]. Therefore in this thesis, we do not normalize our input data. This result, is consistent with a classical paper by J. Kelly that reach the same conclusion that for neural network in the domain of NILM it might be beneficial to not normalize our data[53].

### 3.4.2 Weight initialisation

Before training a neural network, the parameters have to be initialized, and how this is done can impact the performance. The bias terms is usually initialized to be zero or to ensure the nodes "fires off" at the begging of the training process. They can also be initialized to have a small value.

For the weights, its important with proper initialization. An example of what not to do is to initialize all parameters with the same value since this results in all weights having the same gradient, and further, they all are learning the same thing.

It has been shown that initializing the weights with small random numbers with zero mean and variance scaled according to the input and output size has shown to work well, and has become a standard method for initialization[54].

### 3.4.3 Batch normalization

Batch normalization is a technique that forces the linear combinations within nodes to have a unit Gaussian distribution. The motivation for performing batch normalization is to reduce the covariance shift, which can be defined as; "... the change in the distribution of the network activation's due to the change of network parameters during training"[55]. Results have shown that this has made models more robust and less dependent on proper weight initialization[55].

1D Batch normalization normalizes an batch to have unit variance and zero mean per dimension. If we have a $d$-dimensional input $x = (x_1, x_2, ..., x_d)$, where each element of $x$ is a vector. The normalization per dimension can be written out as a linear transformation

$$\mathrm{BN}(x_i)_{\gamma_i,\beta_i} = \frac{x_i - \mathrm{E}[x_i]}{\sqrt{\mathrm{Var}[x_i] + \epsilon}}\gamma_i + \beta_i \tag{3.63}$$

where $\gamma_i$ and $\beta_i$ is a parameters per dimension that scales the shift of the normalized values, these parameters may be optimized during training. The idea to introduce

$\gamma$ and $\beta$ is to make sure to not constrain what the layers may represent[55]. An example of this is shown in Figure 3.9 that shows normalizing the input of a sigmoid activation reduces the non linearity of the activation.

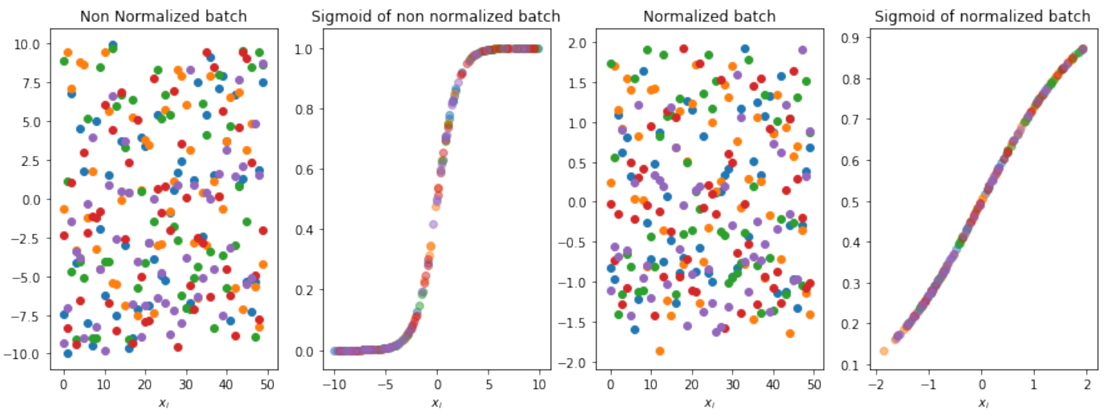


Figure 3.9: Comparing normalized and non normalized batch input into an sigmoid activation. Showing that normalized activation's becomes close to linear.

## 3.5 Summary

In the theory section, we have described feature-based clustering of time series, and the implemented clustering method of GMM. This theory is relevant for the problem of EV load profiling.

Further, we presented the supervised classification/segmentation problem, and details about the deep learning frameworks relevant for the problem of EV event detection. Which is dense and convolutional neural networks.

In the next chapter we will provide a detailed description of our methods for solving the problem of EV load profiling, and EV event detection.

# Chapter 4

# Methods

In this chapter we describe the methods, for solving the two problems relevant for this thesis, that is

- **EV load profiling by Gaussian mixture modeling of detrended load profiles:** We propose detrending the raw smart meter series before extracting load profiles. With the aim to discover underlying patterns unique to EV owners, that charges their EV at home.
- **EV event detection by exploring different CNN architectures:** We present the models based on previous research, as well as a newly proposed modified version of UTime. The aim with these models is to capture instances when an EV is charging from raw smart meter data.

## 4.1 EV load profiling: Clustering of weekly-hourly load profiles

Our proposed method of feature-based clustering method inspired by the data-driven approaches in [8][24]. The motivation for including EV load profiling is because of lack of ground truth in our dataset. Resulting in the need for more validation of our problem of EV detection, and therefore we aim to utilize the EV load profiling results for comparison.

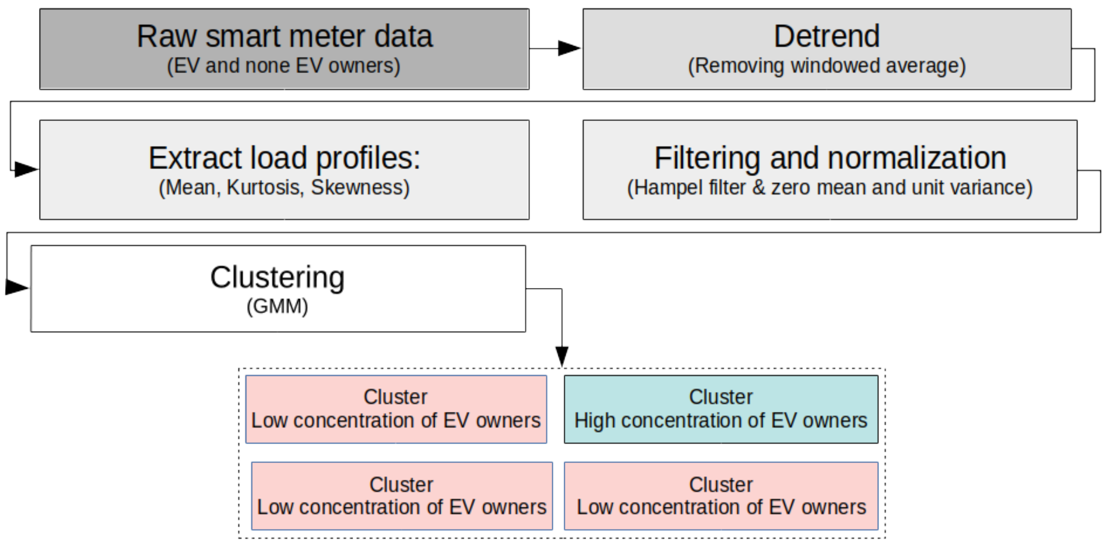

Raw smart meter data
(EV and none EV owners)
Detrend
(Removing windowed average)
Extract load profiles:
(Mean, Kurtosis, Skewness)
Filtering and normalization
(Hampel filter & zero mean and unit variance)
Clustering
(GMM)
Cluster
Low concentration of EV owners
Cluster
High concentration of EV owners
Cluster
Low concentration of EV owners
Cluster
Low concentration of EV owners

Figure 4.1: Diagram showing the steps of the implemented method for EV load profiling.

A diagram of the clustering process is summarised in Figure 4.1, and can be described in four stages.

1. Detrending
2. Feature extraction
3. Filtering and normalization
4. Clustering

### 4.1.1 Detrending

There are several reasons for why we propose detrending the smart meter series before feature extraction:

- The smart meter series has a strong seasonal trend, as shown in Figure 4.2. Extracting load profiles from such raw time series leads to a high variation at each hour of the week. Deterending reduces this variation, and while preserving meaning-full information regarding EV detection.
- In the EIDSIVA dataset most of the smart meter series have an duration for about an year. If we where to extract load profiles for each season of the

year, we would get sparse data points. In order to get more data points, which are comparable regardless of seasonal change, detrending is required.

- EV charging is assumed to have a high power consumption for shorter periods. Therefore we might assume that removing the seasonal trend will still preserve the higher power peeks from EV charging.

Although several detrending algorithms, such as empirical mode decomposition (EMD)[56] and locally estimated scatterplot smoothing (LESS)[57], were considered, we have chosen a process of removing the mean of neighboring values. The reasoning for this approach is because of its robustness and speed when implemented on a large amount of smart meter series. There is also little need for supervision to verify whether the proper trend where removed, making it easier to implement for large datasets. Further, by removing the average, it is reasonable to assume that the characteristics of EV charging are preserved if we choose a proper window size since these events have a relatively short pulse of high power consumption.

The implemented detrending process can be described as follows: If we assume a time series $x = [x_1, x_2, x_3, ..., x_T]$, with duration $T$ and define a window size of $L$ the trend $c_j$ for point $x_j$ can be computed as

$$c_j = \frac{1}{L} \sum_{i=j-\frac{L}{2}}^{j+\frac{L}{2}} x_i \qquad \text{for} \quad \frac{L}{2} < j < T - \frac{L}{2} \tag{4.1}$$

To remove seasonal trend we may simply subtract the trend $c_j$ from point $x_i$

$$x_j - c_j \qquad \text{for} \quad \frac{L}{2} < j < T - \frac{L}{2}. \tag{4.2}$$

Note the first and last $\frac{L}{2}$ elements of the series is omitted when detrending. Figure 4.2 shows an example of detrending a consumer in the Eidsiva data set.

After detrending, we extract weekly-hourly load profiles (a feature for every hour of the week).

### 4.1.2 Feature extraction

The three separate features we extracted is

1. Mean: $\mu_h = \frac{1}{N_h} \sum_{j=1}^{N_h} x_{h,j} \qquad \text{for} \quad h = 1, 2, .., 168$

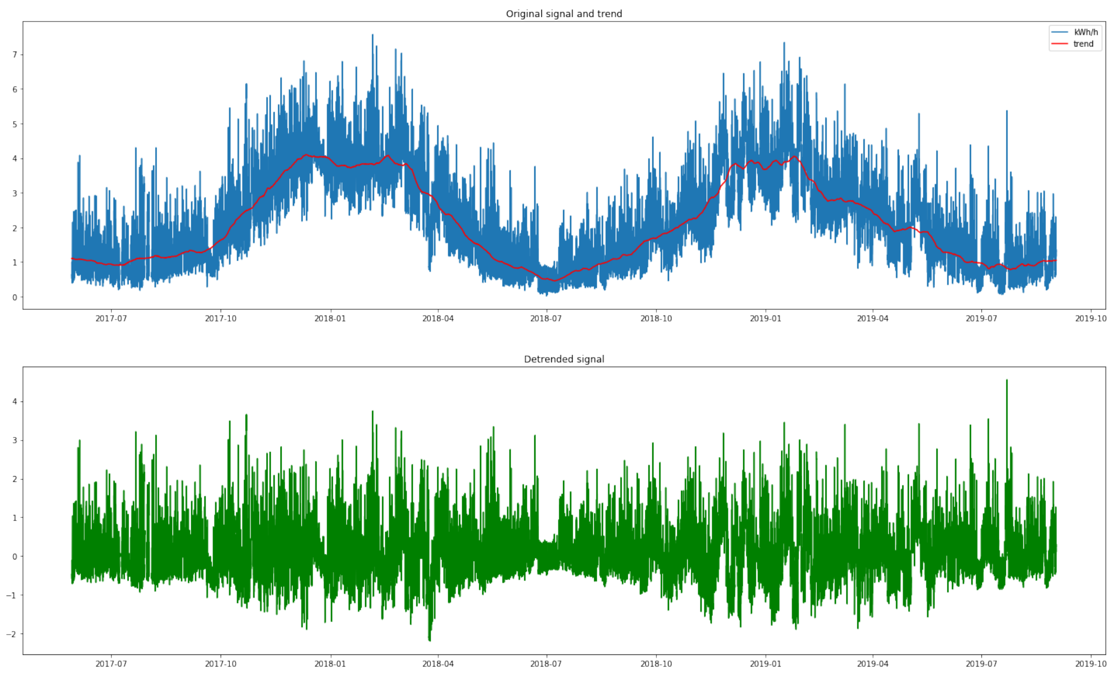


Figure 4.2: Detrended smart meter load series by removing the rolling mean from the eidsiva dataset.

2. Skewness: $s_h = \frac{1}{N_h} \frac{\sum_{j=1}^{N_h}(x_{h,j}-\mu_h)^3}{(\frac{1}{N_h}\sum_{j=1}^{N_h}(x_j-\mu_h)^2)^{\frac{3}{2}}}$ for $h = 1, 2, .., 168$

3. Kurtosis: $k_h = N_h \frac{\sum_{j=1}^{N_h}(x_{h,j}-\mu_h)^4}{(\sum_{j=1}^{N_h}(x_j-\mu_h)^2)^2}$ for $h = 1, 2, .., 168$

$N_h$ is how many measurement points there is for the $h$th hour of the week, starting from Monday and ending on Sunday. These features are extracted for each customer. Figure 4.3 shows an example of mean and kurtosis feature extracted from Figure 4.2.

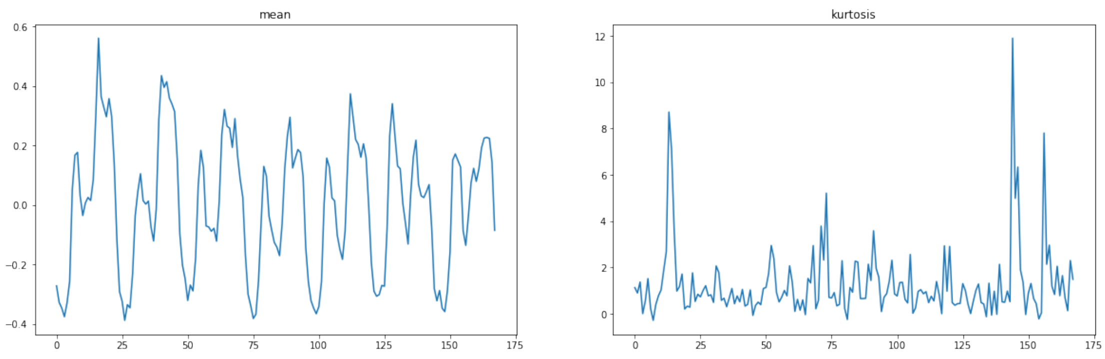


Figure 4.3: Mean and kurtosis features extracted from Figure 4.2.

### 4.1.3 Filtering and normalization

Since outliers can negatively influence clustering algorithms, a Hampel filter is applied to replace outliers above a certain threshold from the median, typically three standard deviations from the median[8][58]. In practice, the Hampel filter is applied elementwise in the feature space, which means that for each hour of the week, outliers more than three standard deviations from the median at the current hour of the week is replaced with the median.

Further each element $x_i$ in feature vector $x$ is normalized to have zero mean and unit variance

$$x_i = \frac{x_i - \mu}{\sigma} \tag{4.3}$$

where $\mu$ and $\sigma$ is the mean and variance of feature vector x, respectively. By normalizing the relative shape compared to its windowed, the trend is emphasized.

Due to the spike of the kurtosis and skewness in the weekends (see Figure 4.3) the hours from 129 to 168 is excluded, resulting in input index has the range $h = 1, 2, .., 128$. Meaning, the feature vector is hourly profiles from Monday to Friday night. These filtered and normalized features will be used as input observation into the Gaussian mixture model.

### 4.1.4 Gaussian mixture modelling (GMM)

The choice of the clustering algorithm is GMM, and the algorithm is described in Chapter 3.1.2. Motivation for this is because of its resemblance to K-means that has previously been used for clustering of smart meter data but allows for soft cluster assignment and none symmetric decision boundaries in the feature space[30][15]. Allowing for different variations in the each feature dimension. GMM does not require a predefined distance measure. A challenge with GMM is that they require a predefined number of clusters $k$. However, it has been shown that the methods have proven successful for clustering time series[59].

For this thesis, we aim to separate registered EV owners into distinct clusters. Therefore we can experiment with the different numbers of components in order to achieve good separation. Furthermore, we will use the concentration of EV owners within a cluster to determine cluster has captured uniqueness between some EV owners. The concentration of EV owners within a cluster is simply defined as

$$\frac{\text{Number of EV owners within a cluster}}{\text{Total number of customers within a cluster}}$$

### 4.1.5 Implementation

The prepossessing and detrending procedure by the popular Python libraries NumPy and Pandas[60][61].

The implemented Gaussian mixture is from scikit-learn, a python library with different tools for machine learning tasks, which optimizes by using the described EM algorithm[62]. For GMM the number of components needs to be predefined as well as the structure of covariance matrices. These settings need to be explored, in the experiments reported in Chapter 6.

## 4.2 Proposed models for EV event detection

As an observant reader may have noticed, we have chosen an approach of using convolutional and dense components in our deep learning models. There are mainly two types of models we explore in this thesis:

1. The first one is based on the CNN architectures from earlier research of EV event detection. In this project, they are referred to as convolutional+atuoencoders. In essence they consist of convolutional layers, followed by dense layers.

2. The second model is UTime, which is a fully convolutional network inspired by the popular UNet. Utime has previously been proposed for the segmentation of medical sleep stage data[4]. Since our problem are a segmentation problems as well, we propose a modified version of UTime for the application of EV event detection.

### 4.2.1 Convolutional + Autoencoder

The main idea behind the Convolutional+Autoencoder architecture is that the convolutional layers extracts low level features from an aggregated power signal. Further, the output from the convolutional layers are passed through the autoencoder that performs reconstruction of the EV charging signal, or in our case prediction of EV event detection[2]. The convolutional feature extracting/pattern-matching phase is similar to the cross-correlation filtering in [23]. However, the filter weights in the convolutional+autoencoder is trained and not cross correlated with charging signatures from a database.

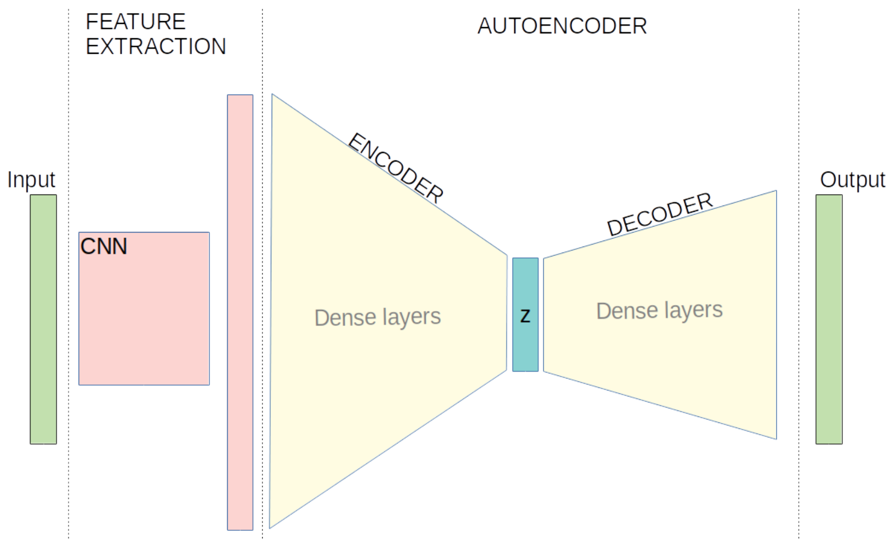


Figure 4.4: Overview of a CNN+autoencoder. The CNN performs pattern matching, with the raw input signal, and the autoencoder maps the out put to from the CNN to the prediction output space.

### Autoencoder

The proposed autoencoder consist of an encoder and a decoder. In our case the input to the encoder is the output from the last convolutional layer, see Figure 4.4 for an overview.

The encoder $g$ is a deconstruction phase consisting of dense layers that performs the feature mapping of input features $x$ to a lower dimension embedding $z$

$$g : x \rightarrow z. \tag{4.4}$$

Followed by the encoder there is decoder phase that performs ”reconstruction” of z into the predictive space $y$

$$h : z \rightarrow y. \tag{4.5}$$

Figure 4.4 shows an outlier of the CNN+autoencoder architecture proposed in [2] and will be referred to as CNN+AUTO. A variation to this is to remove the decoder phase and use $z$ as predictive output. This type of model with only the encoder, is the type of model proposed in [6][3], and will be named CNN+ENCODER, and are more similar to a traditional CNN architecture described in Chapter 3.

### Proposed models

In this project, three different convolutional + autoencoder is explored. All architectures are summarised in Figure 4.5, showing all the details of all models. The models will be referred to as CNN+AUTO, CNN+ENCODER, and CNN+DENSE.

- CNN+AUTO is the proposed convolutional network proposed in [2].
- CNN+ENCODER is equivalent to the model proposed in [6][3].
- CNN+DENSE is a new model for the task of EV detection. The architecture is similar, but not identical to the model described in a survey of 1D CNN[40].

