

Technische Hochschule Ingolstadt
Faculty of Mechanical Engineering
Renewable Energy Systems

Master's thesis

**Multi-Feature based Development of a Power-Disaggregation Algorithm
for Dairy Farms**

Anuj Sinha

Matrikelnummer: 00086760

Issued on : **30.06.2020**

Submitted on : **22.01.2021**

First examiner : **Prof. Dr.-Ing. Wilfried Zörner**


Second examiner : **Prof. Dr. -Ing. Tobias Schrag**

Declaration

I hereby declare that this thesis is my own work, that I have not presented it elsewhere for examination purposes and that I have not used any sources or aids other than those stated. I have marked verbatim and indirect quotations as such.

Ingolstadt, 22.01.2021

Place, Date



Anuj Sinha

00086760

Abstract

Smart meter technology implementation in the last decade had initiated many data collection processes, which have provided a strong foundation for the development of Artificial Intelligence (AI) based load monitoring systems. It is easier to identify the energy-saving potential with the help of advanced load monitoring systems. Since 2015, deep-learning-based Nonintrusive load monitoring (NILM) is being focused in the research community. It requires minimal hardware, which can justify its development and maintenance cost. Several AI-based models and tools are available for load monitoring, but it is challenging to identify a suitable model for the specific application. There is still a domain-specific transformation, and considerations are usually required. The residential sector has been the focus area due to the market size, but the industrial sector still has massive potential for research and development.

Thus, in the presented thesis, dairy farms in Germany are targeted for developing a power disaggregation algorithm based on deep learning, which can identify the on/off state of individual appliances in the farm from the aggregated load profile data. Mainly four appliances named milk cooling (MK), milk pump (MP), vacuum pump (VP), and cleaning automatic machine (SA) are targeted for disaggregation. NILM is a promising approach to identify individual operating times of appliances. Thus, deep neural network-based algorithms are developed, focusing mainly on one-dimensional convolution neural network (1D-CNN) and recurrent neural network (RNN).

Literature research was carried out to determine the state-of-the-art of deep-learning-based NILM and understand AI technology. Data acquisition for model development and testing was made from four dairy farms based out of Bavaria, Germany. The presented work provides a detailed discussion about data pre-processing and development of models. The result shows that deep-learning-based disaggregation algorithms outperform for this application area, and the proposed model successfully identifies the states of individual appliances. The presented work provides a foundation for modifying the proposed algorithm or developing a new algorithm for real-time power disaggregation.

Acknowledgement

The work presented in this thesis was carried out at the Institute of new Energy Systems (InES), THI, Ingolstadt, and Faculty of Mechanical Engineering at Technische Hochschule Ingolstadt. I would like to thank my supervisor, Mr. Abdessamad Saidi, for guiding me in various aspects of the projects, providing ideas and alternatives to the problems I encountered along the way. His guidance helped me improve the quality of research during my thesis, which also reflects in the writing part. I would like to thank my supervisors, Prof. Dr.-Ing. Wilfried Zörner and Prof. Dr.-Ing. Tobias Schrag, for giving me the opportunity to do my thesis on this particular topic and for their continuous presence, useful comments, remarks, observations, and engagement, all through this master thesis's learning process. I would like to thank my parents and friends for their constant support and motivation, directly or indirectly, that helped me complete this thesis.

Table of contents

Declaration	I
Abstract	II
Acknowledgement	III
List of figures	VI
List of Tables	VII
Abbreviations and Symbols	VIII
1. Introduction	1
1.1 Background and Motivation.....	1
1.2 Goal of the thesis	4
1.3 Research Questions.....	4
2. Theoretical Background	5
2.1 Load Monitoring.....	5
2.1.1 <i>Event and Event-less Load Monitoring Approach</i>	5
2.1.2 <i>Intrusive and Non-intrusive Load Monitoring</i>	6
2.2 Machine Learning.....	8
2.2.1 <i>Markov Chains</i>	9
2.2.2 <i>Hidden Markov Model</i>	10
2.3 Deep Learning.....	10
2.3.1 <i>Basic Concept and Brief History</i>	10
2.3.2 <i>Deep Learning Libraries</i>	11
2.3.3 <i>Artificial Neural Network</i>	12
2.3.4 <i>Deep Neural Network Libraries</i>	13
2.3.5 <i>Back Propagation</i>	13
2.3.6 <i>Recurrent Neural Network (RNN)</i>	14
2.3.7 <i>Long Short-Term Memory (LSTM)</i>	15
2.3.8 <i>Gated Recurrent Unit (GRU)</i>	16
2.3.9 <i>Bidirectional Recurrent Neural Network</i>	17
2.3.10 <i>Convolutional Neural Network</i>	18
2.4 Deep Learning Fundamentals.....	22
2.4.1 <i>Activation Function</i>	22
2.4.2 <i>Loss Functions</i>	24
2.4.3 <i>Learning Rate</i>	25
2.4.4 <i>Optimization</i>	26
2.4.5 <i>Parameter Initialization</i>	27
2.5 Evaluation Matrix.....	27
2.6 Save and load model	28
2.7 Development Tools	29
3. Data Collection and Pre-processing	30
3.1 Data Collection Process.....	30
3.1.1 <i>Background Information of Project Partners</i>	30
3.1.2 <i>Measurement Devices</i>	30
3.1.3 <i>Overview of the Data</i>	32
3.2 Data Munging.....	34
3.2.1 <i>Handling Missing Values</i>	34
3.2.2 <i>Data Transforms</i>	36
4. Model Development	38
4.1 Model-1: 1D-CNN- BDRNN.....	38
4.2 Model-2: 1D-CNN-LSTM	40
4.3 Model-3: LSTM.....	41
4.4 Model-4: Multi-feature 1D-CNN-BDRNN	42
4.5 Model-5: Multi-feature 1D-CNN-LSTM.....	43
4.6 Model-6: Multi-feature LSTM.....	44
5. Results and Comparison	45
6. Conclusion and Outlook	56