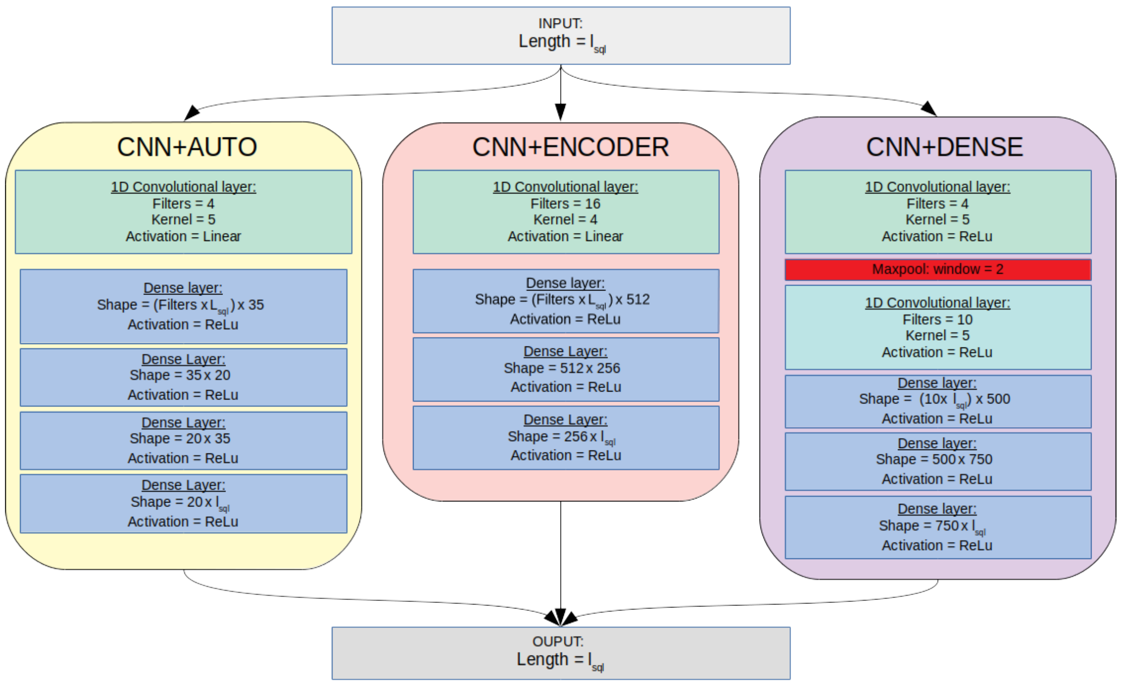

INPUT:
Length = $l_{sql}$
CNN+AUTO
1D Convolutional layer: Filters = 4 Kernel = 5 Activation = Linear
Dense layer: Shape = (Filters x $L_{sql}$) x 35 Activation = ReLu
Dense Layer: Shape = 35 x 20 Activation = ReLu
Dense Layer: Shape = 20 x 35 Activation = ReLu
Dense Layer: Shape = 20 x $l_{sql}$ Activation = ReLu
CNN+ENCODER
1D Convolutional layer: Filters = 16 Kernel = 4 Activation = Linear
Dense layer: Shape = (Filters x $L_{sql}$) x 512 Activation = ReLu
Dense Layer: Shape = 512 x 256 Activation = ReLu
Dense Layer: Shape = 256 x $l_{sql}$ Activation = ReLu
CNN+DENSE
1D Convolutional layer: Filters = 4 Kernel = 5 Activation = ReLu
Maxpool: window = 2
1D Convolutional layer: Filters = 10 Kernel = 5 Activation = ReLu
Dense layer: Shape = (10x $l_{sql}$) x 500 Activation = ReLu
Dense layer: Shape = 500 x 750 Activation = ReLu
Dense layer: Shape = 750 x $l_{sql}$ Activation = ReLu
OUPUT:
Length = $l_{sql}$

Figure 4.5: The three proposed CNN+Autoencoder architectures. CNN+AUTO is based on [2], CNN+ENCODER is based on [3] and CNN+DENSE is the first guess of CNN from the initial experiment exploring different sampling rates.

In the convolutional layer, CNN+AUTO and CNN+DENSE have a kernel size of 5, and CNN+ENCODER has a kernel size of 4. CNN+AUTO and CNN+ENCODER have no pooling layers and linear activation resulting in a relatively low receptive field in the feature extraction phase. Therefore we might expect a long sequence input not to be beneficial since the feature extraction phase mostly captures local dependencies. However, we might also guess that these models might work if the filter sizes match patterns of charge events. The CNN+DENSE model has two convolutional layers with ReLu activation and with max-pooling in between. This increases the receptive field in the second convolutional layer. Otherwise, the model has a similar architecture to the CNN+ENCODER model in its dense layers.

### Number of parameters

For all the proposed CNN+autoencoder architectures; varying the input lengths requires modification of the first dense layer to be fully connected with the output from the last convolutional layer. Also the last layer requires modification to have the same output lengths as the input. If we assume that the convolutional layers is zero padded such that the dimensionality is preserved. The input length of the first dense layer in CNN+AUTO and CNN+ENCODER becomes

$$l_{1.\text{dense}} = D \cdot (l_{sql} - 1) \tag{4.6}$$

where $l_{sql}$ is the input sequence length, and D the number of filters in the kernel. Figure 4.6 shows that the number of parameters increases linearly, as the input lengths is increasing.

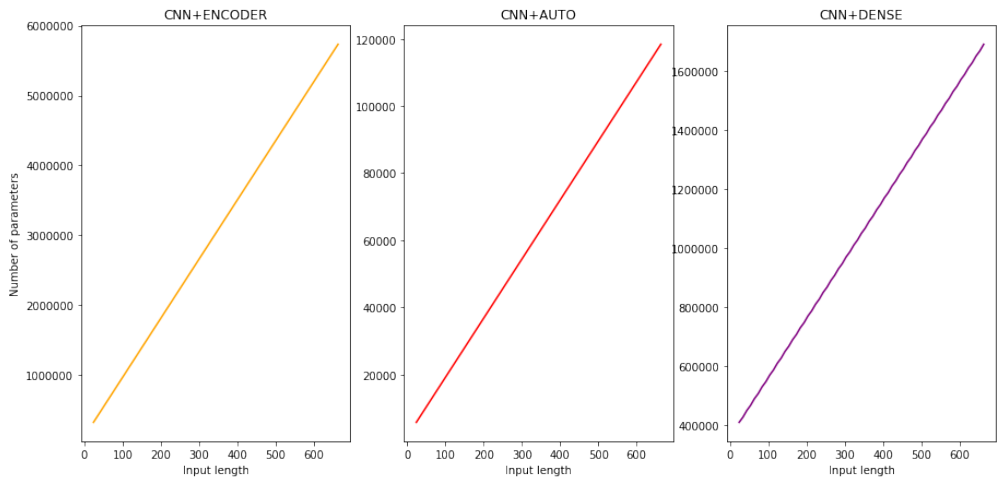


Figure 4.6: Number of parameters in the proposed CNN+Autosencoders for varying input sequence lengths.

In the experiments, we aim to investigate how the models perform with different input lengths. The experimental results and conclusion of the best performing CNN+Autoencoder architecture will is reported in Chapter 7.

### 4.2.2 U-time: A one dimensional U-net

U-net is a fully convolutional network image segmentation[63] (classifying all pixel of an input image) and has proven to work well for a variation of image segmentation tasks, such bio medical image segmentation[63]. Inspired by U-Net a time-series variation of U-Net has been proposed as UTime. UTime is a fully feed-forward deep learning approach for time series segmentation developed for the analysis of sleep stage data. The segmentation is done by classifying every sampling point of a fixed length time series signal and then aggregate these classifications over fixed intervals to preform segment predictions[4].

#### U-time architecture[4]

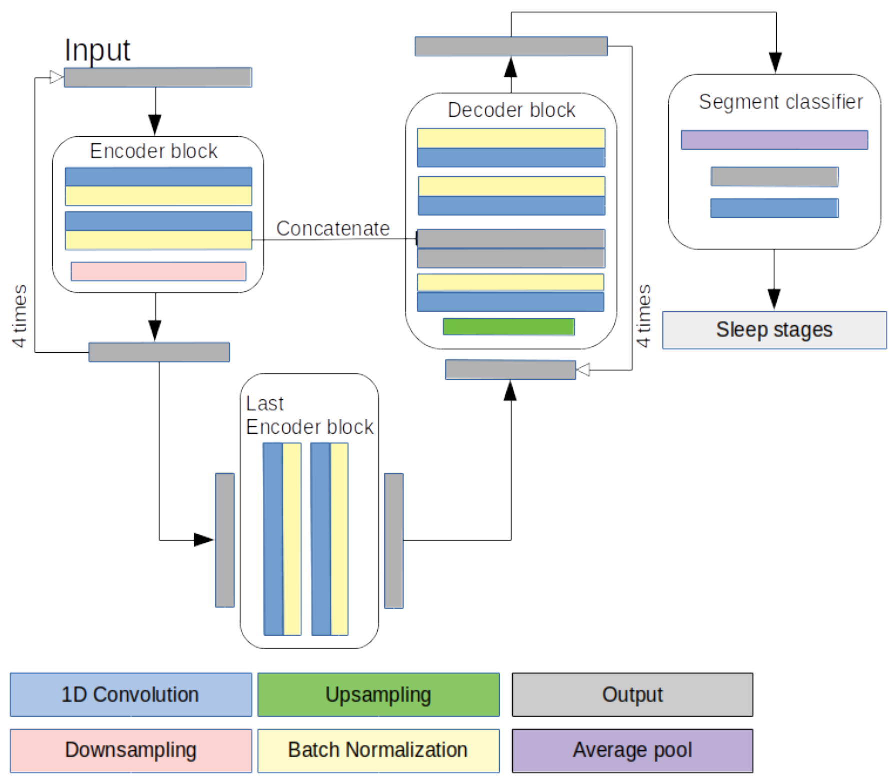


Figure 4.7: Figure of the proposed UTime architecture for sleep stage classification[4].

U-time is a fully convolutional autoencoder network with a segment classifier as its final module. For this project, the segment classifier is omitted since we want predictions for every time point of our time series.

U-time requires a fixed input length $x \in \mathbb{R}^{t \times C}$ where $t$ is the number of sample points and $C$ is the number of input channels. Further U-time aims to give class confidence scores for $T$ connected segments of length $i$ of the connected signal segment. The input length can be rewritten as $t = i \cdot T$.

The U-Time model can be formulated as

$$f(\mathbf{x}; \theta) : \mathbb{R}^{T \times i \times C} \rightarrow \mathbb{R}^{T \times K} \tag{4.7}$$

with parameters $\theta$ that maps $x$ to class confidence scores for predicting $K$ classes for all $T$ segments. The model processes a 1D signal of length $t = T \cdot i$ in each channel[4]. Further, the model can be divided into three sub-modules an encoder, decoder, and a segment classifier:

**Encoder**: Takes the input signal through four convolutional blocks where two one-dimensional dilated convolutions is followed by batch normalization and a max-pooling layer as the final step of the block. As a final step of the encoder, two one dimensional dilated convolutions are followed by batch normalization.

The original UTime proposes stacked dilation and an aggressive downsampling rate by choosing large max-pooling windows in order to increase the receptive field of the model output. If the pooling window $P_b$ after encoder block $b$ the minimum input sequence length can be computed by

$$\text{minimum input sequence} = \prod_{b=1}^{B-1} P_b \tag{4.8}$$

where $B$ is the number of encoder blocks since the convolutional operations are zero-padded such that input dimensionality is preserved.

**Decoder:** ("reconstruction phase") The decoder has four transpose convolutional blocks described in the encoder, meaning that they first up samples at the same rate as the encoder down samples. After each up sample, convolution followed by batch normalization is performed. The output from batch normalization is concatenated with the corresponding batch normalization from the encoder. Then a new convolutional layer is followed by batch normalization. The final convolutional layer is modified to have the dimensionality $\mathbb{R}^{t \times K}$ giving a confidence score for $K$ classes for each timestamp in $x$.

**Segment classifier:** The segment classifier classifies $i$ section of length $T$ from the output of the last decoder block. It achieves this by taking the output of the last decoder block through a layer of average pooling with width $i$ and stride $i$, then passed through a pointwise convolutional kernel (kernel size 1). That results in a classification score array of shape $T \times K$[4].

The original UTime had a kernel size of 5 and a dilation of 9 for all convolutions layers. The encoder blocks had 16, 32, 64 128, and 256 filters respectively, and the decoder had 128, 64, 32, 16 Filters. Since the model tried to segment five sleep stages, the final output of the encoder where a convolution layer with five filters with a tanh activation. The pooling windows where $P_1 = 10$, $P_2 = 8$, $P_3 = 6$ and $P_4 = 4$ such that the minimum sequence length is 1920. For the problem EV detection of hourly smart meter data this results in a minimum input length of 80 days. Since smart meter data often has missing values, the pooling windows should be modified to handle shorter input lengths.

### Modified UTime for EV detection

The proposed modified UTime for EV detection has the same number of filters and kernel size as the original. For the problem of EV detection, the last convolutional layer has one filter because it is a binary classification problem, and as mentioned the segment classifier is omitted. Further, the pooling windows is changed to 2, and dilation to 1. The reasoning for these changes is because of the lower sampling rate of hourly smart meter data. We would expect charge events to have a duration of a few hours (1 - 12 hours), and therefore decreasing the dilation, the filters will still be able to cover the charge events, and capture surrounding dependencies. These modification is also allow handling of short input lengths.

Table 4.1: Number of trainable parameters of the proposed UTime model.

| Encoder blocks | Parameters |
|---|---|
| 1 | 1473 |
| 2 | 15889 |
| 3 | 72881 |
| 4 | 299505 |
| 5 | 1203313 |

Figure 4.8 shows the details of the modified UTime with $B$ encoder blocks and $B - 1$ decoder blocks and pooling windows at the current block $b$ is $P_b$. The number of parameters of the UTime model with different depths is listed in Table 4.1. It should also be noted that the smaller pooling windows make the model more memory-bound since the last batch normalization layers in the encoder blocks has to be stored because of the skip connections (concatenation between the encoder

and decoder blocks).

When experimenting with the proposed UTime architecture, the depth and input length is explored, and the results is reported in Chapter 7.

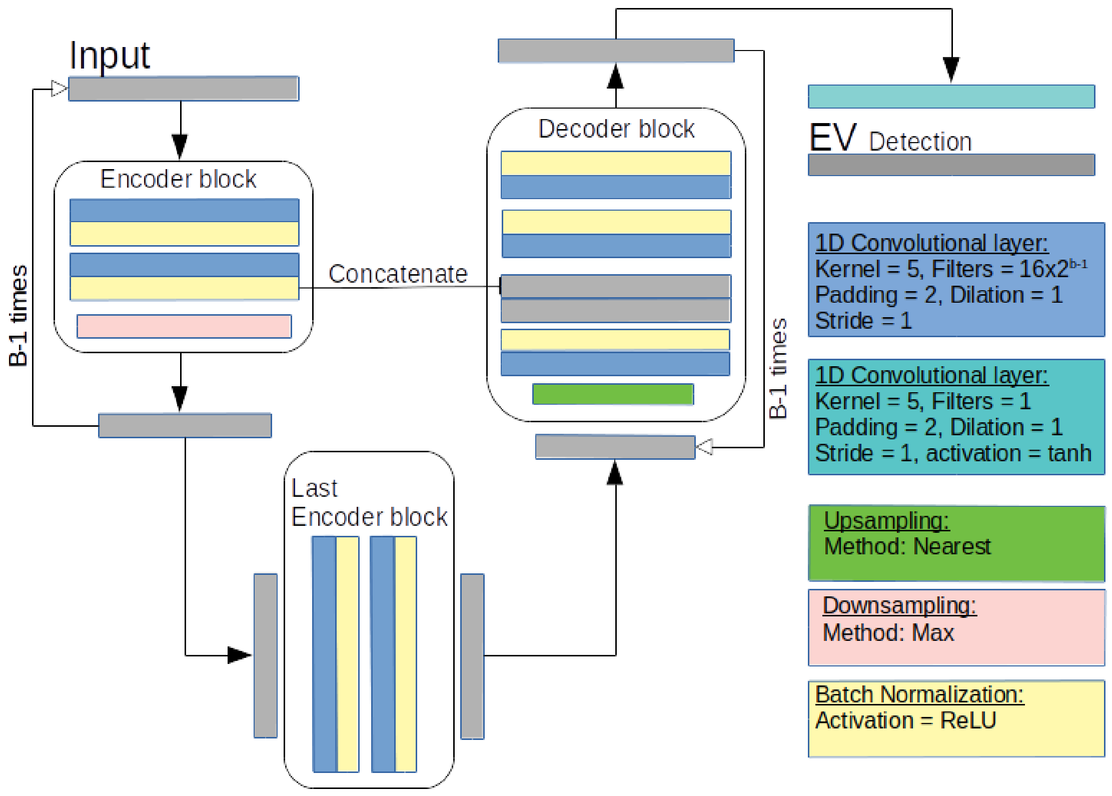


Figure 4.8: Proposed modified UTime for the task of EV detection.

### 4.2.3 Implementation of deep learning models

All deep learning models proposed are implemented with PyTorch, a python library for deep learning that enables automatic differentiation and hardware acceleration both by multiprocessing of CPUs and Cuda enabled GPUs[64].

### 4.2.4 Why CNN?

As discussed in Chapter 3.3.4, a natural choice of temporal data is RNN. However, in this thesis, we use CNN architectures exclusively. Why is that?

Our problem is defined as a time series segmentation problem, where we are allowed to peek into the future to perform predictions, which is usually a constraint in sequence modeling. Removing this constraint increases the resembles of our problem with its 2D image counterpart of image segmentation.

In the field of 2D images, CNN architectures are at the forefront of the state-of-the-art. As discussed in Chapter 3.3.4, CNN is an attractive solution since it have proven to work well for temporal data, and offer benefits such as robustness and speed while training, and the great speedup by parallelization during evaluations.

Further, another consideration is the method our data set is generated. Since we add charge sequences at random on top of smart meter data (see Chapter 5) we might expect that an RNN might have a more difficult time learning temporal dependencies from previous observations.

## 4.3 EV detection of a long smart meter sequence

A smart meter series from a customer have long duration's, compared the required model input length. In this section we propose a sliding window approach to detecting EV charge events of long smart meter sequences, giving us several confidence scores for each time point, and reducing the number of missing values in our predictions.

The models suggested requires a fixed sequence length $l$, where the prediction $\hat{y}_i \in \mathbb{R}^l$ of such an input vector $x_i \in \mathbb{R}^l$ can be written as

$$f(x_i) = \hat{y}_i \tag{4.9}$$

where $f$ is the predictive model. The models can also predict batches $B$ of input sequences for faster computation. The vectorization of $f$ can be written as

$$f(X) = \hat{Y} \tag{4.10}$$

where $X \in \mathbb{R}^{B\times 1\times l}$ and $\hat{Y} \in \mathbb{R}^{B\times l}$.

When predicting EV charge events for a long smart meter sequence $t = [t_1, t_2, .., t_N]$ of length $N$ a rolling window of step size 1 is applied resulting in row wise vector of the input array $X$ to be expressed as

$$X_{i,1,:} = [t_i, t_{i+1}, t_{i+2}, .., t_{i+l}] \qquad \text{for} \quad i = [1, 2, ..., N - l] \tag{4.11}$$

where $X$ is of shape $N - l \times 1 \times l$ with the assumption $l < N$. Since most smart meter signals have some missing values (NaNs), and since $f$ can't handle this. All elements for row vectors $x_i$ that containing NaN is set to zero, and after computing,

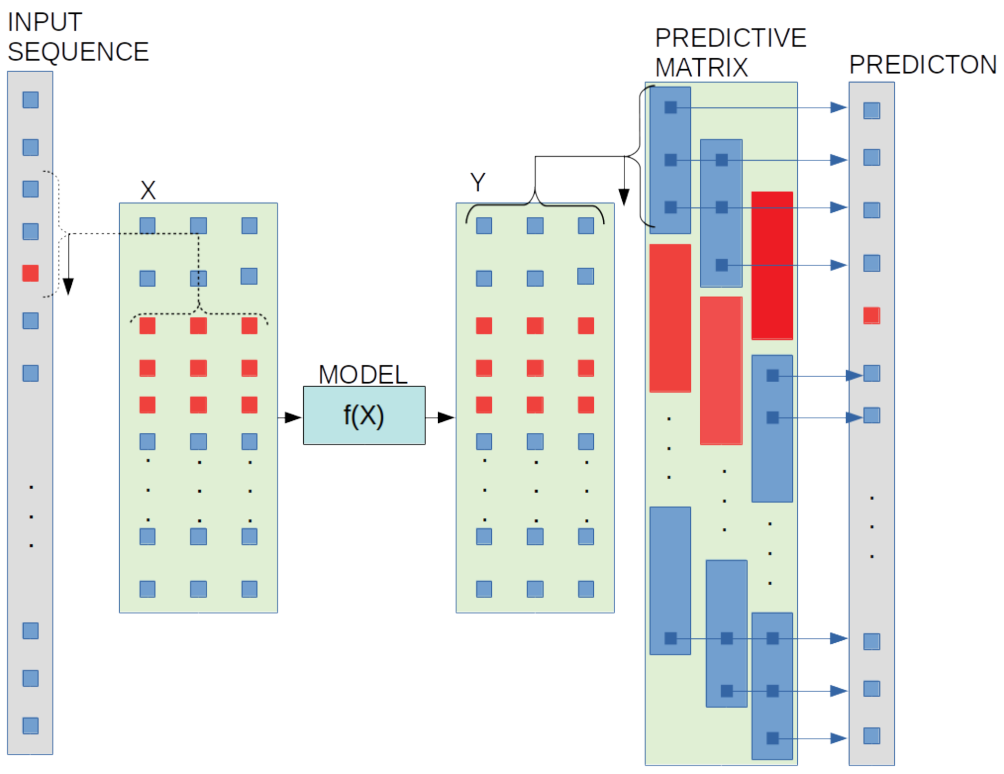


Figure 4.9: Description the implemented prediction of a long smart meter data sequences. The input sequence is stacked to fit the required sequence length of a model $f$ by a rolling window approach. Since model $f$ cant handle missing values (marked as red), they are set to zero in the input matrix X and set back to the missing value in Y before restacked in the predictive matrix. The final prediction of the input sequence is the mean of all relevant predictions (mean along the second axis excluding missing values).

the output $\hat{Y}$, corresponding row vectors containing NaNs in $X$ is set back to all NaNs in $\hat{Y}$. The predictions $\hat{Y}$ is then restacked in a $N \times l$ prediction matrix to fit the prediction of elements in $t$. The final confidence score of a point $t_n$ is done by computing the mean of all non NaN predictions for $t_n$. By using a constructed prediction matrix, this is done by computing the mean along the second axis, excluding all NaN values. Then the final prediction vector of the same size as $t$ will either have a confidence score of EV charging or a missing value where the prediction was not possible. Figure 4.9 summarizes the method of prediction and construction of the prediction matrix.

By implementing this rolling window method, we can see that the number of NaN predictions is significantly reduced compared to if the input sequence is restacked into input matrix X. However, this method is also more computationally expensive, especially for models that require long input lengths. The trade-off of between computation time and the proposed NaN handling technique is worth considering in practical implementations.

## 4.4 Summary

In this chapter we have provided a description about the model details for the problem of EV load profiling, and EV event detection. Further we described the implemented process of EV event detection of long smart meter sequences, which allows for less missing values, and several confidence scores for each time stamp.

Before presenting our experiments, we describe the datasets used in this thesis. The dataset chapter, requires special attention and care full reading, since we combine different data sources to generate datasets, and these will be referenced in the experiments.

# Chapter 5

# Datasets

In this chapter we describe the different data sources used, and the method of generating a labeled dataset. The last section gives and description of the different datasets that will be referenced in the experiments.

## 5.1 Data sources

The three data sources used in this project are:

- ACN-Data: A containing real-life EV charge signals[65]. Charge event from this data source, will be added to charge smart meter data where no EV charging is present.
- UK-dale: High-resolution smart meter data from five residential homes in the United Kingdom[66]. This data source allow us to investigate how the problem of EV event detection is affected by the sampling rate of the smart meter, and we know for certainty that no charge events is present.
- Eidsiva: Large data set provided by Eidsiva (a DSO in Norway). Containing power consumption from residential homes in Norway, as well as information about customers with an EV registered to its household.

### 5.1.1 The ACN-Data dataset

ACN-Dataset is an open dataset that contains workplace EV charging sessions, which is continuously being updated. The dataset comes from two Adaptive Charging Networks (ACN) located in California[65]. The two locations are Caltech and JPL.

Caltech is a research university located in Pasadena. They collect data from 54 EVSEs (Electric Vehicle Supply Equipment or charging stations), including 50 kW DC fast chargers. The Caltech charging stations is open to the public and are located in a parking garage near the campus gym. Since it is close to the gym, many drivers charge their EVs while working out in the morning or evening[65].

JPL is a national research lab located in La Canada. The site currently offers 50 EVSEs and is only open to employees.

For each charging session, a variety of data is collected. For this project, we are interested in the time series of power consumption when an EV is charging. ACN-DATA does not provide this directly. However, we can convert the available current signal to a power signal if we assume a constant voltage and use knowledge about the energy delivered when charging. Figure 5.1 shows nine charging currents downloaded from ACN-DATA.

#### Converting current to power signal

As mentioned, ACN-DATA does not provide a power signal directly, but we can convert the available current signal to a power signal by assuming a constant voltage and in addition use knowledge about the energy delivered during charging.

When converting the current $I(t)$ to a power signal $P(t)$ we assume constant voltage $V$. The proportionality can be written as

$$P(t) = V \cdot I(t).$$

Then the relation between total energy E delivered, current $I(t)$, and power signal $P(t)$ is

$$E = \int P(t)dt = V \int I(t)dt$$

From the above equation we can approximate the constant voltage to be

$$V = \frac{E}{\int I(t)dt} \approx \frac{E}{\sum_i I(t_i)}$$

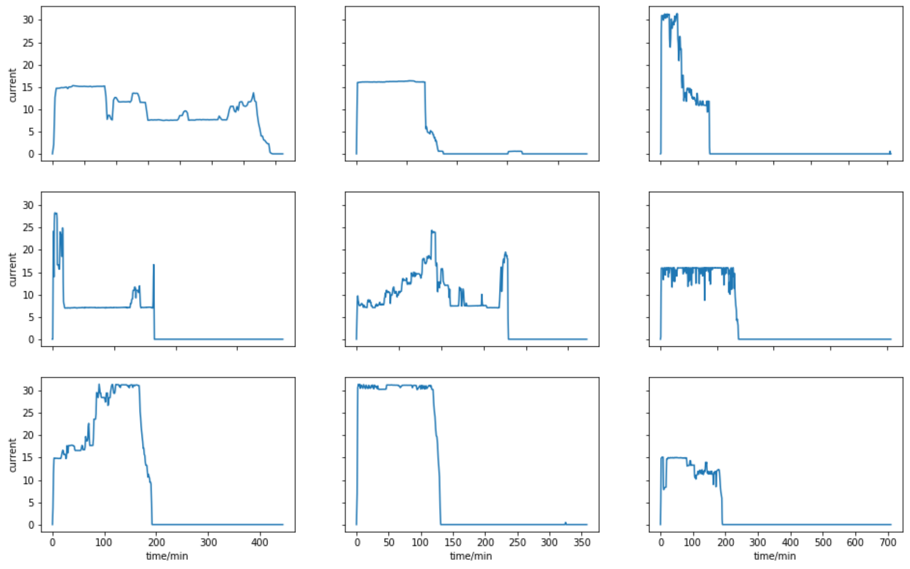


Figure 5.1: Nine examples of charging currents downloaded from ACN-Data. The sampling rate is 1 minute.

Taking into account energy unit of kWh and 1 minute sampling rate the implemented formula for a discrete signal becomes

$$V = \frac{1000 \cdot E}{\frac{1}{60} \sum_i I(t_i)}$$

### ACN-DATA summary

From ACN-DATA current signal from charge events between May 2018 and February 2019 was downloaded. Charging sessions with energy delivery of less than 4kWh was excluded. Note that not all events in this periods where downloaded, only a subset of charge event during this period.

The following cleanup was conducted:

- Keeping charge events within 90% confidence interval of estimated voltage to remove outliers.
- Removing charge events with a duration shorter than 1 hour.

After cleanup 3109 of 3391 charge events remained.

Figure 5.2 and 5.3 show distributions of estimated voltage, power, charge duration, and energy delivered of the dataset after cleanup. Most charge events have a maximum power of between 3kW and 7kW, indicating that the charge events comes from a semi-fast and slow EV chargers that may also be installed in residential homes. However, there are few charging instances with the expected maximum power from a type-c outlet.

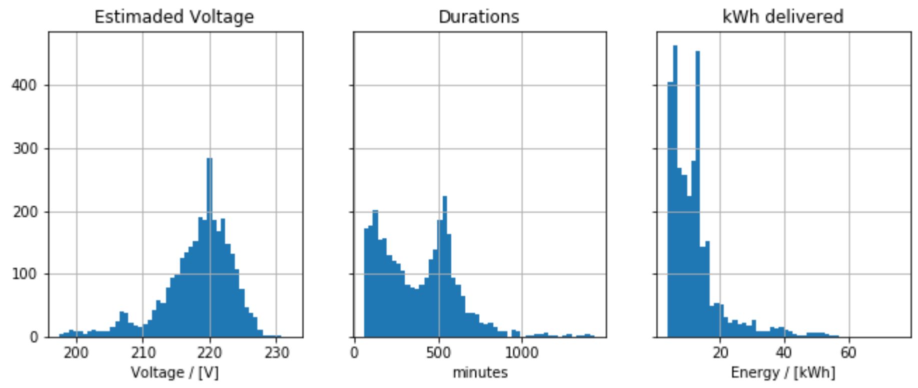


Figure 5.2: Histogram representation of charge events after cleanup

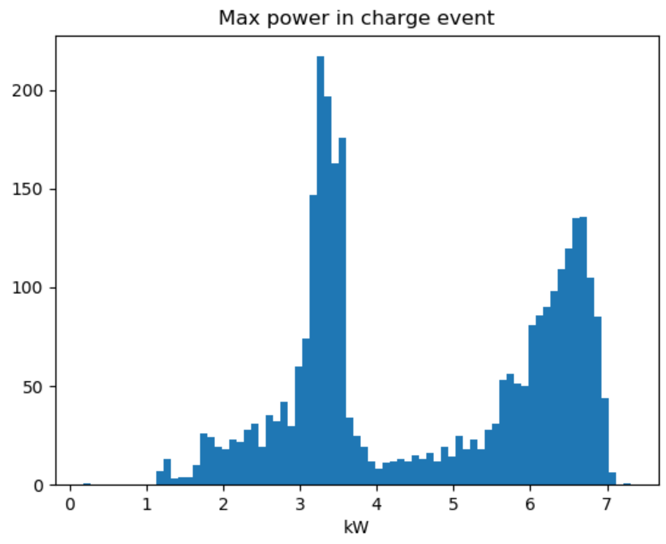


Figure 5.3: Histogram of the maximum power in the downloaded charge events after converting the current signal to power signal. It shows two dominant power peaks (at around 3.25kW and 6.8kW)

### 5.1.2 UK-DALE

UK-DALE (UK Domestic Appliance Level Electricity) is an open dataset from 5 UK residential houses where power demand is recorded. For all five houses, the total and individual appliance power demand is recorded every 6 seconds[66].

In this project, we are interested in the aggregated power signal (sum of all appliances) as a smart meter series. The 6s sampling rate of UK-DALE allows us to explore model performance for higher sampling rates. Table 5.1 shows the days registered for each household.

Table 5.1: Start and end dates, and number of days recorded for each household in UK-DALE.

| House | Start | end | Days of data |
|---|---|---|---|
| 1 | 09/11/2012 | 26/04/2017 | 27852 |
| 2 | 17/02/2013 | 10/10/2013 | 4010 |
| 3 | 27/02/2013 | 08/04/2013 | 673 |
| 4 | 09/03/2013 | 01/10/2013 | 3516 |
| 5 | 29/06/2014 | 13/11/2014 | 2344 |

### 5.1.3 Smart meter data from EIDSIVA

The Eidsiva dataset contains smart meter series from 8316 customers from the Hedmark-Oppland region in Norway. The data is represented as a time-series, with a sampling frequency of one hour. The identity of customers is anonymized internally at Eidsiva before provided to the participants of this project. The time series has two variables "date and time" and "kWh/h". The unit kWh/h is the rate of active energy transferred per unit of time h(hours). Figure 5.4 shows the a summary of the Eidsiva dataset before cleanup resampled to daily consumption.

In addition, Eidsiva has provided information about customers within the data set that has registered an EV (or hybrid vehicle) to its household. This allows us to divide the households into two categories: no EV and EV customers. For the households with registered EV, we know the EV model and month of registration. This information will become useful when validating and training on unlabeled data since we may separate time series with a higher likelihood of containing EV charging.

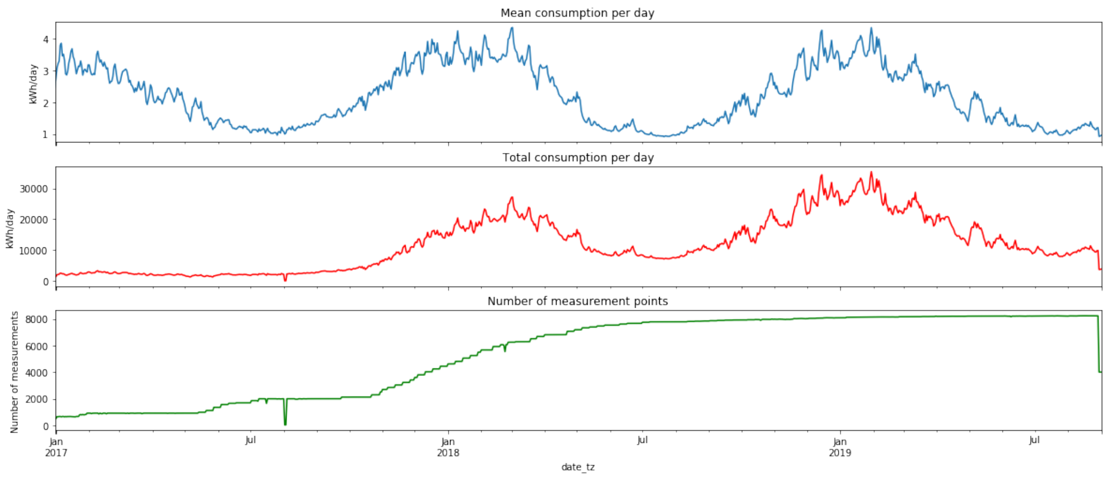


Figure 5.4: Brief summary of the Eidsiva dataset re sampled to daily consumption. Note the obvious seasonal trend and that most smart meter data is from mid 2018 to August 2019.

### Cleaning: EIDSIVA

The data provided from Eidsiva is a real life data set that requires cleaning. In this section we list how the data is cleaned and processed.

- Linearly interpolate data gaps shorter than 3 hours.
- Remove duplicate value and keep the maximum values.
- When anonymizing the data some small negative values occurs, this are handled by setting the negative values to zero.
- Exclude households that are missing 10% of their values.
- Exclude households that have a data gap larger than one week (168 hours). This might indicate faulty meter.
- The starting time for households with registered EV is moved to the time when an EV is registered on the household.
- Households with registered EV and non registered EV is separated.

After cleanup, the Eidsiva dataset is divided into five separate data groups, shown in Table 5.2. The data groups train, test and validation is customers with no registered EV, and the data group EV and Before EV contains smart meter series of EV owners after and before the time of EV registration.

Table 5.2: Number of consumer loads in the respective categorized data set after cleanup. Note that EV loads are owner with registered EV and No EV is owners with non registered EV.

| | No registered EV | | | Registered EV | |
|---|---|---|---|---|---|
| Category | Train | Validation | Test | EV | Before EV |
| Number of smart meter series | 1892 | 924 | 946 | 2487 | 1038 |

## 5.2 Generating a labeled data set

As previously mentioned, training of deep neural networks requires a large dataset to generalize well[40]. A large labeled data set has been difficult to get a hold of, and therefore the approach of generating labeled data is proposed. The main idea is to randomly add power signals from EV charge events on top of smart meter data where we know there are no EV charge events present.

The implemented method of generating a labeled dataset we need to predefine three values:

- A fixed sequence length $l_{sql}$. Since our model requires a fixed input length the smart meter series, needs to be partitioned into segments with a fixed length $l_{sql}$.
- The probability of a partitioned segment is containing EV charging: $p_{EV}$.
- The maximum number of EV charge events $N_{EV}$ that can be added to a partitioned segments.

When generating the data set; we iterate over the smart-meter series that does not contain EV charging: For the current smart meter series the total length is $L$ and the following process is done:

1. Randomly draw $L/l_{sql}$ start positions from a discrete uniform distribution $\mathcal{U}(0, l_{total} - l_{sql})$ and generate segments with a length of $l_{sql}$ from the drawn start positions. If there are any missing values in the segments it is disregarded.
2. For every segments we determine if it will contain EV charging session by drawing from a binomial distribution $\mathcal{B}(1, p_{ev})$.

3. If the value drawn is 1: between 1 and $N_{EV}$ EV charge events will be added to the segments with random start positions, they may overlap. The number and which events from ACN-data to be added is decided at random from discrete uniform distribution.

4. The labeled ground truth of a charge event will have a value of 1 where the power from charge session is larger than 1kW and 0 elsewhere.

Figure 5.5 shows a flowchart how the labeled data set is created.

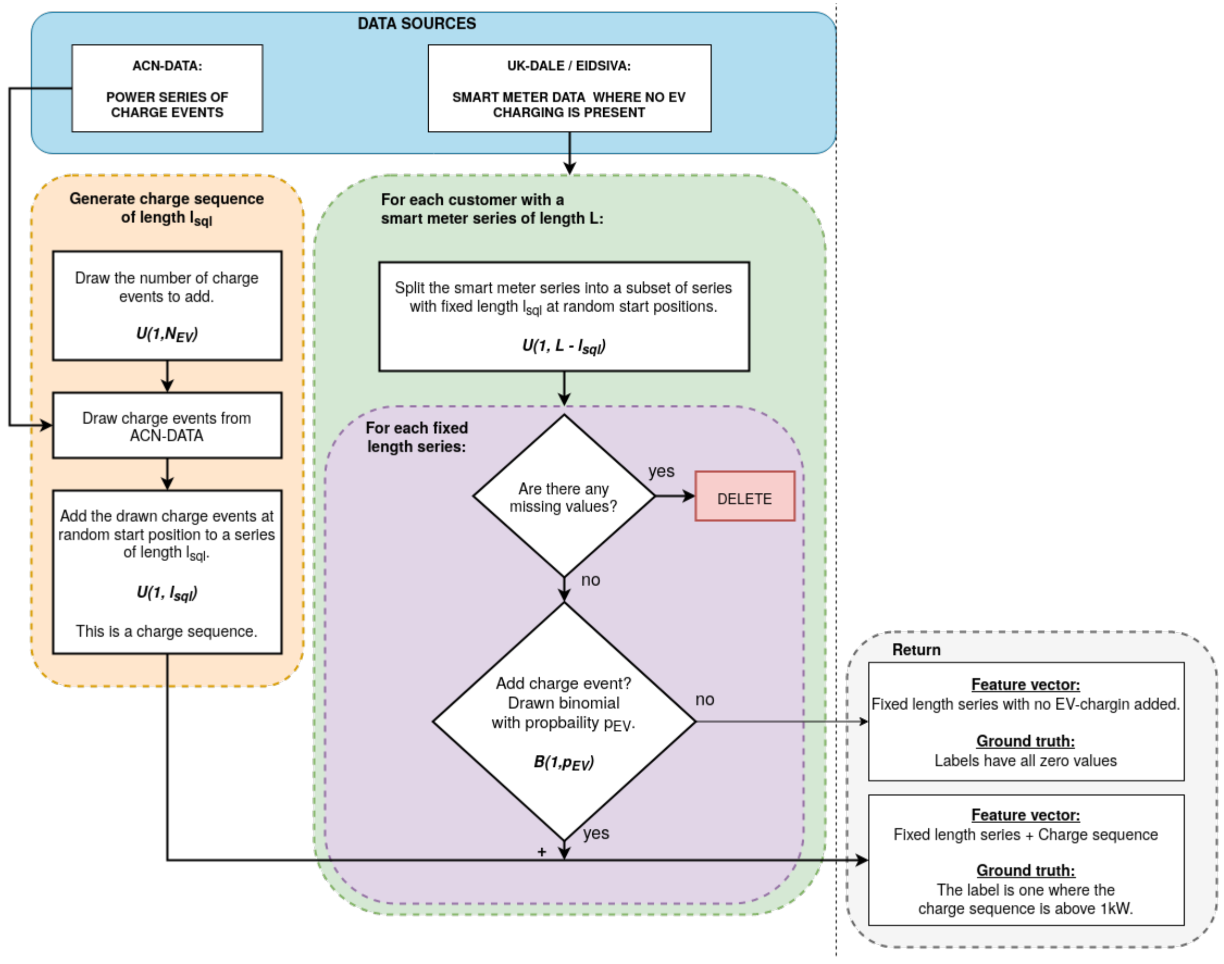

DATA SOURCES
ACN-DATA: POWER SERIES OF CHARGE EVENTS
UK-DALE / EIDSIVA: SMART METER DATA WHERE NO EV CHARGING IS PRESENT
Generate charge sequence of length $l_{sql}$
Draw the number of charge events to add. $U(1,N_{EV})$
Draw charge events from ACN-DATA
Add the drawn charge events at random start position to a series of length $l_{sql}$. $U(1, l_{sql})$ This is a charge sequence.
For each customer with a smart meter series of length L:
Split the smart meter series into a subset of series with fixed length $l_{sql}$ at random start positions. $U(1, L - l_{sql})$
For each fixed length series:
Are there any missing values?
yes
DELETE
no
Add charge event? Drawn binomial with propbaility $p_{EV}$. $B(1,p_{EV})$
no
yes
+
Return
Feature vector: Fixed length series with no EV-chargin added.
Ground truth: Labels have all zero values
Feature vector: Fixed length series + Charge sequence
Ground truth: The label is one where the charge sequence is above 1kW.

Figure 5.5: Flow chart illustration of how a labeled dataset is generated from two data sources.

## 5.3 Datasets

For clarity in the experiments we describe the different datasets used and that will be referenced in the experimentation results. An overview of the datasets using EIDSIVA is shown in Table 5.3.

Table 5.3: The different datasets when using the Eidsiva datasource. These data sets will be referenced during experimentation.

| | | No registered EV | | | Registered EV | |
|---|---|---|---|---|---|---|
| Category | | Train | Validation | Test | EV | Before EV |
| Number of smart meter series | | 1892 | 924 | 946 | 2487 | 1038 |
| EIDSIVA | CLUSTERING | ✓ | ✓ | ✓ | ✓ | |
| | EXPLORE | | ✓ | ✓ | ✓ | ✓ |
| ACN+EIDSIVA (Labled) | Train | ✓ | | | | |
| | Val | | ✓ | | | |
| | Test | | | ✓ | | |

### 5.3.1 EIDSIVA CLUSTERING

EIDSIVA CLUSTERING uses the unlabeled data from Eidsiva and will be used when clustering. The data groups in this dataset are Train, Validation, Test, and EV (see Table 5.3). The Data groups Validation and Test are merged into a single group that will be referenced as **no EV**. When clustering the clustering is performed with respect to no EV and EV. For further validation of cluster assignment, the Train set will be used, with the expectation that the train set will be assigned to cluster whit lower concentration of EV owners.