6.1	General Conclusion.....	56
6.2	Limitation and Future Work.....	56
	References	57
	Appendix	62

List of figures

Figure 1 Significant growth in U.S. granted patents (Zeifman and Roth, 2012, p. 3)	3
Figure 2 Composition of electricity consumption in dairy farming (Neser et al., 2014, p. 14).....	3
Figure 3 An example of event-based energy disaggregation. (Copyright 1992 IEEE) (Pereira and Nunes, 2018, p. 2)	5
Figure 4 Example of event-less energy disaggregation. Copyright 1992 IEEE) (Pereira and Nunes, 2018, p. 2)	6
Figure 5 Basic Markov chain represented in graph form (Fiol, 2016, pp. 9–10).....	9
Figure 6 Example of HMM showing the states and observations (Raiker et al., 2018 - 2018, pp. 381–385).....	10
Figure 7 A simple feed-forward neural network where nodes in each layer are connected to all the nodes in the corresponding layer (Dertat, 2017)	12
Figure 8 Recurrent neural network (dprogrammer, 2019)	14
Figure 9 An unrolled recurrent neural network (oinkina et al., 2015).....	15
Figure 10 The LSTM network consists of four interacting layers: input gate, output (oinkina et al., 2015).....	16
Figure 11 Gated recurrent units (oinkina et al., 2015)	17
Figure 12 General structure of Bidirectional recurrent neural networks (oinkina et al., 2015)....	18
Figure 13 Example of a filter applied to a two-dimensional input to create a feature map (Brownlee, 2020a)	18
Figure 14 Kernel sliding over the Image (Verma, 2019)	19
Figure 15 A typical convolutional neural network.....	21
Figure 16 Kernel sliding over 1 D data (Verma, 2019).....	22
Figure 17 Commonly used activation functions: (a) Sigmoid, (b) Tanh, (c) ReLU,(d) LReLU (Feng et al., 2019, p. 3)	24
Figure 18 Learning rate (Donges, 2020)	25
Figure 19 Local minima and Global minima (Khandelwal, 2018)	26
Figure 20 Typical structure of data pre-processing.....	30
Figure 21 EMONIO P3 (Berliner Energieinstitut GmbH, 2019).....	31
Figure 22 EMONIO Installation in switchboard (Berliner Energieinstitut GmbH, 2019).....	31
Figure 23 Lackmann smart meter (Lackmann GmbH & Co. KG, 2020)	32
Figure 24 Exported files from measurement devices.....	33
Figure 25 Load profiles (Raw Data Sample).....	33
Figure 26 Sample of missing values dataset. (a) Farm1-VP; (b) Farm4-VP	34
Figure 27 Sample of date-time column shift for both the devices.....	35
Figure 28 Architecture of Model-1 (1D-CNN-bidirectional RNN)	39
Figure 29 Architecture of Model-2 (1D-CNN_LSTM)	40
Figure 30 Architecture of Model-3 (LSTM).....	41
Figure 31 Architecture of Model-4 (Multi-Feature 1D-CNN-bidirectional RNN)	42
Figure 32 Architecture of Model-5 (Multi-Feature 1D-CNN_LSTM).....	43
Figure 33 Architecture of Model-6 (Multi-Feature-LSTM)	44
Figure 34 Comparison of result for three models tested for MK on different dates and farms ...	46
Figure 35 F1-Score comparison of three model tested for MK on different dates and farms.....	47
Figure 36 Comparison of the results of three models tested for MP on different dates	48
Figure 37 F1-Score comparison of three model for MP on different dates.....	48
Figure 38 Comparison of results of the three models tested for VP of different dates and farms	49
Figure 39 F1-Score comparison of three model for VP tested on different dates and farms	50
Figure 40 Comparison of results of three model for SA tested on different dates.....	51
Figure 41 F1-Score comparison of three model for SA tested on different dates	51
Figure 42 Comparison of the results of three multi-features model for SA tested on different dates	52
Figure 43 F1-Score comparison of three multi-features model for SA tested on different dates	53
Figure 44 Comparison of the results of three multi-features model for VP tested on different dates	54
Figure 45 F1-Score comparison of three multi-features model for VP tested on different dates	54
Figure 46 Screenshot of query asked from EMONIO Team	62

List of Tables

Table 1 Confusion matrix (Bernard, 2018, p. 119)	28
Table 2 Software information	29
Table 3 Data availability from farms	34
Table 4 Sample of error in polarity and accuracy	36
Table 5 Parameters for transformation to state values	36
Table 6 Example of sample datasets	37
Table 7 Selection of datasets for training	38
Table 8 Selection of datasets for testing	38
Table 9 Training parameters of Model-1	39
Table 10 Training parameters of Model-2	41
Table 11 Training parameters of Model-3	42
Table 12 Training parameters of Model-4	42
Table 13 Training parameters of Model-5	43
Table 14 Training parameters of Model-6	44

Abbreviations and Symbols

AC	Alternative Current
AI	Artificial Intelligence
API	Application Programming Interface
ANN	Artificial Neural Network
BDRNN	Bidirectional Recurrent Neural Network
CNN	Convolutional Neural Networks
Cos(Φ)	Power Factor
CPU	Central Processing Unit
FN	False Negative
FP	False Positive
GPU	Graphical Processing Unit
GRU	Gated Recurrent Unit
HMM	Hidden Markov Model
I	Current (A)
InES	Institute of new Energy Systems
JVM	Java Virtual Machine
L	loss
LReLU	Leaky Rectified Linear Units
LSTM	Long Short-Term Memory
MIT	Massachusetts Institute of Technology
MILA	Montreal Institute for Learning Algorithms
MK	Milk Colling
ML	Machine learning
MP	Milk Pump
MR	Maschinenringe
MSE	Mean of the Squared Errors
N	Neuron
NILM	Non-Intrusive Load Monitoring
p	Probability

P	Active Power (kW)
Q	Reactive Power (kVAR)
ReLU	Rectified Linear Units
RNN	Recurrent Neural Networks
S	Apparent Power (kVA)
SA	Rising automatic machine
t	Time (second)
THI	Technische Hochschule Ingolstadt
TN	True Negative
TP	True Positive
U	Voltage (Volts)
UK	United Kingdom
VP	Vacuum Pump
y	Actual Value
1D	One dimensional
2D	Two dimensional
3D	Three dimensional
e.g.	exempli gratia/ example
%	Percentage
f(x)	Predicted Value
$\nabla J(\theta)$	Gradient of loss-function
$J(\theta)$	Gradient function with respect to parameters θ
x_t	input vector (m x 1)
h_t	hidden layer vector (n x 1)
o_t	output vector (n x 1)
b_h	bias vector (n x 1)
u, v	parameter matrices (n x m)
σ_h, σ_y	activation functions
η	Learning rate
σ	sigmoid activation function

W_f	weight vector of the forgot gate
b_f	bias vector of the forgot gate
\tanh	hyperbolic tangent activation function
W	learned weight vectors
b	learned bias vectors
Φ	Phase Angle

1. Introduction

This chapter outlines the relevance of this thesis, summarises related work, and closes by describing the organizational framework of subsequent chapters.

1.1 Background and Motivation

Globally, governments are facing four common energy issues: increasing renewable energy share in the electric grid, decrement of fossil fuel reserves, the effects of changing climate, and obtaining a sustainable energy supply (MacKay, 2010, pp. 2–11). Moreover, with global population growth, the energy demand will rise with negative implications on the environment (e.g., CO₂ emissions) (Zoha *et al.*, 2012, p. 16839). Substantial energy waste can be prevented through fine-grained monitoring of energy consumption and passing that information back to the relevant consumers (Vine *et al.*, 2013, pp. 7–15). Renewable energy is a clean solution, but it is variable and depends on exogenous weather conditions. Germany is often considered a front runner in using various sources of renewable energy. According to the Federal Ministry for Economic Affairs and Energy, in the year 2018, the government intended to expand the increase renewable energy share in gross electricity consumption to 45% by the year 2025 and at least 80% by the year 2050, compared to its share of 36% in the year 2017 and only 3% in the early 1990s (Zerrahn *et al.*, 2018, pp. 259–279; Federal Ministry for Economic Affairs and Energy, 2016). The progressive transformation of the German energy system from conventional and centralised power plants to decentralised renewable power plants introducing considerable challenges in the nationwide electricity grid. To ensure the electric grid stability and security under such circumstances, the implementation of the smart grid represents a promising approach. It demands a digital and transparent infrastructure for interaction among grid operators, energy suppliers, and consumers to have an eye on precise energy generation and consumption, which can help design optimization techniques.