### 5.3.2 EIDSIVA EXPLORATION

EIDSIVA EXPLORE will be used to investigate whether our proposed model for EV event detection is able to detect EV charging. Therefore, the Train set will be excluded when exploring, see Table 5.3 for the data groups included. Note that this data set does not contain ground truth about EV charge events.

### 5.3.3 ACN+EIDSIVA

ACN+EIDSIVA will be used to train, validate, and compare/test the proposed model for EV detection. This dataset is labeled from combining ACN with EIDSIVA data groups that do not have registered EV owners and have a sampling rate of one sample per hour.

### 5.3.4 ACN+UK-DALE

ACN+UKDALE is a labeled data set generated from combing ACN and the UK-DALE data sources. This data set has the lowest sampling rate of one sample for every sixths second. This data set will be used to explore EV detection for different sampling rates.

## 5.4 Summary

We proposes combining different data sources to generate labeled datasets, and further define the different dataset used during experimentation. To summarize the data set EIDSIVA CLUSTER will be used for the problem EV load profiling, while ACN+UKDALE, and ACN+EIDSIVA will be used for training and comparing the proposed models for EV event detection. For EV event detection, the models are also compared by using the unlabeled dataset EIDSIVA EXPLORE.

In the next chapter we will present the EV load profiling results where we use the EDIDSIVA CLUSTERING dataset.

# Chapter 6

# EV load profiling: Results

As previously stated, the motivation for clustering is to investigate whether we are able to capture some EV owners in separate clusters, which further can be used as comparisons with the proposed methods of EV event detection. Since the main focus of this thesis has been the problem of EV event detection, the approach when clustering has been rather exploratory and highly biased with a single goal in mind. Therefore the approaches used in this chapter use an exploratory methodology that we suggest should further be improved upon in later projects.

## 6.1 GMM of weekly load profiles

When performing the described clustering algorithm in Chapter 4, the window size when detrending is set to 2 weeks and the Hampel filter has a threshold of 3 standard deviations.

#### Dataset

Since the aim is to compare clusters with the supervised methods, the train group is excluded when optimizing the Gaussian mixture model, and EIDSIVA CLUSTERING dataset is used (see Chapter 5.3). The clusters are being fitted with respect to 4357 consumer loads, where 57% has registered an EV (EV). This is a relative balanced dataset, if we assume not all EV owners charges their EV at home.

The data group "Train" will be used as a weak validation whether we are able to capture some uniqueness to EV owners since this group is assumed not to contain any customer that charge their EV at home.

## 6.2 Experimentation

When performing clustering, the aim is to find individual clusters that capture some uniqueness to EV owners. To measure this "uniqueness," we use the concentration of EV owners (the number of EV owners relative to the total number of customers in a cluster) as an important metric and will be used to determine a sufficient amount clusters.

The implemented method using sklearn allows for four assumptions to the covariance matrix (CM):

- tied: "All components share the same general covariance matrix".
- full: "Each component has its own general covariance matrix."
- diag: "Each component has its own diagonal covariance matrix."
- spherical: "Each component has its own single variance."

To derive a good choice of components and CM, we conduct the following experiments for all three features:

- We perform GMM from 2 to 8 clusters for all four options of CM.

The clusters with the highest concentration of owners as well as the number of customers within this cluster will determine whether we have achieved a good separation. There is also a tradeoff between the number of clusters and EV concentration.

#### Results

The results show that all normalized features were able to generate clusters with a high concentration of EV owners. From the experimental results, we have chosen following settings:

- Mean features: 4 components with general diagonal CM for all components.
- Kurtosis features: 3 components with diagonal CM settings.
- Skewness features: 3 components with diagonal CM settings.

and for the non-normalized features:

- Mean features 3 components with general diagonal CM for all components.
- Kurtosis features: No separation
- Skewness features: No separation

In the next section, we will present the final clustering results for all feature spaces that were able to separate some EV owners. To ensure stable clustering, we saw that the EM algorithm reach a stable convergence over 100 initialization. In general, when performing clustering, we saw stable convergence when fitting all models. In the next sections, the final clustering results will be presented.

## 6.3 Final clustering results

Table 6.1 summarizes the final clustering results and Figure 6.2 to 6.4 shows the detailed clustering results for each of the features. These figures is presented at the end of this section.

Table 6.1: Cluster assignment for each data group. The clusters with high concentration of EV owners is marked with bold text.

| | Cluster assignment for each data group (% | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Mean | | | Normalized Mean | | | Normalized Kurtosis | | | Normalized Skewness | | |
| cluster | EV | No EV | Train | EV | No EV | Train | EV | No EV | Train | EV | No EV | Train |
| 1 | **45.0** | **4.4** | **4.9** | 16.3 | 42.4 | 40.9 | **37.9** | **7.3** | **8.2** | **30.7** | **7.3** | **7.1** |
| 2 | 44.2 | 39.1 | 40.7 | 15.9 | 30.2 | 29.9 | 28.6 | 50.9 | 50.6 | 42.6 | 37.8 | 38.4 |
| 3 | 10.8 | 56.5 | 54.4 | **33.4** | **6.5** | **6.2** | 33.5 | 41.2 | 41.2 | 26.7 | 54.9 | 54.4 |
| 4 | | | | 34.4 | 20.9 | 22.9 | | | | | | |

### GMM of means

Results show clustering for normalized, and non-normalized mean features were able to generate a cluster with a high concentration of EV owners. The non-normalized means that managed the best EV separation of all feature spaces show that some EV owners generally have higher loads during the evening and afternoon. This result is evident in the heat map in Figure 6.2.

When clustering with respect to the normalized mean, we can see that some EV owners have a shift of higher energy consumption during the later evening and extending into the night. This is evident in both the cluster means and the heat map in Figure 6.2.

### GMM of kurtosis

The clustering results of the kurtosis features shows that only the normalized features were able to form a cluster with a high concentration of EV owners. The normalized clustering results of kurtosis shown in Figure 6.3, show that some EV owners have a larger fluctuation with peak kurtosis in the middle of the day, and lowest during the afternoon. By looking at the cluster assignment and feature space, we can see that we were able to capture a clear trend between some EV registered customers.

### GMM of skewness

The normalized skewness features clustering results are shown in Figure 6.4. As for kurtosis, the cluster with a high concentration of EV owners has prominent peeks during the night, and larger fluctuations in general. It is indicating that during the night, the distribution is shifted towards higher energy consummation than its mean.

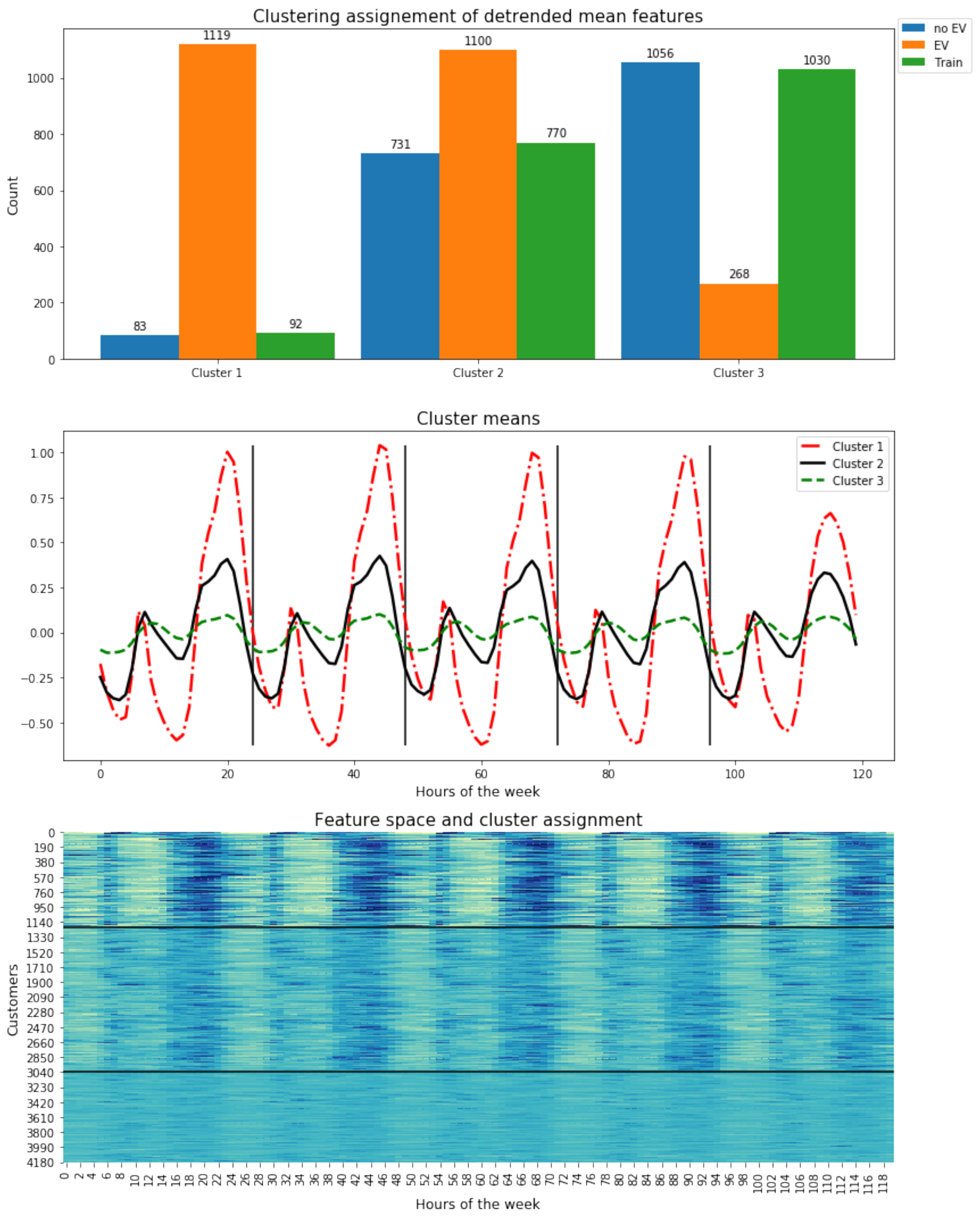


Figure 6.1: Final clustering results of the detrended mean features. We can see a clear trend of some EV owners having a higher peek demand, in already high demand periods. This is evident in both the cluster means, and the heatmap of the feature space.

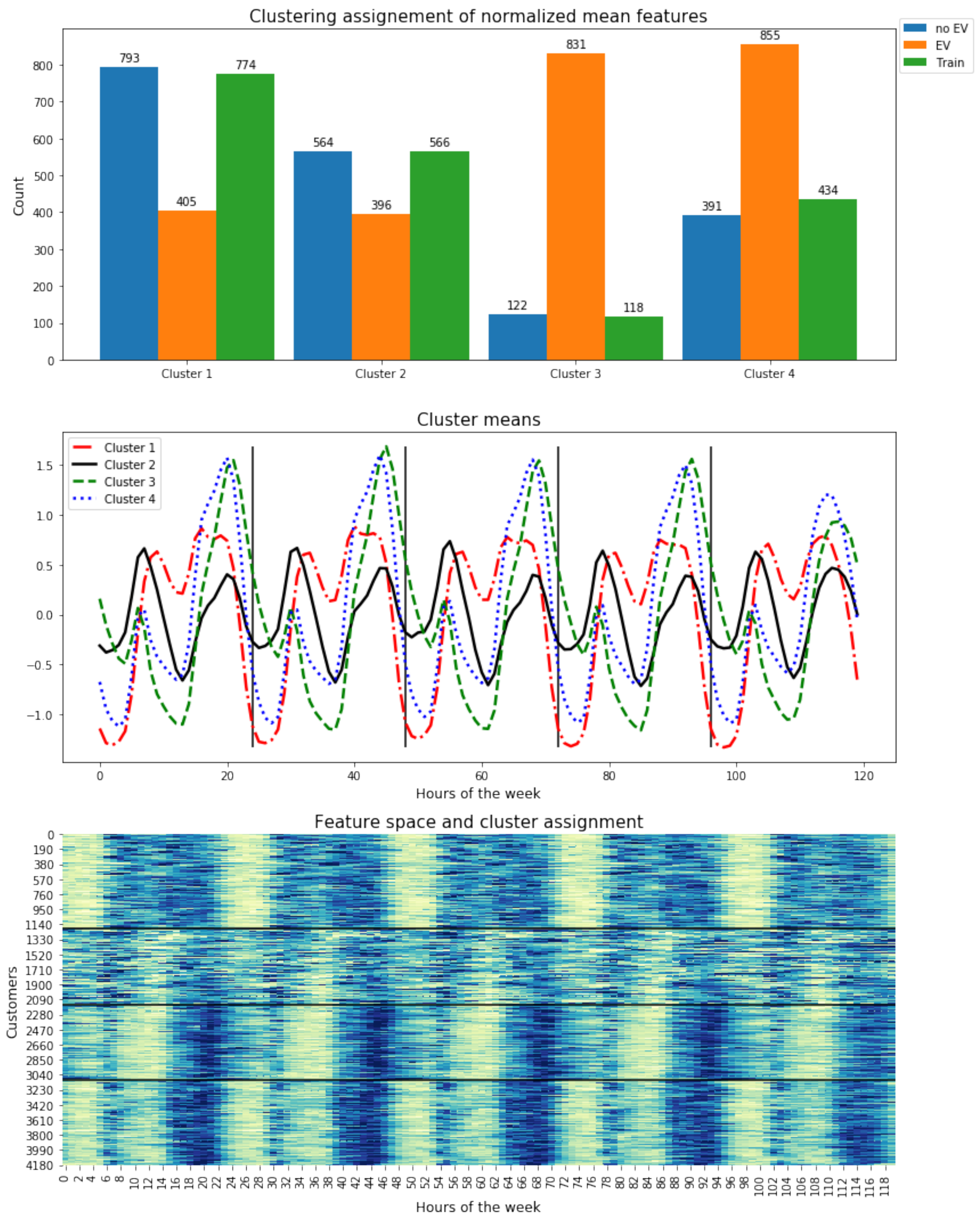


Figure 6.2: Final clustering results of the normalized mean features. We can see that the cluster with a high concentration of EV owners has an shift of higher power consumption during the night (green dotted line), and an larger peek consumption during the afternoon.

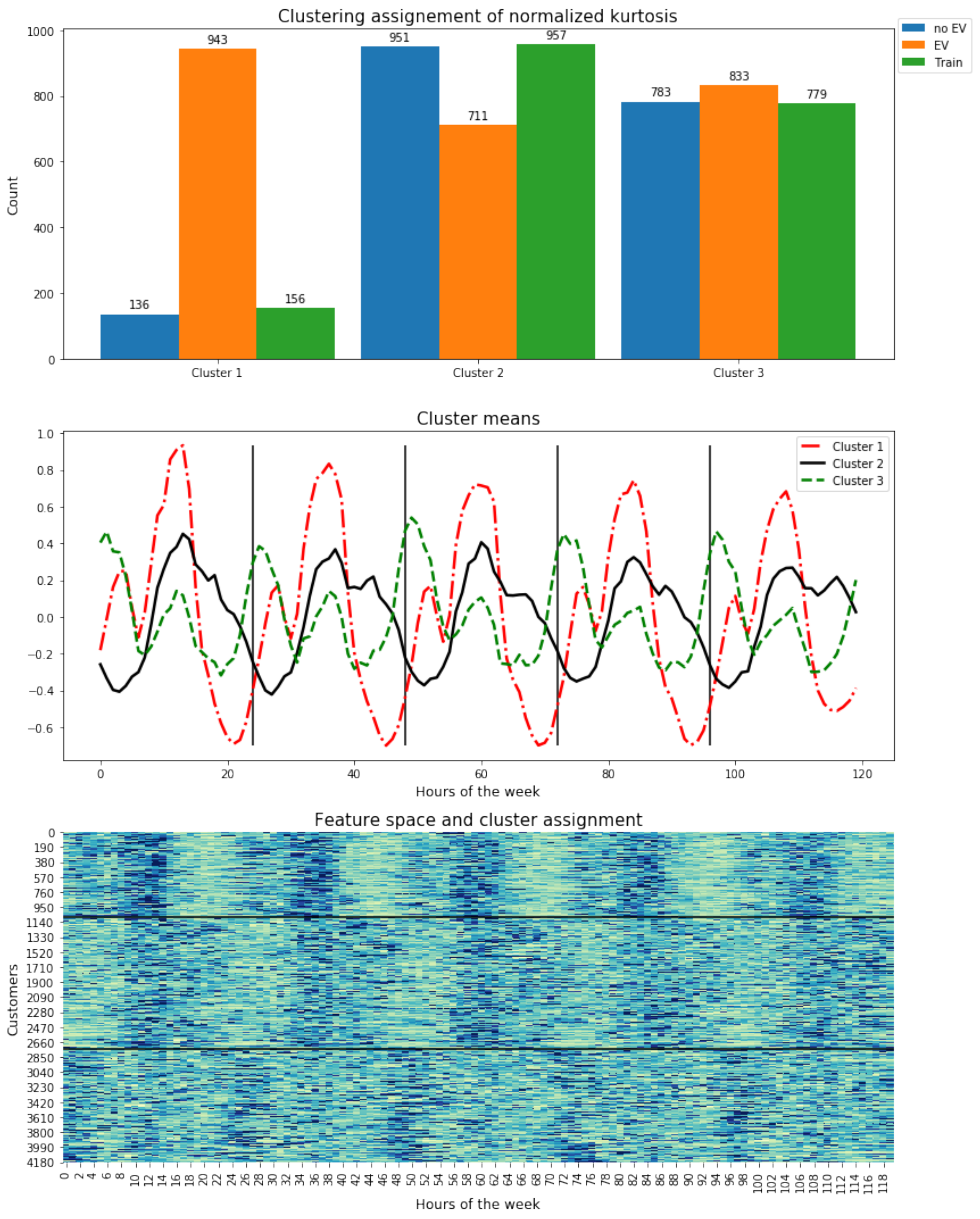


Figure 6.3: Final clustering results of the normalized kurtosis features from EIDSIVA CLUSTERING.

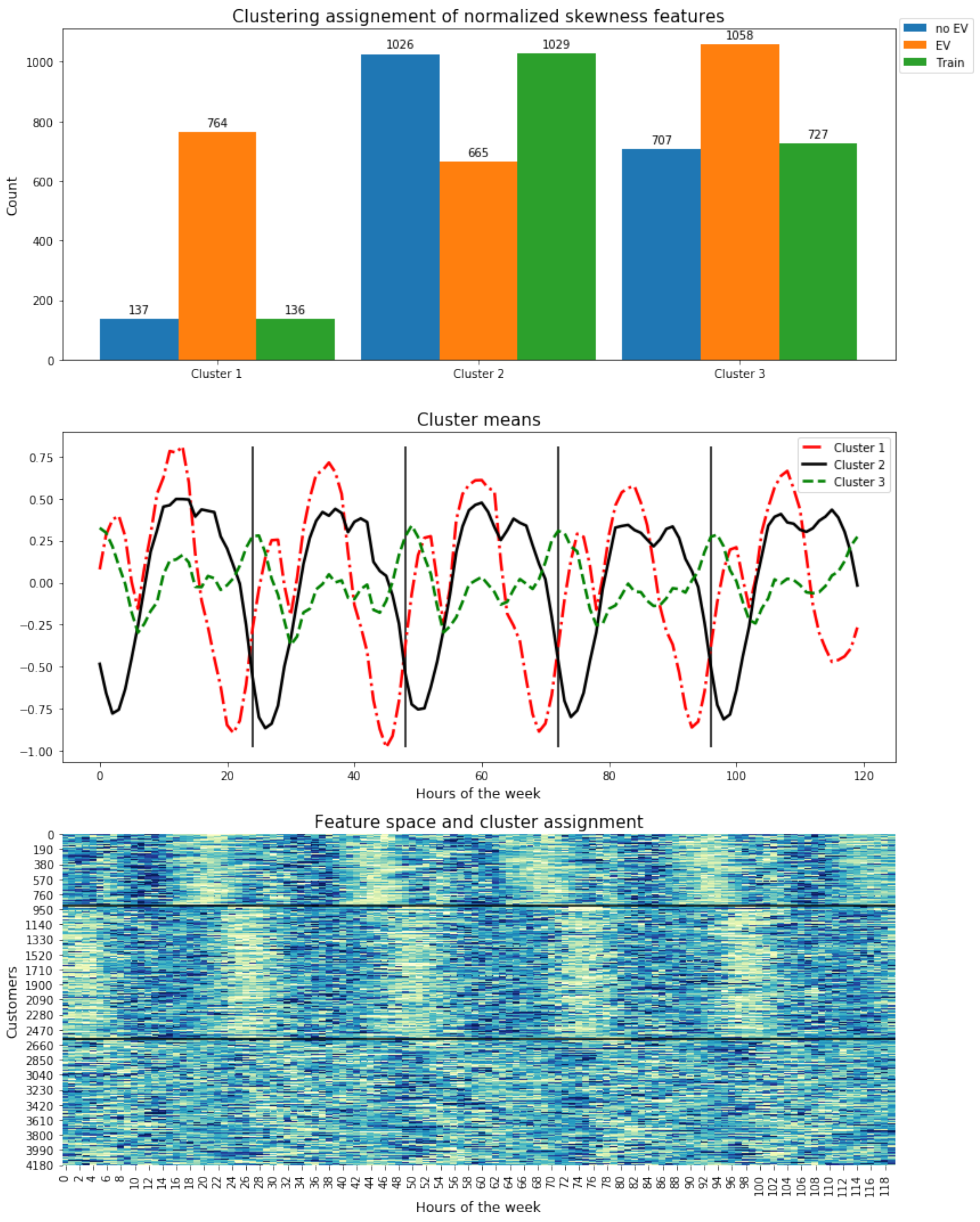


Figure 6.4: Final clustering results of the normalized skewness features from EIDSIVA CLUSTERING. There is a clear trend of two prominent peeks, in the cluster with a high concentration of EV owners.