The Smart grids are usually supported with smart metering systems, which allows the utility companies to monitor the grid more truthfully, which will enable them to detect the failures rapidly, to regulate the generation more dynamically, to adapt the pricing more smartly, and to predict the demand more accurately (Zhang *et al.*, 2019, pp. 23–48). In 2016, with the introduction of an act ‘Digitization of the Energy Turnaround Act’ (Federal Ministry for Economic Affairs and Energy, 2016; Kelly, 2018), a green signal was given for the use of smart meter, and that also boosted the funding for the research and development in the field of the smart meter, which came up as the central component for the communication infrastructure (BUNDESMINISTERIUM FÜR WIRTSCHAFT UND ENERGIE, 2019; Scully, 2019). According to a detailed review based on around 60 feedback studies suggest that direct feedback mechanisms can help to achieve

maximum energy saving (i.e., real-time appliance level energy consumption information) as compared to indirect feedback mechanisms (i.e., regular monthly advice along with the energy usage bill). Traditional smart meters have limitations to measure energy consumption up to house level. Thus, to achieve it, research efforts lead to the development of several power disaggregation techniques of load monitoring (Ehrhardt-Martinez *et al.*, 2010, pp. 14–29). The disaggregation technique of load monitoring was initially targeted to residential consumers, but it proved beneficial for industries equally with the features like fault detection. The concept of load monitoring is decades old but recently gained renewed attention in the research area with the parallel development of sensing technology, data communication and networks, artificial intelligence, and machine learning. Among the several techniques, the NILM approach got its popularity due to the least requirement of measuring devices (Zoha *et al.*, 2012, pp. 16838–16866).

Non-Intrusive Appliance Load Monitoring, also known as Non-Intrusive Load Monitoring, was first introduced in 1985 by an American geometer named George W. Hart at Massachusetts Institute of Technology (MIT) (Hart, 1985, p. 2). Another milestone was achieved in 1998 when Cole and Albicki proposed their theory to take into account power spikes (Cole and Albicki, 2000, pp. 1–6), which typically occurs in load profile at the time of switch on or off. They suggested to use it as a ‘signature’ to identify appliance. Such events are commonly known as transients (Fiol, 2016, p. 7). In the early 2000s, the new era of algorithms came when the meter technology improved. With the new features like harmonics and signal waveform, the signal processing technology changed, such as, now using Fourier transform, ‘power signature’ could be detected. A boom in machine learning approaches has been seen in recent years. In 2010, an algorithm named DDSC was introduced by J. Zico Kolter (Kolter *et al.*, 2010, 1153–1161). Jack Kelly and William Knottenbelt introduced another very popular and recent algorithm for disaggregation using neural networks. They targeted the residential sector based on mainly United Kingdom (UK) based datasets (Kelly and Knottenbelt, 2015, pp. 55–64). Figure 1 shows that the period of the year 2007-2011 was booming for NILM development. Both the small and large-scale companies (4home, PlotWatt, Enetics, Navetas, Belkin, GE, IBM, Intel) jumped in the development of NILM (Zeifman and Roth, 2012, p. 3). Though, the energy sector is still waiting for a rigorous, reliable, and robust algorithm for energy disaggregation.

There is a growing appeal for the power disaggregation for industrial buildings (Kelly, 2017, pp. 1–4). In this thesis, the dairy farm industry is targeted for the development of a power disaggregation algorithm. The agricultural sector has not yet gained considerable attention for research in this field. Since the rural areas in Germany are characterised by highly heterogeneous renewable power producers, and they are predominantly affected by the issue of power grid overload, smart energy management solutions in agriculture.

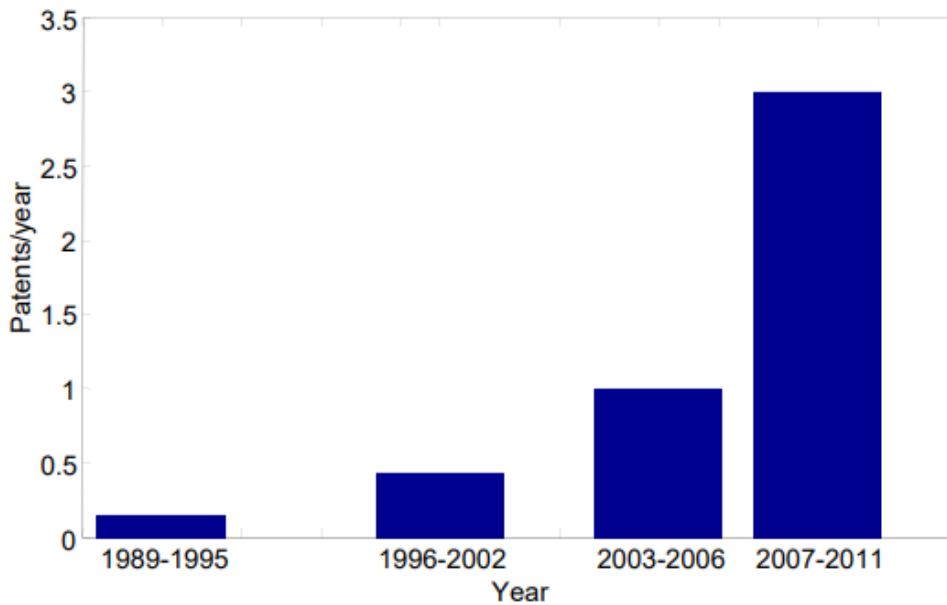


Figure 1 Significant growth in U.S. granted patents (Zeifman and Roth, 2012, p. 3)

They can play a significant role in a comprehensive promotion of sustainable energy supply. In this context, dairy farms as an energy-intensive category of agriculture show high potential for power grid-oriented demand-side management in addition to benefits such as comprehensive energy monitoring and the identification of energy savings potentials. The primary consumer in dairy farming with an average share of 60% in total electricity consumption, the milking production appliances are focused in the thesis as shown in Figure 2.