## 6.4 Summary and discussion

In this chapter, we have shown that the proposed method of clustering, are able to capture clusters with a high concentration of EV owners. We have learned that some EV owners generally have higher peek loads in high demand periods. We also saw that the normalized features were able to separate EV owners into distinct clusters, with a shift of higher energy consumption during the night. This assumption is further backed up by a survey that reported people charged their EV during the night-time[14].

Further, we showed that all the proposed feature spaces can be used in EV load profiling applications. This verifies the workings in [7][8].

Before presenting the experimental results from EV event detection, the author wants to address some aspects we might consider as further work when clustering with the aim of discovering EV charging.

- The weekly profiles show a similar trend for all days of the week: Is it possible to reduce the dimensionality of the profiles e.g., weekday profiles?
- GMM assumes the grouping is separated by multivariate Gaussian distribution, which is not necessarily the case. The choice of GMM was chosen since it gave better separation than K-means (not reported). However, we suggest as further work different choice of clustering algorithms.
- Merging the mean, kurtosis, and skewness and perform clustering from a combined feature space. Since we observed the different features spaces separated different EV owners(not reported).
- Further analysis of the robustness and generalization of these clustering results.

# Chapter 7

# EV event detection: Experiments and Results

In this section, we present the experimental results from the proposed models for EV event detection. The experiments were conducted by using the dataset described in Chapter 5. Further, this chapter is divided into seven sections:

- First, we describe the validation metrics used during experimentation.
- Second, we explore the CNN+DENSE model for varying sampling rates.
- Third, we experiment and tune the proposed CNN+autoencoder architectures.
- Fourth, we explore tuning of the modified UTime for EV detection.
- Fifth, we compare the tuned models on the same generated labeled dataset.
- Sixth, compare the tuned model predictions on an unlabeled test data set.
- Seventh, Investigate whether the overall best performing model, are able to determine the time of registering an EV.

## 7.1 Validation metrics

As mentioned in Chapter 3, validation of unbalanced data needs care full considerations other than traditional accuracy scoring. In this section we aim to described the different validation metrics used, for the problem of EV event detection, which is precision, recall, f1 score, and ROC.

### 7.1.1 Precision and recall

To get further insight into how the classification error occurs, we define the four outcomes for a binary classifier:

- True positives (TP): Number of labels correctly classified as 1,
- True Negative (TN): Number of labels correctly classified as 0,
- False positives (FP): Model classifies 1 but the true label is 0,
- False Negatives (FN): Model classifies 0, but the true label is 1,

where none EV charge events are labeled 0 and charging events have label 1.

From these measurements, we can define precision and recall as follows

$$\text{Precision} = \frac{TP}{TP + FP} \tag{7.1}$$

$$\text{Recall} = \frac{TP}{TP + FN}. \tag{7.2}$$

Precision and recall are useful for imbalanced data since the two measurements give a description of how the miss-classification occurs, with emphasis on positive predictions. Precision describes the rate of true positives and recall the coverage of actual positive samples. Ideally, we want a classifier with both high precision and high recall. However, in most real-life scenarios, there is a tradeoff between precision and recall. One way to represent such tradeoff is in a precision-recall curve (PRC), where precision and recall are plotted on along two different axis for a range of prediction thresholds.

### 7.1.2 Receiver operating characteristic (ROC)

Another metric for evaluating classifiers is a receiver operating characteristic (ROC)[67][34]. For the case of binary classifiers that outputs a probabilistic confidence score (values between zero and one), the prediction is made by choosing the most likely class or give a positive prediction if the confidence score exceeds a certain threshold. A ROC graph is a plot of the true positives rate (TPR) and false positive rate (FPR) along the axis over a range of thresholds. TPR and FPR is defined as

$$\text{True positive rate (TPR)} = \frac{TP}{TP + FN} \tag{7.3}$$

$$\text{False positive rate (FPR)} = \frac{FP}{FP + TP}. \tag{7.4}$$

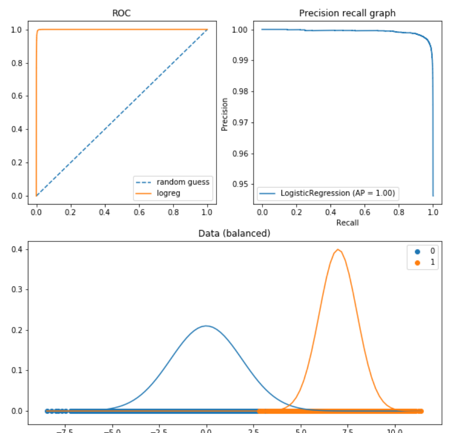
Note that TPR has the same definition as as recall.

### 7.1.3 F1 score

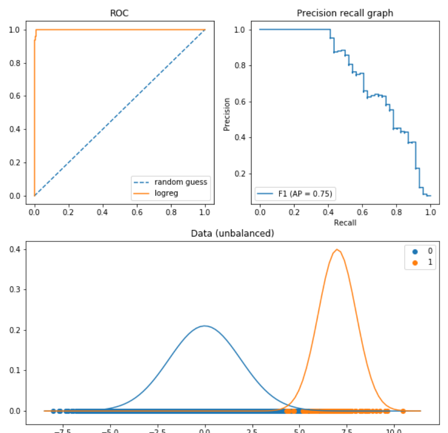
Closely related to PRC is $F_1$ score which is defined as the harmonic mean of precision and recall when both are evenly weighted. F1 score is defined as

$$F_1 = 2(\frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}). \tag{7.5}$$

It has been shown that for imbalanced data that precision recall curve works better than other validation metrics such as receiver operating characteristic (ROC) curve[34]. Therefore in this thesis precision recall curve and $F_1$ score will be our main metric when evaluating the models. Figure 7.1 shows how ROC graph and PRC may differ between an unbalanced and balanced data-set. Note that there is no to little difference between ROC graphs, however the PRC shows a the difference in performance.



(a) Balanced data



(b) Unbalanced

Figure 7.1: ROC graph, and PRC of a logistic regression classifier preformed on both a balanced (a), and unbalanced dataset (b). The data is drawn from two normal distributions and has same mean and variance for both (a), and (b). In (a) the both distributions has 100000 sample points. For (b) there are 100000 samples drawn with zero label and 100 data points with label 1.

## 7.2 Model performance for lower sampling rates

To the author's knowledge, most research into EV event detection has been conducted for a sampling rate of 1 minute (see summary in Table 2.1). However, in this thesis, the main goal is to detect EV charge events from hourly smarter meter data.

To address whether this problem is feasible to solve, we explore the performance of the CNN+DENSE model for varying sampling rates on a generated data set. The motivation for this experiment is to conclude whether this problem is feasible to solve and how different sampling rates effects performance.

### Dataset

To achieve a data set with varying sampling rates, we generate a labelled dataset by combining the data sources UK-DALE and ACN-DATA, this data set will be referred to as ACN+UKDALE (see Chapter 5.3). The sampling rates explored is one sample every 1, 30, and 60 minutes. When generating the labels, the maximum number of charge events is two per day. We have chosen a fixed duration of two weeks resulting in input lengths of 20160, 672, and 336 for the sampling rates of 1, 30, and 60 minutes respectively.

The probability of an input sequence having EV charging present is about 96%, a high probability compared to what's expected from a real-life data set. The last 30% of all households in UK-DALE is reserved as a test set.

### Training

When training, the models are optimized with respect to mean square error loss, and ADAM optimizer with a learning rate of $1 \cdot 10^{-4}$ and default $\beta$ parameters is used. With this setup, two experiments where conducted:

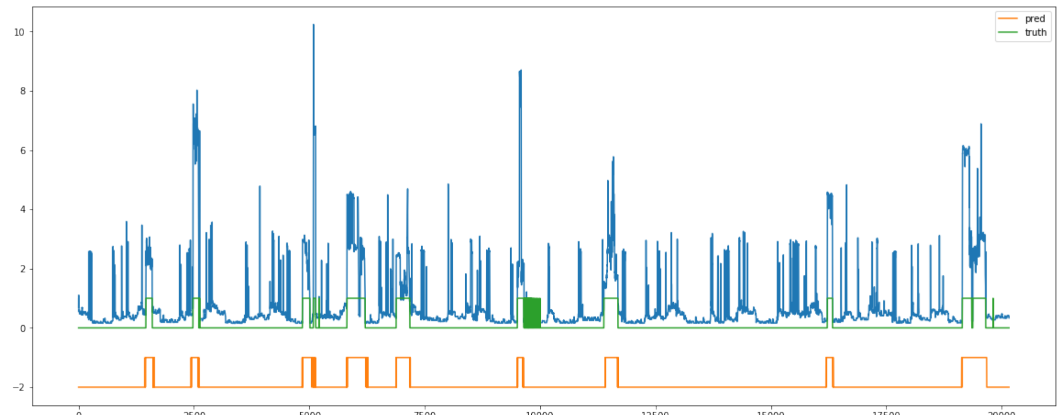
- Train and evaluate the CNN+DENSE model for all three sampling rates (1, 30, and 60 min) for a sequence length of two weeks.

### Results

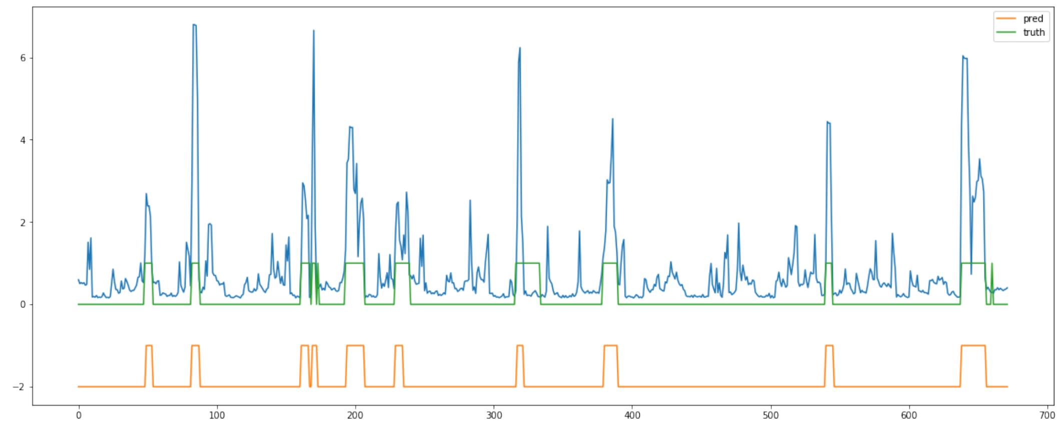
The result is reported in Table 7.1 and an example of predicted time series for different sampling rates is shown in Figure 7.2. The result shows that it is possible to detect EV charging for all sampling rates, and the worst F1 score was at a sampling rate of 1 per minute. Figure 7.2 shows why this may be the case. For lower sampling rates, we can see that prominent peaks with short duration are removed when resampled, while high loads with a longer duration are still visible. Making charge events easier to detect. However, also note that the CNN+DENSE model for 1-hour sampling rate was not able to detect the first event with a lower peak.

Table 7.1: CNN+DENSE model performance for different sampling rates (sr), and corresponding input lengths (sql), trained an validated on ACN+UKDALE.

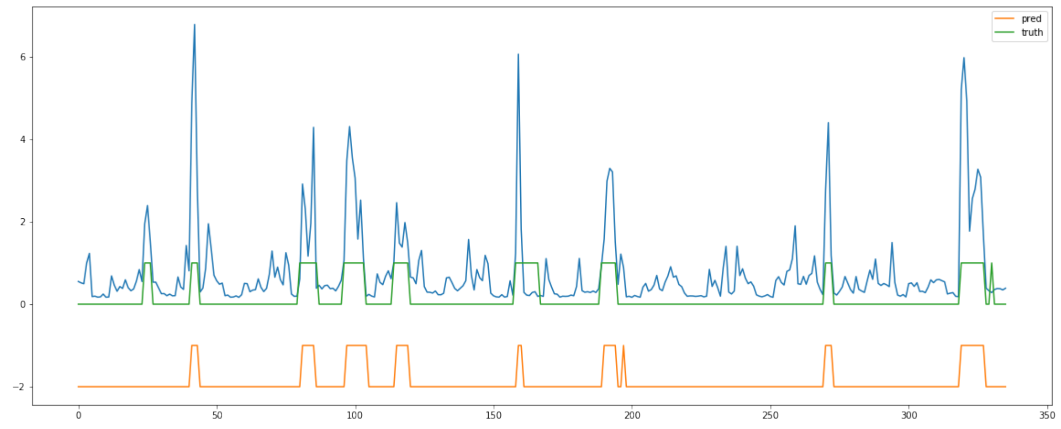
| | Accuracy | | Precision | | Recall | | F1 score | |
|---|---|---|---|---|---|---|---|---|
| sr / sql | Train | Test | Train | Test | Train | Test | Train | Test |
| 1 min / 20160 | 0.98 | 0.98 | 0.77 | 0.76 | 0.93 | 0.94 | 0.84 | 0.84 |
| 30 min / 672 | 0.98 | 0.98 | 0.82 | 0.83 | 0.94 | 0.96 | 0.87 | 0.89 |
| 60 min / 336 | 0.98 | 0.98 | 0.80 | 0.82 | 0.95 | 0.95 | 0.87 | 0.88 |



(a) Prediction of CNN+DENSE for 1 minute sampling rate.



(b) Prediction of CNN+DENSE for 30 minute sampling rate.



(c) Prediction of CNN+DENSE for 60 minute sampling rate.

Figure 7.2: Predictions using CNN+DENSE of the same two week load series for different sampling rates. The green line indicates the ground truth while the orange line is the predicted EV charging.

### Summary and discussion

The experiments demonstrate that it is possible to detect EV charging from hourly smart meter data. The observed predictions for lower sampling rates showed that high loads with short duration's got flatten out when resampled, making it easier to visually see the EV charge events that generally have a longer duration. This effect may also occur when resampling the charge sequences from ACN-DATA, making the charge signals to be discovered less noisy and more similar to square waveforms.

It should also be emphasized that these results are from data generated from five residential homes in the UK, with a large number of charge events added. These results may not reflect the prediction from real-life data.

We observed that reducing the concentration of EV events, also dramatically changed the performance, possible due the high expectation that an EV event was present. This observation is taken into account in the next experiments.

The main conclusion from this initial experiment is that it is possible to detect EV charging from hourly smart meter data.

In the next experiment, we explore performance of different CNN+Autoencoder architectures based on previous research, on the much larger dataset ACN+EIDSIVA.

## 7.3 Experimentation with different CNN + Autoencoder architectures

In this section, we aim to explore the three different CNN+Autoencoder architectures and train them on the ACN+EIDSIVA dataset. The models are summarized in Figure 4.5, where the first and last dense layer can be modified to handle different input lengths (see Chapter 4.2).

### Dataset

For this experiment, all three (CNN+AUTO, CNN+ENCODER, CNN+DENSE) models is trained and evaluated on a generated data set from the ACN+EIDSIVA dataset. The labeled dataset is generated according to Chapter 5.2. The probability of a series contains an EV is set to 0.2, and the maximum number of charge events is set to 2 per day.

Table 7.2: When generating data, as the input length increases, the data size will decrease since segments containing missing values will be deleted. This table reports the percentage of hours lost, according to the sequence length. The percentage of hours lost is compared with the sequence length of 24 hours.

| Input length | Relative hours of positive labels | % of lost hours |
|---|---|---|
| 24 | 0.042 | 0 |
| 48 | 0.035 | 0.9% |
| 72 | 0.033 | 1.7% |
| 168 | 0.030 | 4.4% |
| 336 | 0.03 | 8.5% |
| 504 | 0.029 | 11.9% |
| 672 | 0.029 | 15.4% |
| 840 | 0.029 | 18.2% |

The validation set is identical for the same sequence length since it is generated from the same random seed. Therefore, models with the same input length are comparable. However, for different sequence lengths, the validation set differs.

When the labeled data set is generated, sequences containing missing values will be deleted. A consequence of this, is that for longer sequence lengths the data size shrinks since more of the generated sequences gets deleted. Table 7.2 addresses this issue, showing that as the sequence length increases the number of data points gets smaller. Further, from Table 7.2 shows that the number of positive labels also varies for each input length, making direct comparison more difficult.

### Training

When training, both Dice loss and binary cross-entropy (BCE) loss is considered, and the parameters are optimized with ADAM optimizer having a fixed learning rate of $1 \cdot 10^{-4}$ . The reported results are from the model state with the lowest validation loss after 1000 epochs of training. Since the CNN+Autoencoder architectures increase its number of trainable parameters as the input length increases, a maximum input length of one week (168 hours) is chosen.

In the next sections the experimental results is reported and briefly discussed.

### Results

Table 7.3 shows the experimental results for the different architectures trained with BCE loss and Table 7.5 summarises the number of trainable parameters for each model. The results show that CNN+AUTO performed best for an input length of 24, CNN+ENCODER with an input length of 72, and CNN+DENSE performed the best with an input length of 168. The highest F1 score was achieved by the CNN+DENSE architecture with an F1 score of 0.8906, followed closely by the CNN+ENCODER with an F1 score of 0.8870. The CNN+AUTO was by far the worst-performing model with overall the worst F1 score for all input lengths.

As a further experiment, each of the best performing models were trained with respect to none regularised Dice loss. The result is shown in Table 7.4 and shows worse or close to no improvements with respect to BCE loss.

Table 7.3: Experimental result of different CNN+Autoencoder architectures trained with BCE loss. The reported results is from the ACN+EIDSIVA validation set, with lowest loss after 1000 epochs.

| Models | CNN+AUTO | | | CNN+ENCODER | | | CNN+DENSE | | |
|---|---|---|---|---|---|---|---|---|---|
| Input length | precision | recall | F1 | precision | recall | F1 | precision | recall | F1 |
| 24 | 0.8410 | 0.7438 | 0.7894 | 0.8853 | 0.8268 | 0.8550 | 0.8688 | 0.7984 | 0.8321 |
| 48 | 0.8196 | 0.7257 | 0.7698 | 0.8972 | 0.8616 | 0.8790 | 0.8902 | 0.8365 | 0.8626 |
| 72 | 0.7938 | 0.6814 | 0.7333 | 0.9055 | 0.8692 | 0.8870 | 0.9022 | 0.8686 | 0.8851 |
| 168 | 0.6909 | 0.4538 | 0.5478 | 0.8893 | 0.8029 | 0.8439 | 0.9020 | 0.8795 | 0.8906 |

Table 7.4: The best performing architectures trained with Dice loss. The results shows no improvements over BCE loss.

| Model | Input length | Precision | Recall | F1 score |
|---|---|---|---|---|
| CNN+AUTO | 24 | 0.7921 | 0.7842 | 0.7881 |
| CNN+ENCODER | 72 | 0.8772 | 0.8616 | 0.8694 |
| CNN+DENSE | 168 | 0.8733 | 0.8462 | 0.8595 |

### Summary and discussion

The performance for different sequence lengths is varying, with the models having different ”optimal” inputs lengths. For CNN+AUTO that showed its best performance when lowering the input length. This results could indicates that the

Table 7.5: Number of trainable parameters for the different CNN+Autoencoders. The reported result is from ACN+EIDSIVA validation set.

| | Number of trainable parameters | | |
|---|---|---|---|
| Input lengths | CNN+AUTO | CNN+ENCODER | CNN+DENSE |
| 24 | 5738 | 326504 | 409508 |
| 48 | 9962 | 529280 | 457532 |
| 72 | 14186 | 732056 | 505556 |
| 168 | 31082 | 1543160 | 697652 |

embedding in the encoder is to small to store sufficient information to decode to the predictive output space when the input length is increased.

The two main differences between CNN+ENCODER and CNN+AUTO, is the complexity of the dense layers, and the number of convolutional filters. We saw that CNN+ENCODER could handle longer input lengths than CNN+AUTO, this might indicate that the higher complexity in the dense layer, and no decoder allowed for better performance. However we cant say for sure, since the number of filters is also different.

At last the best performing model where the initial proposed CNN+DENSE, that is similar with CNN+ENCODER in the dense layers, however have two less filters in the convolutional layers, with max-pooling in between. This means that CNN+DENSE has the highest receptive fields, in the feature extraction phase, and it seems like the model benefited from this. Since it showed best overall performance of the three architectures, and showed better performance for longer input lengths.

To summarize the best configurations of the different architecture where:

- CNN+ENCODER with input length of 72
- CNN+AUTO with input length of 24
- CNN+DENSE with an input length of 168

These models will be used for further comparison with the proposed modified UTime for EV detection (Chapter 4.2.2) discussed in the next paragraph.

## 7.4 UTime for EV event detection: Experimentation

In this section, we will explore how the depth and input length of the modified UTime for EV detection described in Chapter 4.2.2 will affect performance on the generated ACN+EIDSIVA dataset.

### Dataset and training

The experiments are done by the generated ACN+EIDSIVA dataset; the probability of a series containing EV charging is set to 0.2, and the maximum charge events are two per day. When training the model, the parameter is optimized with respect to the Dice loss and by using ADAM optimizer with a fixed learning rate of $1 \cdot 10^{-4}$ and default $\beta$ parameters. The model weights are initialized by Xavier uniform initialization.

Note as for the CNN+Autoencoder experiments, there is a certain randomness when generating training and validation sets and therefore direct comparison should be taken with a grain of salt. However, for each input length, the data is generated from the same seed and, therefore, comparable.

### Results

The experimental results are shown in Table 7.6. Generally, increasing the depth improves the performance, and the model performs its best with 5 encoder blocks. The results also indicate that choosing a longer input sequence might improve performance. But note, direct comparison is more difficult since the concentrations of positive labels and the size of the dataset differs for each input length. This is summarised in Table 7.2.

As a final experiment of the proposed UTime we investigate if the choice of evenly weighted binary cross-entropy (BCE) loss function gives better results as it did for the CNN+Autoencoder architectures. The comparison of UTime trained with BCE and dice loss is summarized in Table 7.7, showing that BCE loss gave similar performance. Figure 7.3 shows the loss during training. These plots show the general trend of little overfitting occurring, that we observed for all models.

Table 7.6: Experimental results of UTime with different depths and sequence lengths. The data set used is ACN+EIDSIVA validation.

| Encoder blocks | 2 | | 3 | | 4 | | 5 | |
|---|---|---|---|---|---|---|---|---|
| Sequence length | loss | F1 | loss | F1 | loss | F1 | loss | F1 |
| 24 | 0.15145 | 0.84887 | 0.14349 | 0.85679 | 0.14275 | 0.85754 | 0.14286 | 0.85742 |
| 48 | 0.14836 | 0.85198 | 0.11592 | 0.88437 | 0.10911 | 0.89116 | 0.10909 | 0.89110 |
| 72 | 0.14726 | 0.85321 | 0.10522 | 0.89507 | 0.103822 | 0.89813 | 0.09262 | 0.90759 |
| 168 | 0.14557 | 0.85473 | 0.10342 | 0.89676 | 0.08150 | 0.91870 | 0.07756 | 0.92260 |
| 336 | 0.14225 | 0.85820 | 0.09580 | 0.90460 | 0.07868 | 0.92142 | 0.06462 | 0.93541 |
| 504 | 0.14122 | 0.85921 | 0.09509 | 0.90510 | 0.07902 | 0.92115 | 0.07010 | 0.92999 |
| 672 | 0.13991 | 0.86056 | 0.09280 | 0.90765 | 0.07497 | 0.92506 | 0.06856 | 0.93179 |
| 840 | 0.14555 | 0.85506 | 0.09259 | 0.90742 | 0.07692 | 0.92336 | 0.06548 | 0.93463 |

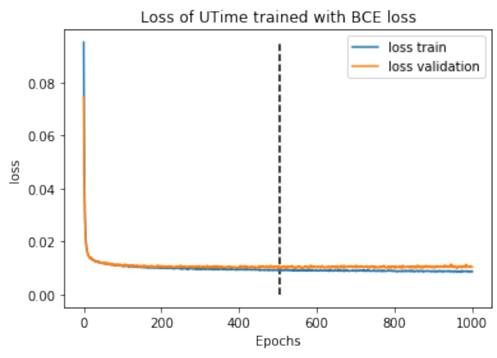
Table 7.7: Comparing difference loss function when UTime with 5 encoder blocks is trained with an input length of 336. The reported result is from ACN+EIDSIVA validation. The difference is small but Dice loss show better balance between precision and recall.

| Loss function | Precision | Recall | F1 score |
|---|---|---|---|
| BCE | 0.9286 | 0.9444 | 0.9364 |
| Dice | 0.9391 | 0.9318 | 0.9354 |

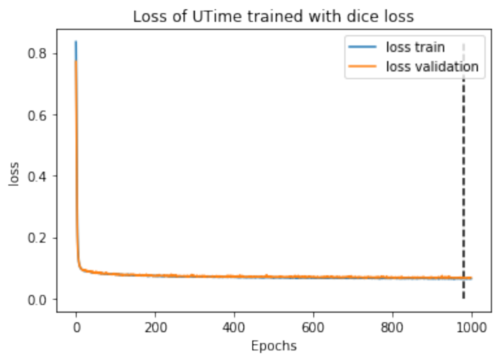
**Summary and discussion**

We saw that increasing the depth of the modified UTime gave better results. This might indicate that our proposed modification to UTime could have benefited with larger dilation and pooling windows to increase the receptive field. These modifications should be considered, especially if the aim were to have fewer encoder blocks. However, the case might also be that the problem benefits from a more complex model since shorter input lengths also benefitted from more encoder blocks.

Overall the results are promising, and from the generated validation set, it seems that UTime outperforms the previously proposed models. In the next section, we aim to compare UTime with 5 encoder blocks, trained with an input length of 336, with the best performing CNN+autoencoder architectures.



(a) Binary cross entropy



(b) Dice loss

Figure 7.3: Training and validation loss for Utime with 5 encoder blocks and input length of 336. Both the training with BCE and dice loss is shown in separate plots. The dashed vertical line indicates the epoch with the lowest validation loss on the ACN+EIDSIVA validation set, 504 for BCE (a) and 982 for dice (b).

## 7.5 Comparing best performing models on the test set

From experimentation, we concluded the best performing models where

- Utime with 5 encoder blocks and input length of 336
- CNN+ENCODER with input length of 72
- CNN+AUTO with input length of 24
- CNN+DENSE with an input length of 168

In this section, we compare our tuned models on the same generated labeled ACN+EIDSIVA test set and further explore model predictions on the unlabeled EIDSIVA EXPLORATION set.

### 7.5.1 Test dataset

Since the models take different input lengths, the final comparison is done by generating long smart meter sequences with EV charging. The method for generating ACN+EIDSIVA test set is similar to the previous experiments with the modification that EV charging is added on top of the entire smart meter signal from a customer, while missing values is ensured to be preserved.

The generated ACN+EIDSIVA test set has 946 customer loads and a total of 15335800 data points, where 0,4% is missing values, and 6% contains EV charging. For the test set, half of the customers has added EV charging, with between 0.2 and 1.5 charge events per week.

### 7.5.2 Comparison results

The ROC, PRC and the F1 curve of the models is shown in Figure 7.4, further summary of validation metrics and computation time is shown in Table 7.8.

#### Validation metrics

The results show that UTime has the best F1 score. Lowering prediction thresholds shows slight improvements for all models. The CNN+AUTO is the poorest performing model and shows a more rapid decline in the PRC and F1 curve, indicating it is more wrong in its most confident predictions. This observation is similar to whats reported in [6]. CNN+ENCODER and CNN+DENSE show very similar performance with some tuning benefits when the threshold is lowered. CNN+ENCODER is the model with the lowest number of false positives (FP) closely followed by UTime. Even though UTime shows overall best performance, the inference time is significantly higher with the current implementation. This is mainly due to the larger input length when predicting a long smart meter sequence, and that UTime is more memory-bound because of storing of the skip connections.

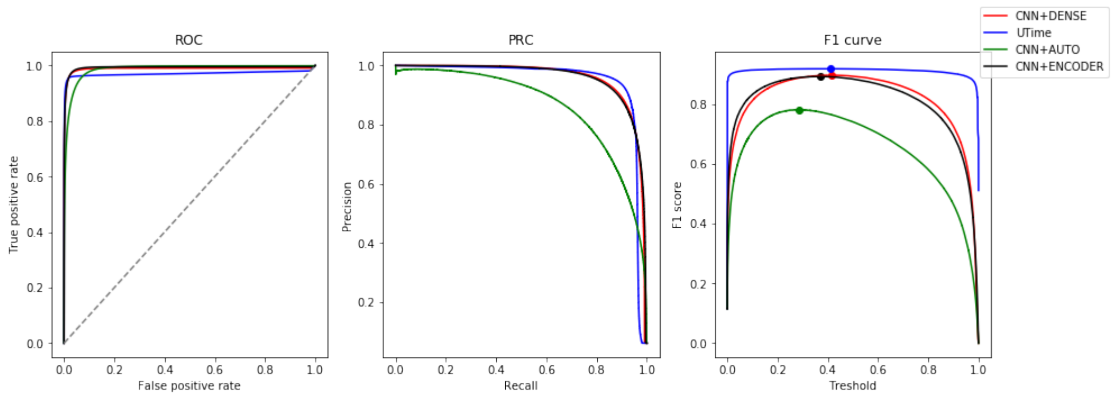


Figure 7.4: ROC, PRC and F1 curves for all models on the same generated ACN+EIDSIVA test set, where the highest F1 score is marked.

Table 7.8: Supplementary information to Figure 7.4 showing the threshold for the maximum F1 score and inference time as well as normalized confusion matrix values for each model.

| Model | UTime | CNN+ENCODER | CNN+AUTO | CNN+DENSE |
|---|---|---|---|---|
| (threshold, max F1) | (0.43,0.92) | (0.37,0.89) | (0.29,0.78) | (0.42,0.90) |
| $F1_{0.5}$ | 0.92 | 0.88 | 0.74 | 0.89 |
| Precision | 0.94 | 0.94 | 0.89 | 0.93 |
| Recall | 0.90 | 0.84 | 0.64 | 0.86 |
| TP | 0.0545 | 0.0508 | 0.0386 | 0.0523 |
| FP | 0.0037 | 0.0035 | 0.0050 | 0.0042 |
| FN | 0.0060 | 0.0098 | 0.0220 | 0.0083 |
| TN | 0.9357 | 0.9359 | 0.9344 | 0.9352 |
| Inference time | 10m 56s | 1m 5.13s | 47.8s | 1m 5.76s |

**Type of load detected**

As a further comparison we compare what type of EV load the different models are able to detect. The power drawn is divided into four categories according the type of EV charging in residential homes[13]:

1. Charging from standard type-c outlet: $[0, 2.2\text{kW}\rangle$
2. Slow EV chargers: $[2.2\text{kW}, 4.5\text{kW}\rangle$
3. Semi fast charger or overlapping slow charger: $[4.5\text{kW}, 7.2\text{kW}\rangle$
4. Overlapping charge events: $[7.2\text{kW}, \rightarrow\rangle$

Note that the categories are not according to maximum power from each charge event, but the charging load for each positive label. A comparison of what type of charging the different models were able to detect is shown in Table 7.9.

Results shows that UTime are able to detect the most EV charging with lower loads and general trend of the CNN+Autoencoder is that they are able to detect more of the higher loads. The model that where able to detect most EV charging in total is UTime even though it did performed the worse in detecting the higher loads.

Further comparison will be presented in the next section, where we compare the models on the unlabeled data set EIDSIVA EXPLORATION.

Table 7.9: Comparing the type of load from EV charging the models where able to detect and the percentage of positive labels for each category in the generated ACN+EIDSIVA test set.

| | Detection according to the power of EV load | | | | |
|---|---|---|---|---|---|
| Model | $[0, 2.2\text{kW}\rangle$ | $[2.2\text{kW}, 4.5\text{kW}\rangle$ | $[4.5\text{kW}, 7.2\text{kW}\rangle$ | $[7.2\text{kW}, \rightarrow\rangle$ | TOTAL (TPR) |
| UTime | 73.5% | 91.9% | 97.2% | 97.7% | 90.0% |
| CNN+ENCODER | 54.6% | 86.4% | 98.2% | 98.9% | 83.8% |
| CNN+AUTO | 13.4% | 60.7% | 97.0% | 98.9% | 63.7% |
| CNN+DENSE | 61.7% | 89.2% | 97.6% | 98.3% | 86.3% |
| Percentage of positives | 23.2% | 39.5% | 33.4% | 4.2% | 100% |

## 7.6 Comparing unsupervised predictions

In this section we aim to compare the best performing models on the unlabelled EIDSIVA EXPLORATION set. First by comparing detection within each data group, and second by comparing prediction profiles (detection for each hour of the week).

### 7.6.1 Detection within each data group

Table 7.10 shows the percentage of customers with any EV detection in EIDSIVA EXPLORATION dataset.

Table 7.10 shows all models are able to detect the most customers with EV charging among the customers with registered EV. Generally, there are more customers with detection after the time of registering an EV. For the data groups without EV registration, we can see that there are assumed false positives present, where UTime differentiates the data groups with and without EV registration the most.

Among the customers with EV detection, the distributions of relative hours of detection are shown in Figure 7.5 as box plots. The distribution shows fewer hours detected for the data groups where we assume no EV charging. The CNN+AUTO detects the most while UTime detects the least charging within all groups. This is consistent with the results in Table 7.10 that shows that the better performing models detect less charging for customers with and without registered EV.

Table 7.10: Percentage of customers with any detected EV charging according to the data group in the EIDSIVA EXPLORATION data set.

| Model | reg EV | Before reg EV | Test | Validation | Total |
|---|---|---|---|---|---|
| Utime | 66.1% | 23.2% | 14.6% | 13.1% | 39.7% |
| CNN+ENCODER | 72.8% | 48.8% | 37.1% | 36.5% | 62.7% |
| CNN+AUTO | 96.1% | 81.6% | 75.0% | 73.0% | 85.6% |
| CNN+DENSE | 88.2% | 29.6% | 19.9% | 18.8% | 45.9% |
| Number of loads | 2487 | 1038 | 946 | 924 | 5395 |

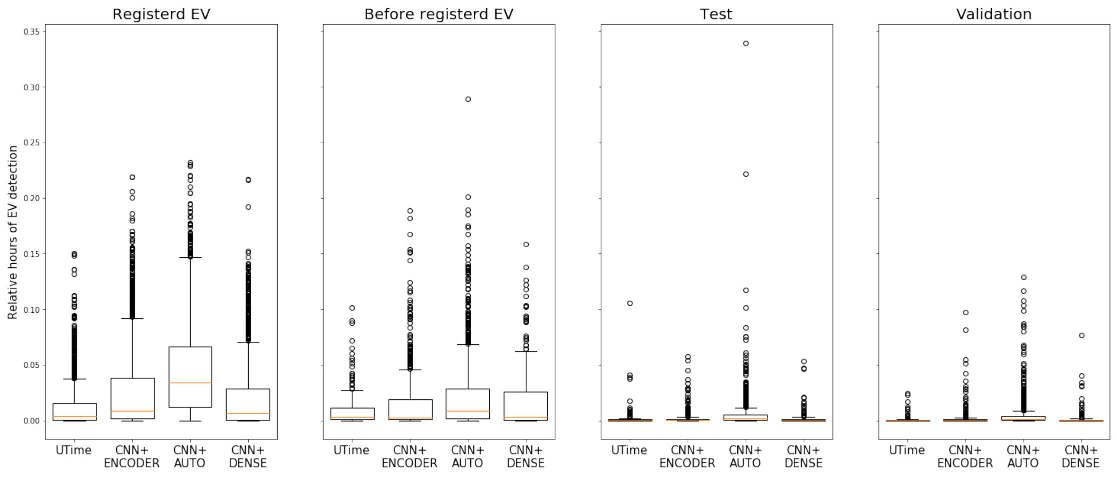


Figure 7.5: Boxplot and whiskey graph for relative hours detect when a customer with zero prediction is excluded. The yellow line indicates the median, and the box and whiskers show the quartiles.

### 7.6.2 Comparing Prediction profiles: Predictions at each hour of the week

As a final comparison, we visualize the distribution of hours detected at each hour of the week among the EV owners; this is visualized as box plots in Figure 7.7 and means of all observations in Figure 7.6. There is a clear trend of an increasing amount of predictions in the afternoon. As the load is known to be higher for these periods independently of EV charging, we cannot conclude that the models detect EV charging events or just general higher load (confound variable). There are also fewer predictions during the weekend, which is interesting as average load in the weekend is similar to average load during the working days.

The trend of more detection during high demand periods is is especially a

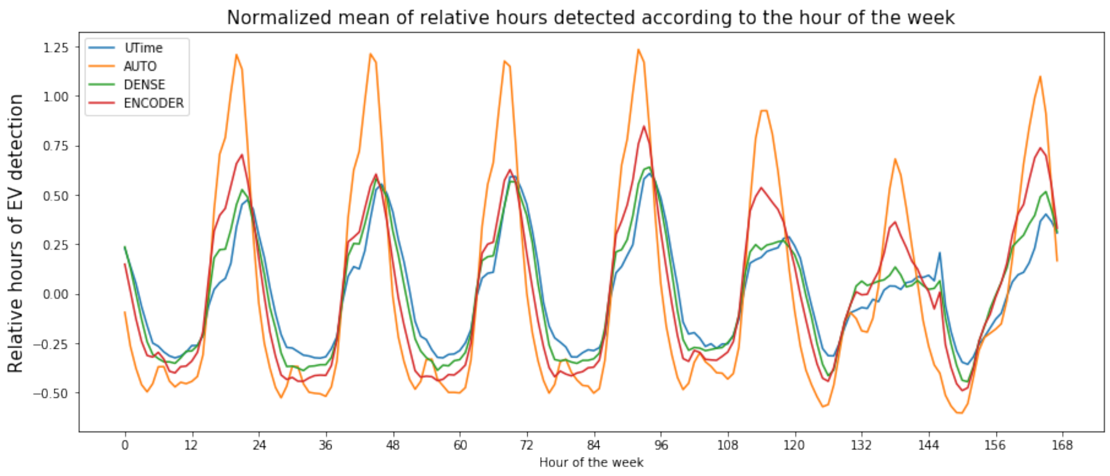


Figure 7.6: Comparing mean of the prediction profiles for each model.

concern for the CNN+AUTO and CNN+ENCODER that shows more detection on Friday and Saturday than the better performing models. In addition, the overall poorest performing model, CNN+AUTO, has an evident spike in the beginning of the days which the other models do not have. This is also a period the day when the general load is higher. Figure 7.7 also shows that the better performing UTime and CNN+DENSE have slightly more predictions during the night when the relative hours detected are normalized.

### 7.6.3 Comparison summary

The results shows that the proposed UTime outperforms the other models on the same generated labeled test set. While Utime is able to detect more charge events with lower loads, it did the worst when detecting overlapping charge events (see Table 7.9), however these events are very few. The biggest drawback with the current implementation of UTime is computation time, which are in the ball park of 10 times slower than to the other models with the current implementation.

Further we investigated the predictions on an unlabeled data set from Eidsiva. Result showed all model predicted more charge events for customers with registered EV. Comparing the hours of detection between EV and non EV owners we saw that EV owners had a distribution with longer tail towards more hours of detection.

Even though UTime had the least number of charge events detected, the other models showed more EV detection for customers we did not expect EV charging. This gave concerns whether the models are classifying higher loads as charge events (see Figure 7.6). This is especially a concern for the poorest performing models.

In the last section of this chapter, we aim to use modified UTime for EV detection, to explore whether we are able discover the time an customer register an EV.

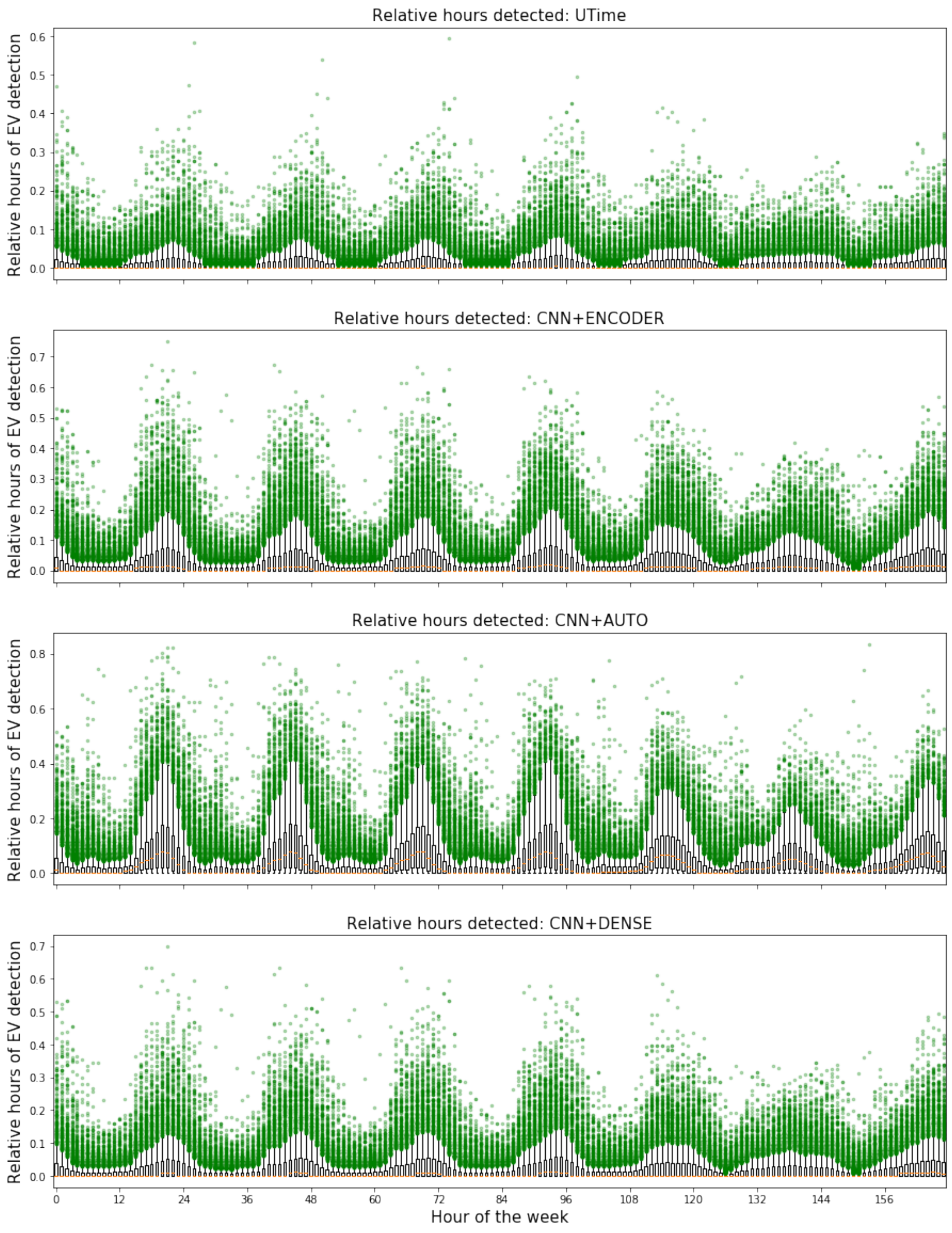


Figure 7.7: Box plots of relative hours detected per the hour of the week (starting at Monday) from customers with registered EV that has EV charge detected.