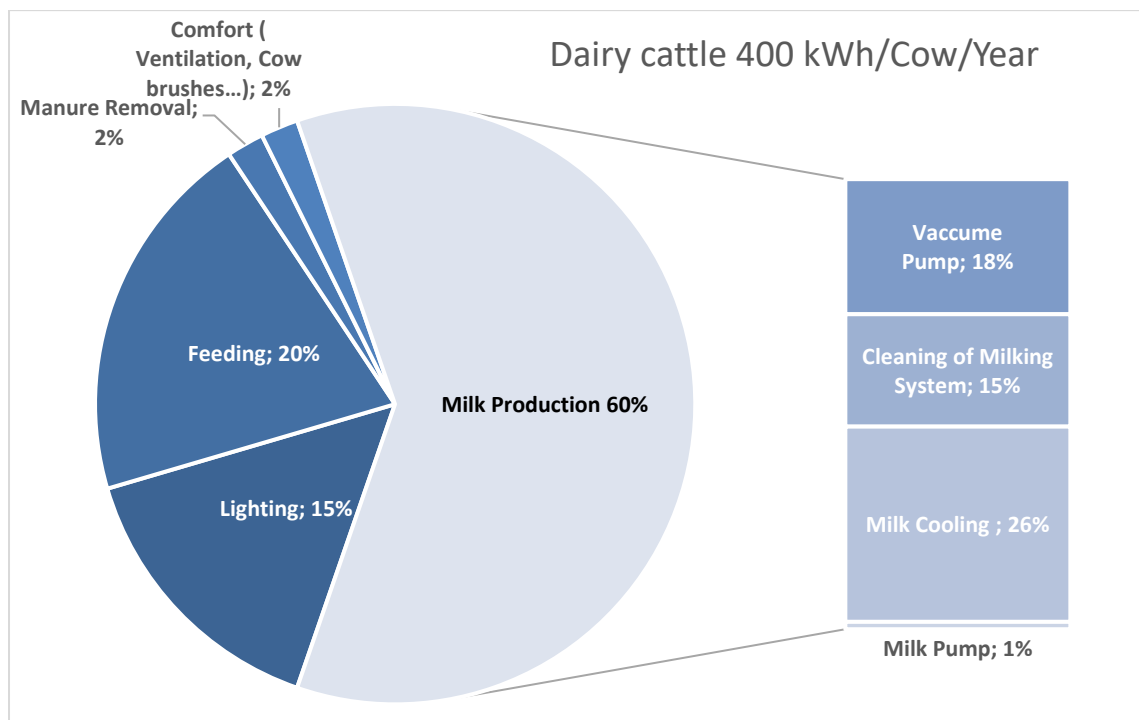


Figure 2 Composition of electricity consumption in dairy farming (Neser et al., 2014, p. 14)

Besides the monitoring and benchmarking, energy disaggregation provides different application fields in the context of demand-side management and load shifting, which

represent a base for promising technological innovations and business models in the agricultural sector. Use of NILM based algorithm and a comprehensive On-farm energy management system, significant energy-saving improvements, and additional revenue streams for local farmers can be provided. Furthermore, the intelligent system allows sufficient load management for future developments such as increased demand for electrical capacities for e-mobility. In addition to that, there is an additional benefit of NILM. It also allows the appliance manufacturing companies to better understand the machines and their operational pattern on the ground level, improving the performance and better maintenance plans (Zoha *et al.*, 2012, pp. 16838–16866).

1.2 Goal of the thesis

This main objective of the present thesis is a comparative analysis of existing deep learning based NILM-methods and the development of a Python-based algorithm, mainly using Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs). This project is a composite of:

- Discussion of the overview of the load monitoring techniques.
- Literature research on deep neural network with an overview of Machine Learning.
- Measurement of the load profiles of individual appliances from several dairy farms.
- Development of an algorithm, using multiple features to identify the ‘**on-off**’ state of selected appliance from the aggregated load profile of a dairy farm.

1.3 Research Questions

1. How can Deep Learning be used to provide useful solution for load monitoring?
2. Which method is suitable for the NILM?
3. How does the accuracy change with one and multiple features?

2. Theoretical Background

This chapter presents an overview of the concepts of load monitoring approaches and technologies, machine learning, and deep learning methods, which are essential to understand the methodology, implementation, and evaluation results presented in the thesis work.

2.1 Load Monitoring

2.1.1 Event and Event-less Load Monitoring Approach

Early days studies made on NILM were focused on an event-based approach. In this approach, studies performed by disaggregating the total power consumption profile by detecting and labelling every single appliance linked to it. A typical example of event-based energy disaggregation is presented in Figure 3. The main idea is to classify switching events of appliances (e.g., a Refrigerator turning on or off) from the load profile. Such features can be detected based on the changes in the power demands at the time of switching on-off of the device. For this, it is crucial to have high-frequency data measurements, which should be 1Hz or above. For low-frequency data, it becomes difficult to discriminate among the switching behaviour of different appliances (Parson, 2014, pp. 1–31; Parson *et al.*, 2014, pp. 1–19). Furthermore, this approach shows a performance limitation. The measurements being used for this project are in the low frequency range. Therefore, this approach will not be considered for this work.

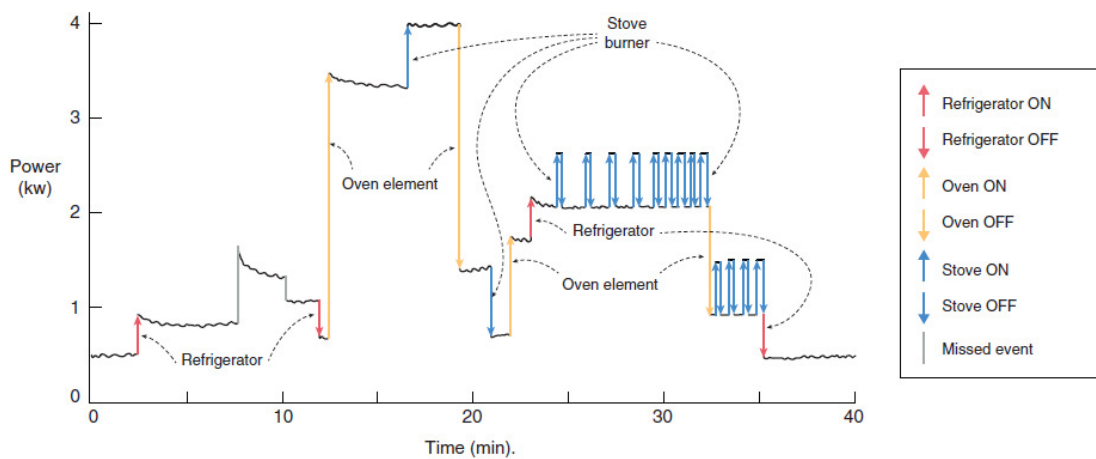


Figure 3 An example of event-based energy disaggregation. (Copyright 1992 IEEE) (Pereira and Nunes, 2018, p. 2)

In contrast to the event-based approach, event-less approaches do not require a separate event detection and classification process. In this approach, every sample of power consumption of a specific appliance attempts to match with the aggregated power sample measured in the same time frame employing machine-learning. A classic example is presented in Figure 4. The fundamental advantage of this approach is that training dataset does not require any labelling. Only the aggregated and specific power consumption data of the appliances are required, which turns this approach more economical and straightforward compared to an event-based approach (Pereira and

Nunes, 2018, pp. 1–17).

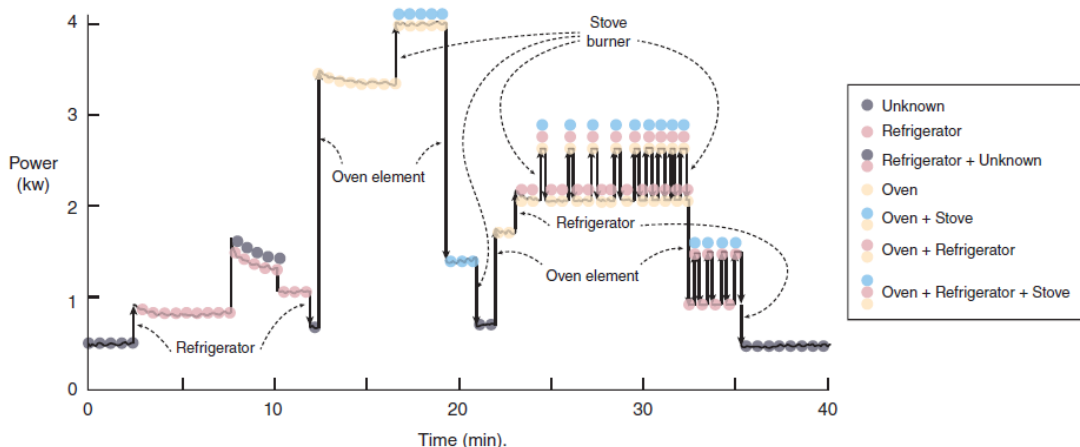


Figure 4 Example of event-less energy disaggregation. Copyright 1992 IEEE (Pereira and Nunes, 2018, p. 2)

2.1.2 Intrusive and Non-intrusive Load Monitoring

Intrusive load monitoring is a decentralised method of measurement. It requires several sensors, power measuring devices, communication devices, and other equipment based on the number of appliances connected to ensure the precise measurements and to transmit the information for monitoring. Hence, it is a cost intensive and impractical option of load monitoring. Intrusive load monitoring can be categorised as direct and indirect monitoring (Parson, 2014, pp. 1–31; Parson *et al.*, 2014, pp. 1–19).

In the **direct intrusive load monitoring**, every individual appliance needs to be connected to a power measuring device or a sensor to measure the electric characteristics of that appliance. There are mainly three methods to accomplish direct load monitoring. The first method can be named as Electrical Sub-metering. In this method, individual appliances are monitored by one meter per appliance. Such meters are mostly of two categories, i.e., either Plug-in meter or Clamp-on meter. They have the capability to monitor appliances as well as control the flow of electricity. The upgraded version of this method is known as the Smart Appliances method. In which the smart appliances can self-report the specific power consumption to a central hub. These devices can be connected with wired or wireless technology, although old appliances might not be suitable for this operation and hence need to be replaced or modified, which entails significant initial investment. Thus, for a large user base, these approaches are not economic. The third method of intrusive load monitoring is known as Electrical Probing. In this method, the appliances are designed to send an additional signal into the electric circuit to determine the mode of operation on-off to a central hub, which is further used for feature extraction. This technology has a drawback that the power quality delivered to the appliance gets considerably affected due to this injected signal in the electric circuit. All of the mentioned technologies are expensive because of the involvement of enormous numbers of measuring and communication devices and their installations (Parson, 2014, pp. 1–31).

Indirect intrusive load monitoring is not limited to the measurement of the appliance power. With this method, other parameters, which are influencing power consumption are also being measured. Likewise, direct load monitoring, indirect load monitoring can also be accomplished by mainly three methods. The first method is known as Appliance Tagging. Every individual appliance needs to be modified in such a way that it can emit a unique signal in the main circuit, which indicates the central hub about the turning on-off of that appliance. Thus, the central hub estimates the power consumption of each appliance with time. The second method is called Ambient Sensors. Multiple wireless sensors are required to monitor several measures, including audio signals, ambient temperature, device temperature, light sensors, and others, which have an impact on the appliance usage. The main idea is to identify the relations among them and disaggregate the consumption of the appliances as well as identify human behaviour. The third method in this category is the Conditional Demand Analysis. Unlike previous mentioned methods, this approach needs only electricity consumption bills of a household. Although it demands a massive database of multiple houses and along with that, it also requires a detailed questionnaire from each household. These questionnaires include the details of the consumers, weather, the usage behaviour of appliances like the number of times per day and the number of hours. Based on the collected database, a multivariate regression technique is performed to predict and analyse the appliances. Due to the lack of sensors and measurement devices requirement, this method can get confused with the non-intrusive load monitoring technology. However, the requirement of parameters other than the electricity consumption makes it fall under the category of intrusive load monitoring (Parson, 2014, pp. 1–31). Being ruled out intrusive load monitoring methods as appropriate solutions for the problem of smart meter energy disaggregation, the industry is turning to the non-intrusive load monitoring method.

Non-intrusive load monitoring is the method, which requires only aggregated power consumption data. For training purposes of a program or a machine, it needs a massive database of the aggregated and individual appliance electrical power consumption for the same time frame from various but similar kinds of facilities (Nascimento, 2016, pp. 4–7) . It is essential to measure various electrical parameters, e.g., the voltage, the current, power factor for the practical training of the machine. These are known as 'features' in machine learning terminology. More and more features enhance the accuracy of the predictions. Thus, unlike intrusive load monitoring, NILM does not require a high number of measurement devices, that is the advantage of this method as it enhances the reliability and precision of measurement. Only for the training of the machine, it needs several meters based on the number of appliances. Although studies performed with low-frequency data also delivered quite promising results (Nascimento, 2016, pp. 4–7).

2.2 Machine Learning

Machine learning (ML) is a subfield of AI introduced in the late 1950s. ML is the study of computer algorithms that automatically improve computer programs through experience as defined by Tom Mitchell. In the past few years, ML techniques (Graville, 2017) have provided solutions to problems such as classification, regression, density estimation, and forecasting (Palma, 2016, p. 2). The approach has gained relevance in various application domains such as bioinformatics, speech recognition, Spam and fraud detection, and social networks. Machine learning can be categorised into as supervised and unsupervised learning.

Supervised learning for training the model is when individual appliance consumption required along with aggregated power consumption data. Thus, the intrusive load monitoring method is performed to collect the data at the appliance level. Most popular supervised learning techniques are:

- Inferring Rules
- Statistical Modelling
- Support Vector Machines
- Decision Trees
- Linear Models-Logistic Regression
- Neural Networks
- Instance-based Learning

For the present thesis, a Neural Networks technique is adopted, which is comparatively newer but showed good results in research.

Unsupervised learning does not require the appliance level data. The model is trained only with the aggregated data sets. It is much more challenging to achieve a good result with unsupervised learning in comparison to supervised learning. However, it is a very desirable method due to the requirement of less hardware such as measuring devices.

Semi-supervised learning is a combination of both. The model is trained based on labelled and unlabelled data set, which reduces the requirement of measurements up-to a considerable amount. For the solutions of NILM, this is one of the most suitable methods. They are different in terms of training and validating the model (Bernard, 2018, pp. 18–19; Nascimento, 2016, p. 6; Figueiredo, 2013, pp. 38–55).

In the next sub-topic of this section, a class of probabilistic graphical models addressing the short-comings of the event-based approaches is discussed. They have been applied to several real-world applications, speech recognition (Rabiner, 1989), which share a number of similarities with energy disaggregation. The aim is to identify the most likely sequence of discrete states(words) corresponding to a time series of continuous

measurements (audio recordings). There are some methods proposed for disaggregation using the Machine Learning technique. An overview of these models is given below.

2.2.1 Markov Chains

Markov Chains are a stochastic and memory-less process, which can be performed on a database having a finite number of states. This method is quite useful for time series databased problem. To perform this, a set of variables is required, and these variables must be indexed. The most common case is a timely indexed set of variables that form a series, which represents the overall evolution of the process. 'A stochastic process is characterised by random state changes. The memory-less property, which is known as Markov Property, states that the state of a system at any time 't' is only dependent on the state of its previous time step 't-1' (Fiol, 2016, pp. 8–12)'. During the transitions from one state to another state follow a certain probability distribution.