## 7.7 Exploring time of EV registration using the best performing model

When comparing models, we saw that all models are able to detect more EV charging after the time of registration. To further investigate this and try to give us some confidence in that we are actually able to detect EV charging. We will use the best performing model, which were modified UTime for EV detection, to investigate the time of registration.

There are some uncertainties about the time of registration since the available data is the month the Norwegian Public Roads Administration registered the EV to a customer, which is not necessarily the precise time the vehicle was available to the customer. Therefore we might expect some variations, but in general, should expect the charge events to occur after the time of registration. For visualization we have chosen a heatmap representation of both the sum and cumulative sum of weekly predictions. The customers are grouped according to the month of registration, and only customers with predicted EV charging is shown. Figure 7.8 shows such heat map for the months of July to December 2018. From this figure, we can see that many of the costumes have a large increase in EV detection around the month of registration, especially when looking at the cumulative sum that has an upper threshold of 10 hours of detection. However, there is also some inconsistencies, especially in December, were most customers have detected EV charging much earlier. This raises the question of whether we are detecting false positives or if there are some anomalies of car registration in December.

As a further comparison, the relative number of consumers with more, less, or zero charge detection is summarized in Table 7.11. Table 7.11 shows that for all months (in 2018), most EV owners have either zero predictions or more EV predictions after the time of registration, with a very few numbers of EV owners with less EV detection after the time of registration.

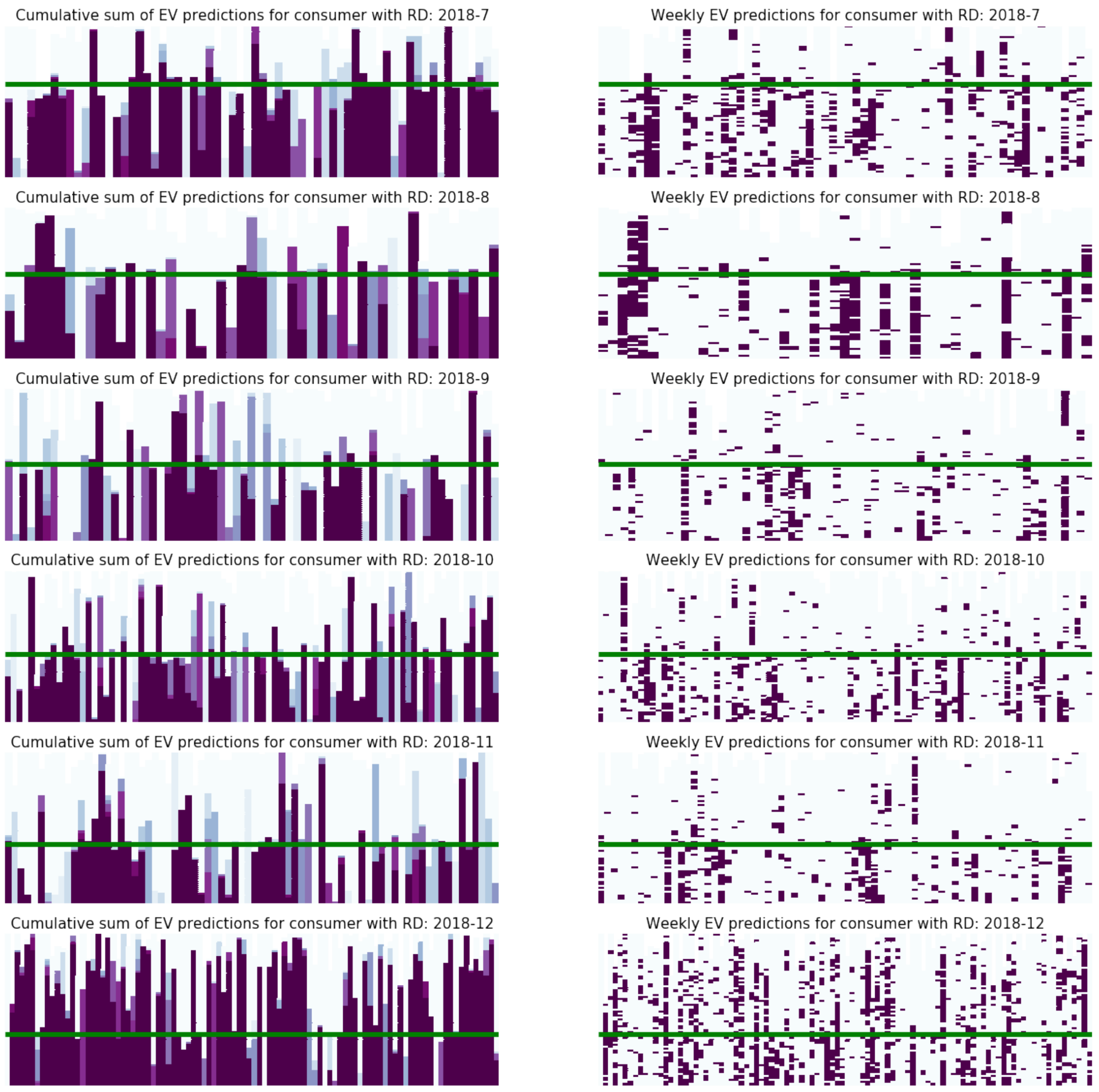


Figure 7.8: Heatmaps of charge detection where customers (x-axis) is grouped after registration date (RD) marked with the green line. The RD ranges from June to December in the year 2018 . Along the y-axis is the weeks after 2018 (starting from the top to the bottom). The heat map range is modified and removed for better visualization, the deepest purple indicates values above 10 for both cumulative sum and weekly predictions.

Table 7.11: Comparing the detection before and after the time of registering an EV. Model used for detection is UTime, and dataset is EIDSIVA EXPLORATION EV.

| | Relative number of customers with | | | Relative hours detected over all customers | |
|---|---|---|---|---|---|
| Time of registration (month in 2018) | more EV detection after | less EV detection after | zero after and before | Before | After |
| 2 | 0.635 | 0.019 | 0.346 | 0.0005 | 0.0096 |
| 3 | 0.696 | 0.018 | 0.286 | 0.0002 | 0.0056 |
| 4 | 0.734 | 0.009 | 0.257 | 0.0003 | 0.0073 |
| 5 | 0.657 | 0.038 | 0.305 | 0.0009 | 0.0105 |
| 6 | 0.573 | 0.012 | 0.415 | 0.0005 | 0.0077 |
| 7 | 0.536 | 0.045 | 0.418 | 0.0009 | 0.0086 |
| 8 | 0.611 | 0.069 | 0.319 | 0.0017 | 0.0108 |
| 9 | 0.585 | 0.106 | 0.309 | 0.0008 | 0.0057 |
| 10 | 0.546 | 0.050 | 0.397 | 0.0009 | 0.0089 |
| 11 | 0.585 | 0.113 | 0.302 | 0.0007 | 0.0064 |
| 12 | 0.610 | 0.110 | 0.279 | 0.0044 | 0.0144 |

## 7.8 Event Detection Summary

In this chapter, we have shown that there it is possible to detect EV charging from hourly smart meter data by using CNN. Since the initially proposed models showed promise, we experimented with different architectures on hourly smart meter data.

Further, we conducted experiments on how the input length, complexity, and loss functions affected the performance of the different models. When comparing the tuned models, the results showed that the newly proposed UTime for EV detection outperforms the previously proposed CNN architectures on the generated test set. When comparing unlabeled predictions, all models showed an difference in detection between EV and none EV owners, where UTime did the best job at separating these two groups. We also observed there was little overfitting occurring, showing promise regarding generalization between the generated training and validation sets.

In addition compare the models, we explored whether UTime was able to detect the time of EV registration. Results showed that UTime was able to detect more charging after than before the time of registration, among most of the customers.

Comparing our results from previous research, it is evident that our results are far better than whats previously reported (Table 2.2). This applies to all models. This raises concerns about the method the dataset is generated. However, direct comparison is difficult since the previous results are from the Pecan Street dataset, and the type of charging present in Pecan Street is unknown to the author.

If Pecan Street contains mainly charging from low-level chargers, we would expect our models also to perform worse since we have shown that our models have a more difficult time detecting charge events from low-level charging (see Table 7.9).

Another reason for our better results, might be due to when resampling the charge events with less than 1kW of power is set to zero, significantly reducing the number of positive labels that is difficult to detect, this is done to ensure our model is able to learn relevant charging patterns than fitting itself to noise.

To summarise: The proposed training method, by generating a labeled dataset, and especially the newly proposed UTime for EV detection shows promise, in the task of EV event detection. However, more verification is needed to know the effectiveness of our models, and training method. Ideally with a properly labeled dataset.

As a last attempt to verify our methods; We aim to compare our results from EV load profiling, with our prediction from modified UTime in the next chapter.

# Chapter 8

# Comparing event detection with load profiling

The motivation for trying to solve two problems of EV detection is due to the lack of ground truths in our dataset.

In this chapter, we further aim to compare the two proposed methods for further validation of whether we were able to detect EV charging. This chapter is divided into two main parts, where the following is presented:

- First, we compare the number of customers with any EV event detection in each cluster from EV load profiling, using the best performing model for EV detection.
- Second, we perform GMM of extracted prediction profiles (mean of prediction at each hour of the week) and compare its features with normalized clustering results in Chapter 6.

The predictions reported in this chapter is from UTime. Since UTime showed overall best performance in the previous chapter.

## 8.1 Comparing EV event detection within the unsupervised clusters

The percentage of customers with any detected EV charging in each cluster is shown in Table 8.1, a further detailed bar graph with EV detection is shown in Figure 8.1.

When comparing the detection within the clusters from Chapter 6.1, we can see a trend of more EV detection within clusters with a high concentration of EV owners. Especially the non-normalized mean feature coincides with EV detection. For the normalized clusters, there is a less noticeable difference in detection between the clusters, but we can see there is somewhat more EV detection in the clusters with a high concentration of EV owners. The only exception being skewness features, that didn't show a noticeable difference.

Table 8.1: Percentage of EV detection within each cluster from Chapter 6. The cluster with a high concentration of EV owners is marked with bold text.

| | Within cluster EV detection (%) | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Mean | | | Normalized Mean | | | Normalized Kurtosis | | | Normalized Skewness | | |
| cluster | EV | No EV | Train | EV | No EV | Train | EV | No EV | Train | EV | No EV | Train |
| 1 | **80.3** | **48.2** | **22.8** | 47.2 | 10.5 | 5.0 | **71.2** | **20.6** | **9.0** | **70.3** | **14.6** | **6.7** |
| 2 | 61.4 | 17.8 | 9.0 | 63.4 | 14.2 | 7.8 | 55.4 | 13.3 | 7.6 | 71.8 | 14.4 | 7.2 |
| 3 | 25.8 | 8.4 | 4.0 | **73.4** | **24.6** | **6.8** | 68.7 | 13.4 | 5.6 | 52.1 | 13.35 | 6.8 |
| 4 | | | | 69.1 | 16.9 | 9.5 | | | | | | |

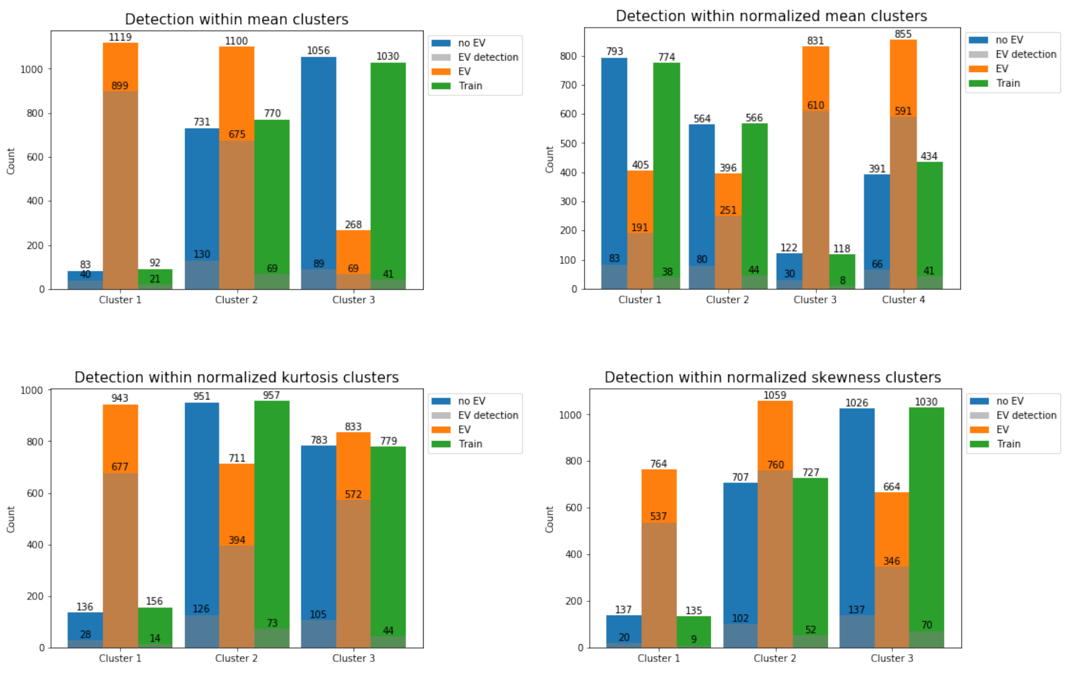


Figure 8.1: EV detection within clusters from Chapter 6.

## 8.2 Gaussian mixture modeling of weekly prediction profiles

In this section, we report the result when applying GMM for weekly-hourly prediction profiles (relative hours detected for each hour of the week). The prediction profiles were extracted by using the best performing model for EV detection, which is modified UTime for EV detection.

The clustering shown in Table 8.2, and Figure 8.2 shows within-cluster predictions of both normalized and non-normalized prediction profiles, for the whole week. Showing that by clustering raw prediction profiles, we are able to filter out some customers with less EV detection.

The next step is to compare the features from each of these clusters, with the clustering results in Chapter 6.

Table 8.2: Final Gaussian mixture results with hard cluster assignment both prediction profiles and normalized prediction profiles.

| | Prediction profiles (%) | | | Normalized prediction profiles (%) | | |
|---|---|---|---|---|---|---|
| cluster | EV | No EV | Train | EV | No EV | Train |
| 1 | 53.7 | 97.1 | 99.3 | **64.7** | **12.6** | **6.6** |
| 2 | **46.3** | **2.9** | **0.7** | 35.3 | 87.4 | 93.4 |

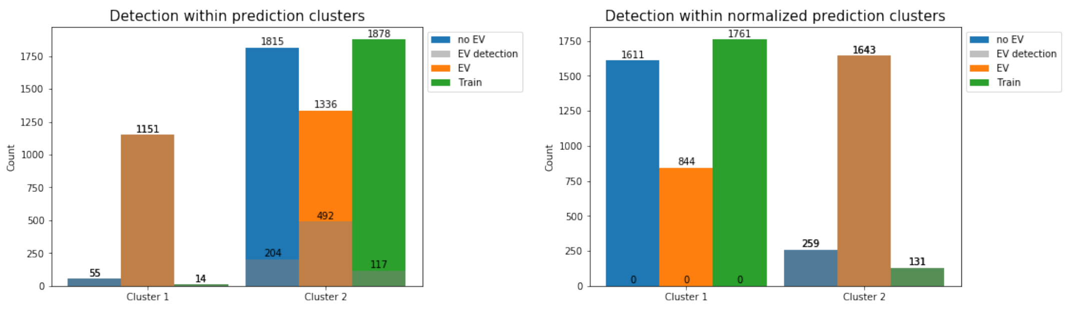


Figure 8.2: Data group cluster assignment within each of the prediction clusters.

## 8.3 Comparing cluster means with customers with predictions

The idea in this section, is to see if there are similarities between the features of clusters with a high concentration of EV owners, and those customers with predictions. This is achieved by comparing the mean of the features within each cluster from Chapter 6, with the feature means from the prediction profile clustering. Comparison is shown in Figure 8.3 to 8.5. In the figure, there is also calculated Normalized cross correlation (NCC) between each mean vector, giving a similarity score between $-1$ and 1 for numeric comparison.

Comparisons shows that the mean features show similarities where customers with EV detection has generally a higher load and a shift of higher power consumption during the night. When comparing kurtosis and skewness we can see the clusters with a high a concentration of EV owners correlates the most with each other. This results shows promise that there are similarities between EV event detection and clustering results.

## 8.4 Summary and discussion

In this chapter we have compared our clustering results from EV load profiling, with the result from EV event detection. Overall the result are promising showing that both methods coincides. However, the comparison is not conclusive and further verification is needed to state conclusively to which degree we are able to detect EV charging. This will be further be discussed in the last chapter of this thesis.

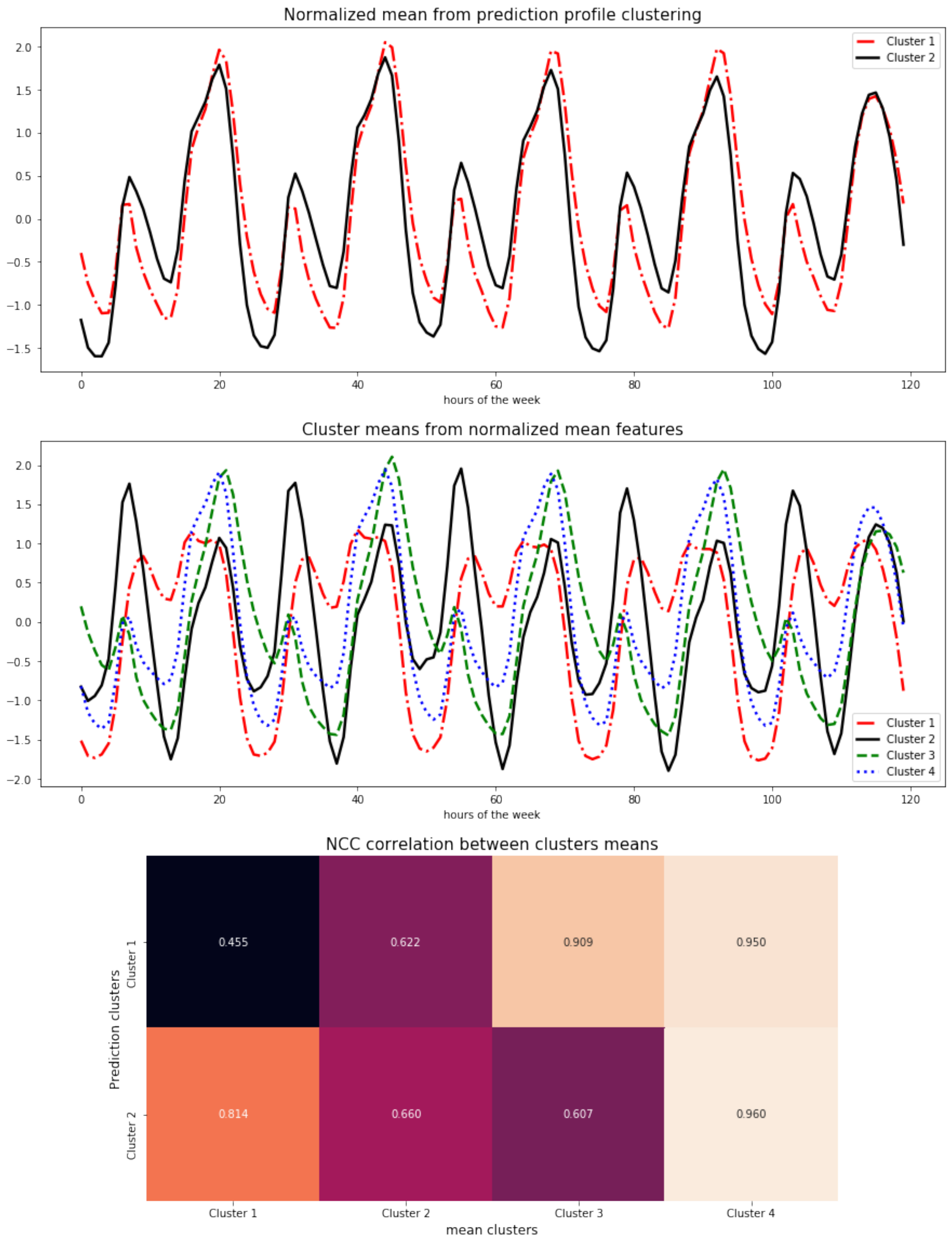


Figure 8.3: Comparing cluster means from normalized mean features with normalized mean features from the clusters from prediction profiles. The cluster with a higher concentration of EV owners is cluster 1 for the prediction profile clustering and cluster 3 for the mean profile clusters. We can see that the both cluster means, have a shift of higher power consumption during the night, and a bigger peek consumption than the other clusters.

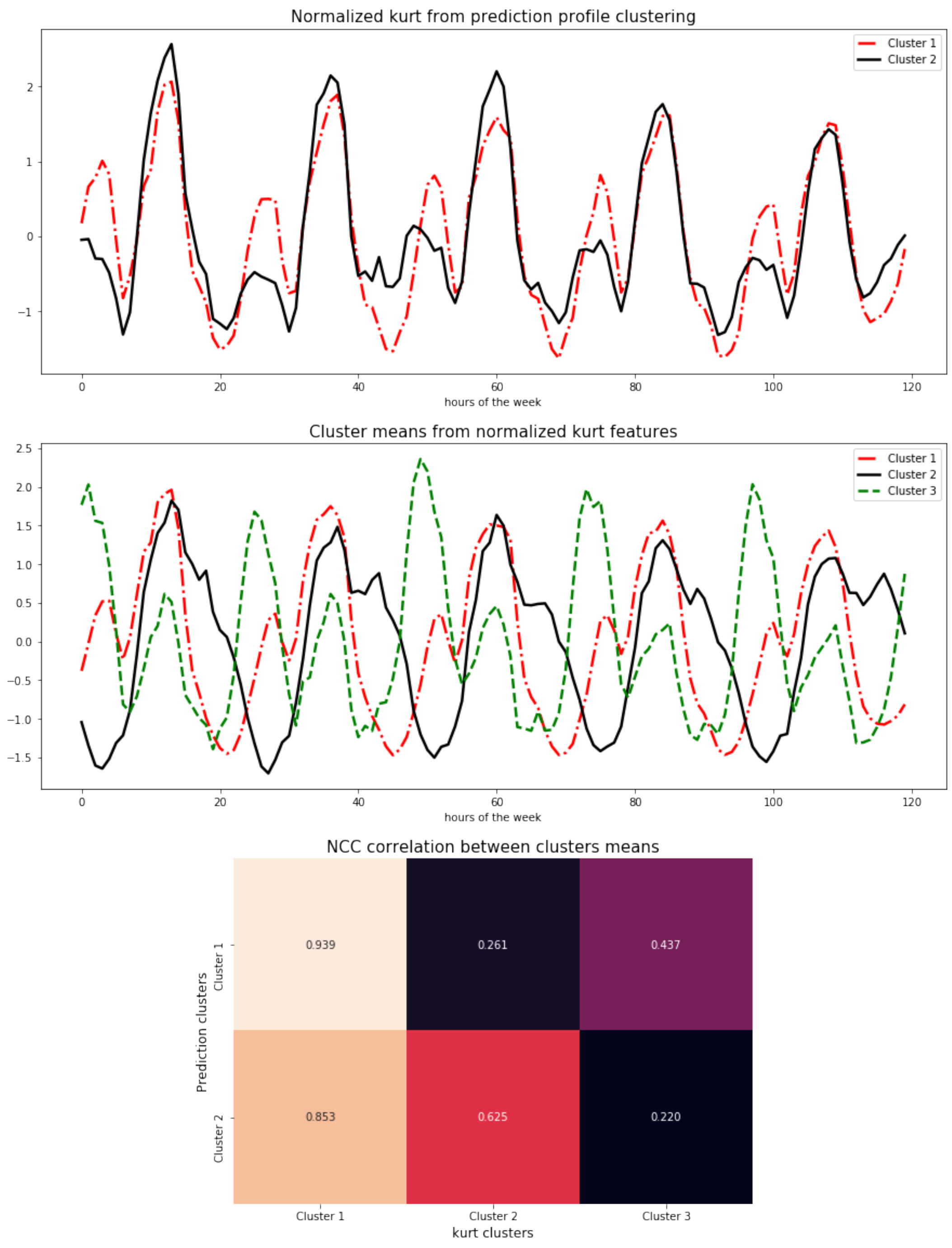


Figure 8.4: Comparing cluster means from kurtosis features with normalized kurtosis features extracted from the clusters of prediction profiles. The cluster with a higher concentration of EV owners is cluster 1 for the prediction profile clustering and cluster 1 for the kurtosis profile clusters. These cluster means correlates the most with each other, and both have a similar pattern, with a spike in the middle of the day.

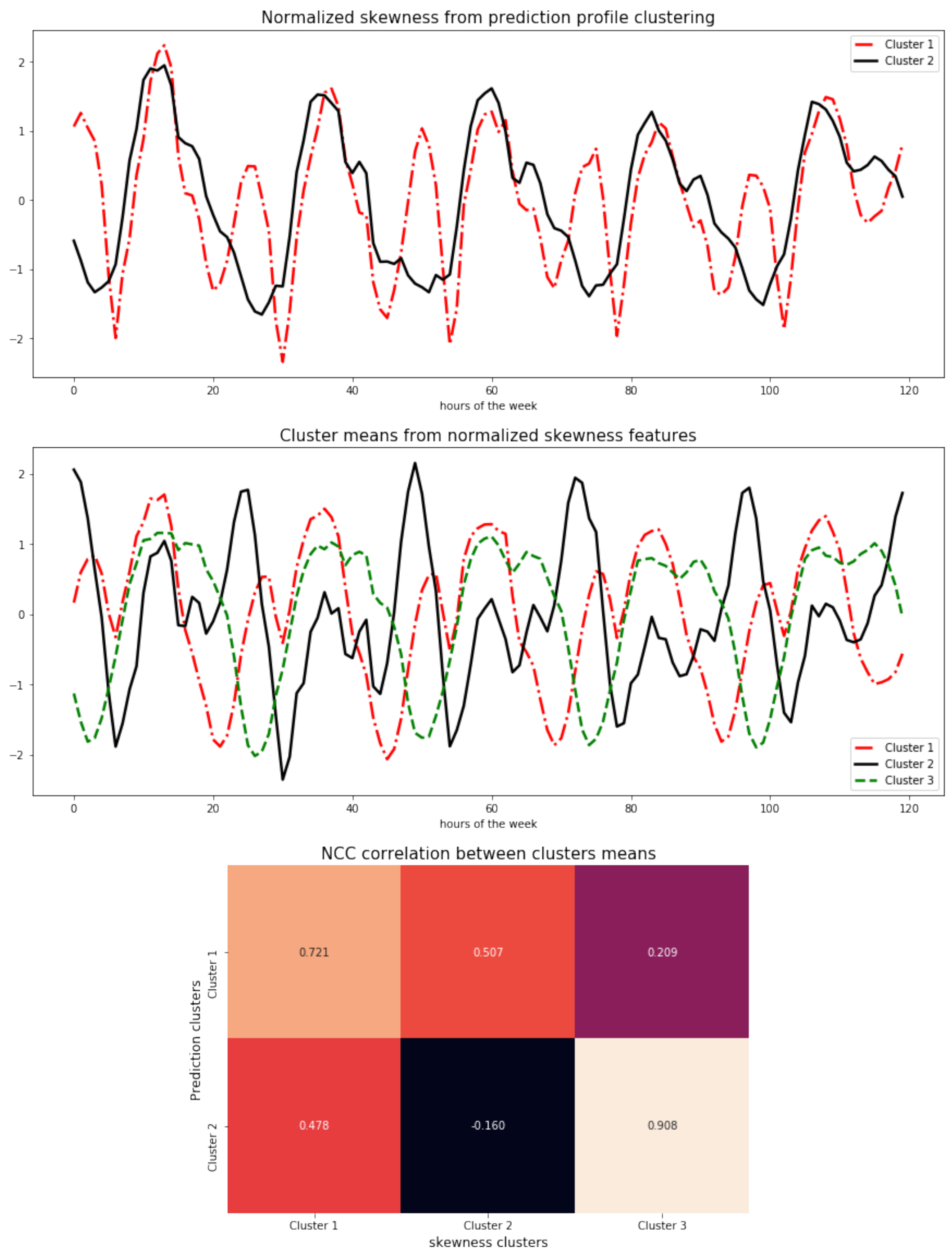


Figure 8.5: Comparing cluster means from skewness features with normalized skewness features extracted from the clusters of prediction profiles. The cluster with a higher concentration of EV owners is cluster 1 for the prediction profile and 1 for the skewness profile. The computed NCC shows these cluster correlates the most with each other. From the means we can see both have a similar spike at the beginning of the day.

# Chapter 9

# Conclusion and further work

In this thesis, we have explored two problems of EV detection. The problem of EV load profiling, and EV event detection.

The main focus have been the problem of EV event detection. But due to the lack of ground truth of charge event in our dataset, we saw the need to explore the problem of EV load profiling for further verification of our methods.

For the problem of **EV load profiling**, we proposed a new method of detrending our data before feature extraction of mean, kurtosis, and skewness profiles. The aim was to capture uniqueness to some EV owners, by using Gaussian mixture modelling.

The results showed that all features were able to capture clusters with a high concentration of EV owners. The data set used was EIDSIVA CLUSTERING, a relatively balanced dataset with 57% registered EV owners.

For the problem of **EV event detection**, we proposed the fully convolutional UTime for EV detection and compared its performance with previously proposed CNN architectures. Results showed that UTime outperforms the other architectures on the generated ACN+EIDSIVA Test set.

Further, we compared the models on the unlabeled dataset EIDSIVA EXPLORE. UTime showed the least detection within each data group and the best separation between EV owners, and none EV owners. For the poorer performing models, we saw they followed the trend of more detection in high demand periods. While the better performing models had a shift in relatively more detection during the afternoon and midnight. This shift is similar to the clustering of the normalized mean features that showed a general trend of higher energy consumption, shifted towards midnight.

Since UTime overall performed the best, UTime where further explored in the task of detecting the time of EV registration of EV owners. The result showed that UTime was able to detect more EV charging after than before the time of EV registration.

As a final comparison, we compared our final clustering results with EV detection using UTime. Except for the skewness feature, we saw more EV detection within clusters with a high concentration of EV owners. Furthermore, we observed that the mean feature of EV owners with many predictions correlated the most with the clusters with a high concentration of EV owners. This further strengthen our belief in that the proposed training method, and modified UTime for EV detection can detect EV charging from a real-life dataset. However, we can not conclusively state the effectiveness of our training method, before the method is validated on a properly labeled dataset. Which brings us to the last section of this thesis; Further work.

**Further work**

There are several things that we wish to address as further work. In this section we list and discuss some of them:

- **More validation:** First and foremost, a big challenge for this project was the lack of a labeled data set. To verify whether our promising results and training method are sound, we suggest further evaluation using a properly labeled dataset where ground truth of EV charge event is present.

- **Improvements in data generation:** From the unsupervised EV detection results, we saw a clear trend of when EV detection is occuring (Chapter 7.6.2). This trend was not taken into account when the data set was generated. We would suggest using this knowledge to generate a better-labeled dataset that is more similar to real-life EV charging patters. This might also improve the models performance, since they might, learn these trending patterns. We also saw that the balance of the generated dataset effected performance. We choose to generate a very unbalanced dataset, with a large variation of charge events for the few segments that had charge event present. The balance of the dataset generated could also be explored further, to improve the predictions on real life charge events.

- **Better data sources:** The charge events added where from commercial charging station in the US, these events might not be similar to charge events that's present in Norwegian residential homes. To have a data source more similar to Norwegian charge sessions could, improve the predictions in Norway. Further our models where only trained to detect EV charging from slow to fast EV charges. This excludes detection of charging from standard type-c outlets, which is common way to charge an EV at home in Norway. In future work, these type of charges could also be added during training.
- **Compare our results with RNN architectures:** As discussed, RNN is the go-to framework deep-learning of sequential data. In this thesis, we have utilized CNN architectures and showed these works well. However, as for further work, we propose, comparing our methods with RNNs. Especially if the method of data generation is improved, such that there is a trend when EV charging is occurring.
- **More research into EV load profiling, and evaluation of the current method:** The clustering presented should be regarded as an initial attempt that reached the requirements for this thesis. The proposed feature spaces showed promising results of identifying EV owners. We recommend more evaluation and exploration of clustering algorithms in future work. This is further discussed in Chapter 6.4.

# Chapter 10

# Bibliography

[1] S. Bai, J. Z. Kolter, and V. Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling," 2018.

[2] S. Wang, L. Du, J. Ye, and D. Zhao, "Robust identification of ev charging profiles," in *2018 IEEE Transportation Electrification Conference and Expo (ITEC)*, pp. 1–6, June 2018.

[3] B. I. Fesche, "Signal discovery in the smart grid-finding: Electric vehicle charging patterns in power consumption data," Master's thesis, University of Oslo, Norway, 2018.

[4] M. Perslev, M. Hejselbak Jensen, S. Darkner, P. Jørgen Jennum, and C. Igel, "U-Time: A Fully Convolutional Network for Time Series Segmentation Applied to Sleep Staging," *arXiv e-prints*, p. arXiv:1910.11162, Oct 2019.

[5] "Hvordan lader du elbilen hjemme?." `https://infogram.com/elbilisten-2018-hvordan-lader-du-elbilen-hjemme-1hxj48817rjq4vg`.

[6] I. K. C. I. N. P. M. Hoffmann Volker, Fesche Bjørn Ingeberg, "Automated detection of electric vehicles in hourly smart meter data," *CIRED 2019 Conference*, June 2019.

[7] A. Verma, A. Asadi, K. Yang, and S. Tyagi, "A data-driven approach to identify households with plug-in electrical vehicles (pevs)," *Applied Energy*, vol. 160, pp. 71 – 79, 2015.

[8] A. Verma, A. Asadi, K. Yang, A. Maitra, and H. Asgeirsson, "Analyzing household charging patterns of plug-in electric vehicles (pevs): A data mining

approach," *Computers and Industrial Engineering*, vol. 128, pp. 964 – 973, 2019.

[9] C. for Sustainable Energy, "What uses watt? how much electricity am i using?," 2019.

[10] International Energy Agency, *Global EV Outlook 2019: Scaling-up the transition to electric mobility.* OECD, June 2019.

[11] International Energy Agency, *Nordic EV Outlook 2018.* OECD, 2018.

[12] "Statistikk elbil." `https://elbil.no/elbilstatistikk/`.

[13] E. E. o. D. S. Christer Heen Skotland, "What does electrical cars mean for the electrical grid? (translated from norwegian)," 2016.

[14] P. I. Sæle Hanne, "Electric vehicles in norway and the potential for demand response," *53rd International Universities Power Engineering Conference (UPEC)*, 2018.

[15] T. H. Y. Wang, Q. Chen and C. Kang, "Review of smart meter data analytics: Applications, methodologies, and challenges," *IEEE Transactions on Smart Grid*, vol. 10, no. 3, May 2019.

[16] "Forskrift om måling, avregning, fakturering av nettjenester og elektrisk energi, nettselskapets nøytralitet mv.." `https://lovdata.no/dokument/SF/forskrift/1999-03-11-301`.

[17] "Smarte strømmålere (ams).." `https://www.nve.no/stromkunde/smarte-strommalere-ams/`.

[18] "Automatisk strømmåling." `https://www.datatilsynet.no/personvern-pa-ulike-omrader/overvaking-og-sporing/strommaling/`.

[19] E. Aladesanmi and K. Folly, "Overview of non-intrusive load monitoring and identification techniques," *IFAC-PapersOnLine*, vol. 48, no. 30, pp. 415 – 420, 2015. 9th IFAC Symposium on Control of Power and Energy Systems CPES 2015.

[20] M. Zhuang, M. Shahidehpour, and Z. Li, "An overview of non-intrusive load monitoring: Approaches, business applications, and challenges," 11 2018.

[21] K. S. Barsim and B. Yang, "Toward a semi-supervised non-intrusive load monitoring system for event-based energy disaggregation," 12 2015.

[22] "Pecan street." `https://www.pecanstreet.org/`.

[23] P. Zhang, C. Zhou, B. G. Stewart, D. M. Hepburn, W. Zhou, and J. Yu, "An improved non-intrusive load monitoring method for recognition of electric vehicle battery charging load," *Energy Procedia*, vol. 12, pp. 104 – 112, 2011. The Proceedings of International Conference on Smart Grid and Clean Energy Technologies (ICSGCE 2011.

[24] A. Shaw and B. P. Nayak, "Electric vehicle charging load filtering by power signature analysis," in *2017 International Conference on Data Management, Analytics and Innovation (ICDMAI)*, pp. 71–75, Feb 2017.

[25] Z. Zhang, J. H. Son, Y. Li, M. Trayer, Z. Pi, D. Y. Hwang, and J. K. Moon, "Training-free non-intrusive load monitoring of electric vehicle charging with low sampling rate," in *IECON 2014 - 40th Annual Conference of the IEEE Industrial Electronics Society*, pp. 5419–5425, Oct 2014.

[26] A. A. Munshi and Y. A. I. Mohamed, "Unsupervised nonintrusive extraction of electrical vehicle charging load patterns," *IEEE Transactions on Industrial Informatics*, vol. 15, pp. 266–279, Jan 2019.

[27] A. A. Munshi and Y. A. I. Mohamed, "Extracting and defining flexibility of residential electrical vehicle charging loads," *IEEE Transactions on Industrial Informatics*, vol. 14, pp. 448–461, Feb 2018.

[28] P. Comon, "Independent component analysis, a new concept?," *Signal Processing*, vol. 36, no. 3, pp. 287 – 314, 1994. Higher Order Statistics.

[29] S. Haben, C. Singleton, and P. Grindrod, "Analysis and clustering of residential customers energy behavioral demand using smart meter data," *IEEE Transactions on Smart Grid*, vol. 7, no. 1, pp. 136–144, 2016.

[30] P. Laurinec and M. Lucka, "Comparison of representations of time series for clustering smart meter data," 10 2016.

[31] S. Aghabozorgi, A. S. Shirkhorshidi], and T. Y. Wah], "Time-series clustering – a decade review," *Information Systems*, vol. 53, pp. 16 – 38, 2015.

[32] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning: data mining, inference and prediction.* Springer, 2 ed., 2009.

[33] J. A. H. Geof .H Givens, *Computational statistics.* John Wiley and Sons, 2 ed., 2013.

[34] T. Saito and M. Rehmsmeier, "The precision-recall plot is more informative than the roc plot when evaluating binary classifiers on imbalanced datasets," *PLOS ONE*, vol. 10, pp. 1–21, 03 2015.

[35] C. H. Sudre, W. Li, T. Vercauteren, S. Ourselin, and M. J. Cardoso, "Generalised dice overlap as a deep learning loss function for highly unbalanced segmentations," *CoRR*, vol. abs/1707.03237, 2017.

[36] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *in COMPSTAT*, 2010.

[37] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[38] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning Representations by Back-Propagating Errors*, p. 696–699. Cambridge, MA, USA: MIT Press, 1988.

[39] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014.

[40] S. Kiranyaz, O. Avci, O. Abdeljaber, T. Ince, M. Gabbouj, and D. J. Inman, "1d convolutional neural networks and applications: A survey," 2019.

[41] O.-J. Skrede, "Lecture notes: Dense neural network classifiers in5400 / in9400 — machine learning for image analysis," January 2019.

[42] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," *Journal of Machine Learning Research*, vol. 15, 01 2010.

[43] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological Cybernetics*, vol. 36, pp. 193–202, apr 1980.

[44] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in Neural Information Processing Systems 25* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), pp. 1097–1105, Curran Associates, Inc., 2012.

[45] H. Qi, “Derivation of backpropagation in convolutional neural network ( cnn ),” 2016.

[46] Jefkine, “Backpropagation in convolutional neural networks,” January 2016.

[47] J. L. Elman, “Finding structure in time,” *Cognitive Science*, vol. 14, no. 2, pp. 179 – 211, 1990.

[48] N. Kalchbrenner, L. Espeholt, K. Simonyan, A. van den Oord, A. Graves, and K. Kavukcuoglu, “Neural machine translation in linear time,” 2016.

[49] Y. Dauphin, A. Fan, M. Auli, and D. Grangier, “Language modeling with gated convolutional networks,” 12 2016.

[50] Q. Chen and R. Wu, “Cnn is all you need,” 2017.

[51] S. Hochreiter, “The vanishing gradient problem during learning recurrent neural nets and problem solutions,” *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 6, pp. 107–116, 04 1998.

[52] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber, “Gradient flow in recurrent nets: the difficulty of learning long-term dependencies,” in *A Field Guide to Dynamical Recurrent Neural Networks* (S. C. Kremer and J. F. Kolen, eds.), IEEE Press, 2001.

[53] J. Kelly and W. Knottenbelt, “Neural nilm,” *Proceedings of the 2nd ACM International Conference on Embedded Systems for Energy-Efficient Built Environments - BuildSys ’15*, 2015.

[54] X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks,” in *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS’10). Society for Artificial Intelligence and Statistics*, 2010.

[55] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” 2015.

[56] P. Flandrin, P. Gonçalvès, and G. Rilling, "Detrending and denoising with empirical mode decompositions," in *2004 12th European Signal Processing Conference*, pp. 1581–1584, 2004.

[57] W. S. Cleveland, "Robust locally weighted regression and smoothing scatterplots," 1979.

[58] R. Pearson, Y. Neuvo, J. Astola, and M. Gabbouj, "Generalized hampel filters," *EURASIP Journal on Advances in Signal Processing*, vol. 2016, 12 2016.

[59] S. Aghabozorgi, A. S. Shirkhorshidi, and T. Y. Wah, "Time-series clustering – a decade review," *Information Systems*, vol. 53, pp. 16 – 38, 2015.

[60] T. Oliphant, *Guide to NumPy*. 01 2006.

[61] W. McKinney *et al.*, "Data structures for statistical computing in python," in *Proceedings of the 9th Python in Science Conference*, vol. 445, pp. 51–56, Austin, TX, 2010.

[62] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[63] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," 2015.

[64] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," 2019.

[65] Z. J. Lee, T. Li, and S. H. Low, "ACN-Data: Analysis and Applications of an Open EV Charging Dataset," in *Proceedings of the Tenth International Conference on Future Energy Systems*, e-Energy '19, June 2019.

[66] J. Kelly and W. Knottenbelt, “The UK-DALE dataset, domestic appliance-level electricity demand and whole-house demand from five UK homes,” vol. 2, no. 150007, 2015.

[67] T. Fawcett, “An introduction to roc analysis,” *Pattern Recognition Letters*, vol. 27, no. 8, pp. 861 – 874, 2006. ROC Analysis in Pattern Recognition.