'A graphical representation of a Markov Chain with a directed graph having a set $S = \{1,2,3\}$ and the transitions represented with edge label is presented in Figure 5 (Fiol, 2016, pp. 8–12).'

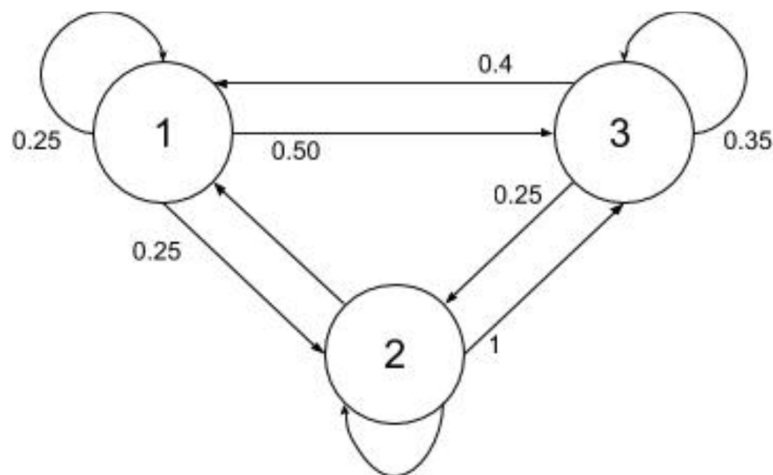


Figure 5 Basic Markov chain represented in graph form (Fiol, 2016, pp. 9–10).

In this case the transition matrix is the following:

$$p = \begin{pmatrix} 0.25 & 0.25 & 0.5 \\ 0 & 0 & 1 \\ 0.4 & 0.25 & 0.35 \end{pmatrix} \quad (\text{eq. 1})$$

If the initial state is considered $x_1 = (0.5; 0.5; 0)$, It can be seen that the distribution over the states at time $t = 1$ will be $x_1 * p = (0.125; 0.125; 0.75)$. Thus, the probability distribution over the states of the system at any time t can also be calculated (Fiol, 2016, pp. 8–12).

'Markov Chains was widespread for being used in several different applications such as ranking websites in search-engines or generating sequences of numbers that follow the desired distribution. However, in numerous cases the true state of the model cannot be measured, and Markov chains got fall short- for such cases, a more robust model was

created known as hidden Markov model (Fiol, 2016, pp. 8–12).

2.2.2 Hidden Markov Model

This model is known as Hidden Markov Model (HMM) as its states are not visible to an observer, but the output associated to them are evident in the form of data, which is also known as a token. These tokens give information about the states. An example of HMM is shown in Figure 6.

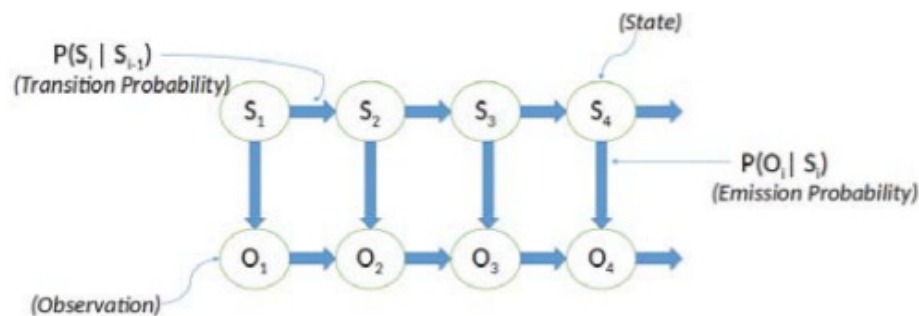


Figure 6 Example of HMM showing the states and observations (Raiker et al., 2018 - 2018, pp. 381–385)

HMM is a well-established method for solving evaluation problem, decoding problem, and learning problem. Therefore, a model for NILM is capable of calculating the probability of a particular observation sequence, which extract information if there is a pattern in on-off switching. An HMM-based model can also find the most likely sequence of states which further generates an associated sequence of observations for prediction purposes. Thus, HMM is an efficient method in extracting the features, which is useful for NILM (Raiker et al., 2018 - 2018, pp. 381–386).

2.3 Deep Learning

Deep learning is a subset of ML, while ML algorithms build their learning process around the input data structure; deep learning-based algorithms use layers of Artificial Neural Networks (ANN) for the learning process. In this work, ANN are used in analogous to AI. Frank Rosenblatt introduced the first artificial neural network called perceptron in 1958 (ROSENBLATT, 1958, pp. 386–408). The idea of ANN is derived from the human biological neurons, which helps us gain various skills by performing the tasks and learning from it. Deep Learning in neural networks denotes the network with many layers (Arnold et al., 2011, pp. 2–13). “Deep Learning allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction” (LeCun et al., 2015, pp. 436–444). The main objective is to learn the hierarchy of features by discovering the convoluted structure in the training dataset with the help of a backpropagation algorithm to identify how a machine should update its internal parameters, which are further used to compute the output for the upcoming layer from the previous layer (LeCun et al., 2015, pp. 436–444; Arnold et al., 2011, pp. 2–13).

2.3.1 Basic Concept and Brief History

Deep Learning models have proven the capabilities of end-to-end learning. They are

flexible, which allows a model working in several applications with a slight modification based on the type of training data sets. These models are capable of extracting feature representations and learn them automatically.

Deep Architectures have been in research for several years but could not get any big breakthrough until the 20th Century. More specifically, in 2006 and 2007 significant research began to be capable of training deeper network (Hinton *et al.*, 2006, pp. 1527–1554), the exception it was the CNN used by LeCun (Lecun *et al.*, 1998, pp. 2278–2324; Nascimento, 2016, pp. 10–11). The new success came with the recent development of several platforms and libraries, which included new optimization techniques and architectures. Also, the large amount of training data is essential for the deep neural network, which improved significantly in the last decades. The computational power of the recent **graphical processing units** (GPUs) has become the backbone of this new era of Artificial intelligence development. Training of a model is only useful when it is provided a large set of data frames. It could extract and learn enough features, but it takes hours by a **central processing unit (CPUs)** for processing such huge amount of data even though its configuration is very high and mighty. GPUs have resolved this issue and cut down the processing time from days to few hours (Nascimento, 2016, pp. 10–12).

The latest approaches in NILM are based on deep learning. Firstly in 2015, Jack Kelly and William Knottenbelt introduced the deep-learning based approach (Kelly and Knottenbelt, 2015, pp. 55–58). They proposed a few models, including Convolutional Neural Networks, Recurrent Neural Networks, and Denoising Autoencoder. Another development was from Anders Huss in the same year; he proposed a hybrid algorithm based on CNNs and a hidden semi-Markov model (Huss, 2015, pp. 1–3). Since then, several developments took place in the last couple of years. In 2019, Zang *et al.* proposed sequence-to-point learning also based on majorly CNN (Zhang *et al.*, 2019, pp. 23–26), they proposed a model using five hidden convolutional layers, and numerous other attempts have been made in solving NILM using deep learning (Kim *et al.*, 2017, pp. 1–5; Valenti *et al.*, 2018 - 2018, pp. 1–8; Rafiq *et al.*, 2018 - 2018, pp. 234–239).

2.3.2 Deep Learning Libraries

Deep learning libraries are the pre-made set of functions or modules which can be called through any program. There are several open-source libraries that are maintained by major industrial stakeholders. On the commercial level, there are some libraries like **Caffe** developed by the University of California, Berkeley. It is written in C++, with a python interface (Jia, Yangqing and Shelhamer, Evan and Donahue, Jeff and Karayev, Sergey and Long, Jonathan and Girshick, Ross and Guadarrama, Sergio and Darrell, Trevor, 2014). Another library is **Deeplearning4j**, which is written in JAVA Virtual Machine (JVM), which was released under Apache license 2.0 (Nicholson and Kokorin, 2013). The

Torch is another machine learning library based on Lua programming language. The development of Torch has been stopped; however, **PyTorch**, which is Torch based, is actively developed as of June 2020. PyTorch is a Python package, and it provides two high-level features, i.e., Tensor Computation and Deep neural networks built on tape-based ‘autograd system’(Chanan *et al.*, 2020). **Theano** is one of the most established Python-based libraries. It was developed by Montreal Institute for Learning Algorithms (MILA) at ‘Université de Montréal’. It was developed in 2007, and the latest version was introduced in 2017. Theano is mostly popular among the university students for research and project purposes. Its application programming interfaces (APIs) are easy to call, which makes it very simplified to use for programming. The accuracy of the algorithms has made it so much popular (Montreal Institute for Learning Algorithms, 2007).

The deep learning library used for the present work is called **TensorFlow**. It is also a free and open-source software library for machine learning, which is developed and continuously maintained by Google Brain Team. It was first released under the Apache License 2.0 in 2015. This library is very versatile and written in Python, C++, and CUDA. It can be run on CPUs or GPUs, and even on mobile operating systems (Agarwal *et al.*, 2015).

2.3.3 Artificial Neural Network

The ANN consists of fundamental adaptive blocks called artificial neurons or nodes densely connected. They are capable of processing information and have the characteristics of handling data with non-linearity, fault and noise tolerance, and generalization capabilities in the learning process (Basheer and Hajmeer, 2000, pp. 3–31). Generally, a group of nodes arranged together forms a layer, and the connection between the layers defines the architecture of the neural network and its capabilities or characteristics as shown in Figure 7.

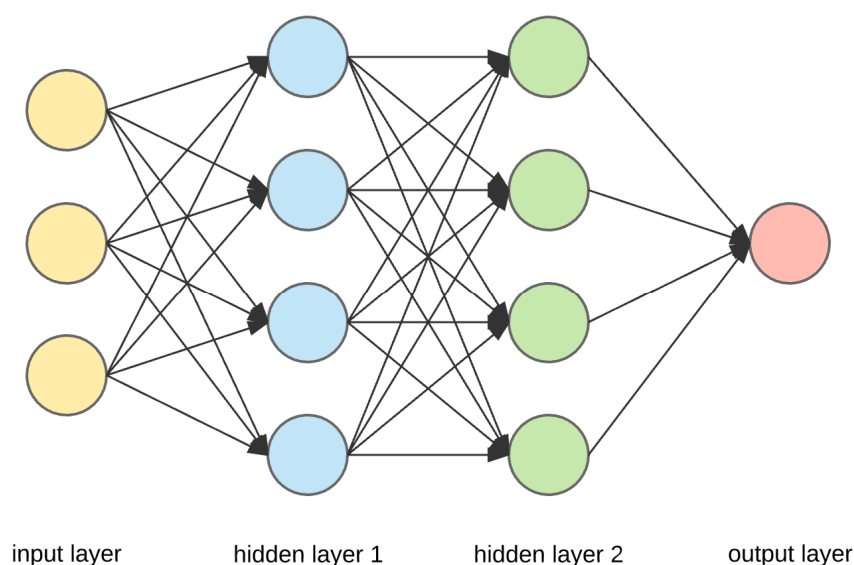


Figure 7 A simple feed-forward neural network where nodes in each layer are connected to all the nodes in the corresponding layer (Dertat, 2017)

If the ANN consists of more than one layer, it is called a multi-layer feed-forward neural network. When all the nodes of one layer are connected to all the nodes of the corresponding layer without any loop, it forms a fully connected feed-forward neural network. It is a simple form of neural network and consists of an input layer, a hidden layer, and an output layer (Basheer and Hajmeer, 2000, pp. 3–31).

2.3.4 Deep Neural Network Libraries

KERAS is an open-source Neural Network Library written in Python programming language. It is a high-level API and runs on top of Libraries like Theano or TensorFlow. It was developed Francois Chollet and released in 2015. The stable release came in June 2020 as version 2.4.0. Its APIs are easy to use and have proved themselves with high-quality performances. It contains plentiful implementations of frequently used neural-network building blocks such as layers, activation functions, optimizers, which are going to be discussed in detail (Chollet 2015).

2.3.5 Back Propagation

Gradient descent is the most established first-order optimization algorithm. Backpropagation aims to update the output weight and hidden weights in the neural network during an iterative process so that the new weights cause the output to be closer to the target. A **weight** can be defined as a parameter which transforms the input data to hidden layers within a neural network. The updates of weights are based on the partial derivative of the total error with respect to hidden weights and output weights, which can be express with the equation below. The algorithm starts with a random guess at the parameters and tries to configure the parameters in the direction in which the loss function steeps downward the most. This process is repeated until the lowest point in the loss function is found (Biansoongnern and Plangklang, 2016 - 2016, pp. 1–4).

$$\theta = \theta - \eta \cdot \nabla J(\theta) \quad (\text{eq. 2})$$

Where,

η : Learning rate

$\nabla J(\theta)$: Gradient of loss-function

$J(\theta)$: Gradient function with respect to parameters θ

The process of learning in neural networks, using gradient descent and backpropagation can be summarised as follows. A neural network is trained by using backpropagation in which, it first propagates forward, calculating the dot products of the input and their corresponding weights. An activation algorithm is applied to this weighted sum, which transforms the input signals to the output signal. It also introduces non-linearity into the model, which allows the present model to learn the complex relationship between the inputs and outputs. After this, it propagates backwards in the network carrying the error

terms. While moving backwards in our network, it updates the weight values using gradient descent. The iterative update of weights is performed by calculating the gradient error function with respect to the weights or the parameters. Afterwards, it updates the parameters in the opposite direction of the gradient of the loss function. This process repeats until the local minima are determined.

2.3.6 Recurrent Neural Network (RNN)

Recurrent neural networks are another branch from the traditional feed-forward networks, which, as the name suggests, is recurrent or has loops in its architecture. Figure 8 shows the general schema of an RNN unit. The loop provides added advantages to this architecture over the simple feed-forward neural networks. This looping enabled RNNs to have self-sustained temporal activation dynamics in the absence of input and recurrent connection networks.

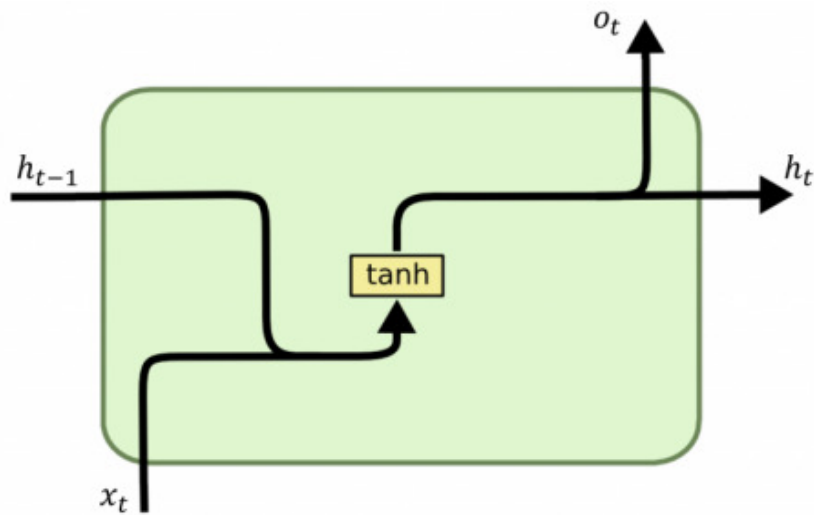


Figure 8 Recurrent neural network (dprogrammer, 2019)

When the input data is provided to the RNN, it preserves this data in an internal state, which means that RNNs have a dynamic memory (Lukoševičius and Jaeger, 2009, pp. 127–149). These make the RNN stand out for applications where long sequences of data should be memorised. Hence, RNNs are used in NILM. The approach is also used in the present thesis work. The following equations represent in case of feed-forward:

$$h_t = \sigma_h(i_t) = \sigma_h(u_h x_t + v_h h_{t-1} + b_h) \quad (\text{eq. 3})$$

$$y_t = \sigma_y(o_t) = \sigma_y(W_y h_t + b_h) \quad (\text{eq. 4})$$

where,

x_t : input vector ($m \times 1$)

h_t : hidden layer vector ($n \times 1$)

o_t : output vector ($n \times 1$)

b_h : bias vector ($n \times 1$)

u, v : parameter matrices ($n \times m$)

v : parameter matrix ($n \times n$)

σ_h, σ_y : activation functions

Figure 9 shows the unrolled RNN, where the green colour box is the neural network, and the arrow indicates the feed-forward. The current time step and previous time step are feed to the next input, as shown.

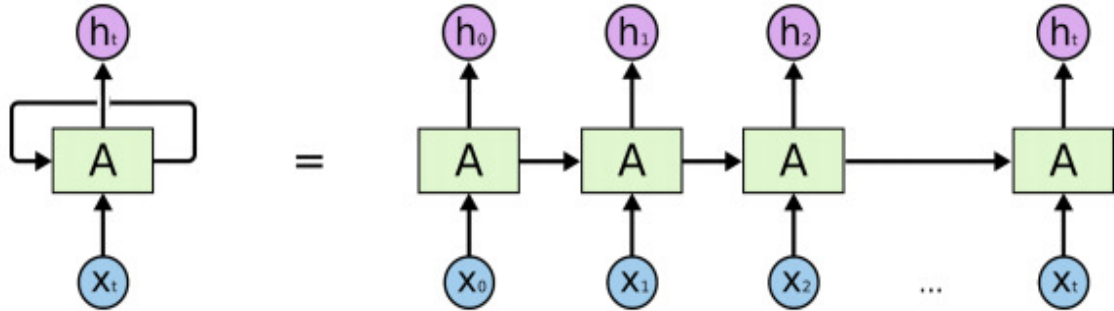


Figure 9 An unrolled recurrent neural network (oinkina et al., 2015)

2.3.7 Long Short-Term Memory (LSTM)

Long short-term memory (LSTM) network is a category of RNN, which can solve sequence prediction problems with learning order dependence capability. Standard RNNs suffer from vanishing and exploding gradient problems. LSTM provides solutions to these problems by introducing new gates such as input and forget gates. It consists of four layers interacting in an especial manner. The architecture shown in Figure 10 enables an LSTM network to remember information for a long time (Hochreiter and Schmidhuber, 1997, pp. 1735–1780). The working principle and the mathematical formulation of an LSTM network are explained below.

The state update h_t , and the output o_t is calculated as follows:

$$h_t, o_t = f(x_t, h_{t-1}) \quad (\text{eq. 5})$$

It consists of a forget gate f_t , which decides what information not to remember. This gate contains a sigmoid activation function, and if the value of it is zero, then the value is thrown away.

$$f_t = \sigma(W_f * [h_{t-1}, x_t] + b_f) \quad (\text{eq. 6})$$

where,

σ : sigmoid activation function

W_f : weight vector of the forgot gate

b_f : bias vector of the forgot gate

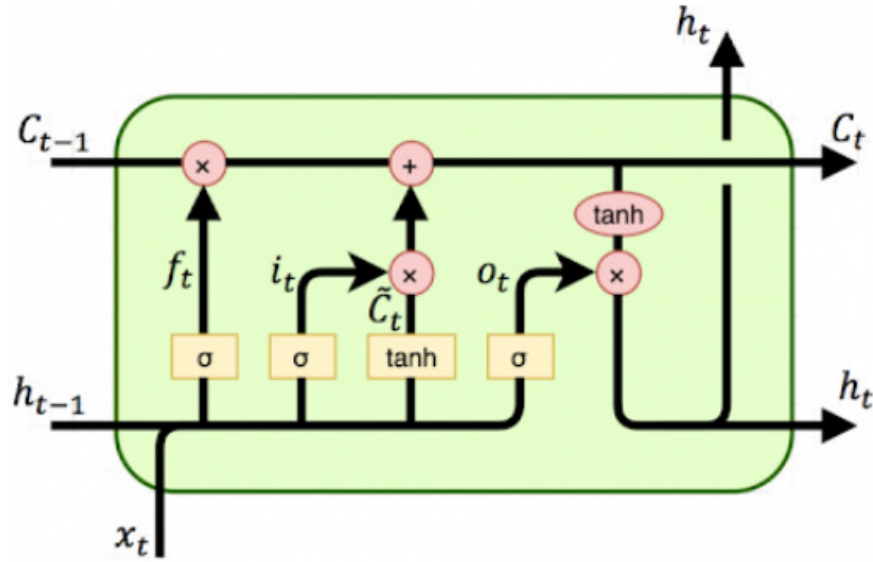


Figure 10 The LSTM network consists of four interacting layers: input gate, output (oinkina et al., 2015)

The next layer is the update gate layer in which the sigmoid layer decides what information to store in the cell state. The layer ‘tanh’ creates a vector of new candidate values \tilde{C}_t that could be added to the state.

$$I_t = \sigma(W_i * [h_{t-1}, x_t] + b_i) \quad (\text{eq. 7})$$

$$\tilde{C}_t = \tanh(W_c \Delta [h_{t-1}, x_t] + b_c) \quad (\text{eq. 8})$$

where,

\tanh : hyperbolic tangent activation function

W : learned weight vectors

b : learned bias vectors

After this, the cell states are updated,

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \quad (\text{eq. 9})$$

The output layer o_t , is a filtered version of the gate state. First, the sigmoid layer is used to decide what part of the cell state to output. Then the cell state passes through a ‘tanh’ activation function and multiplies with the output of the sigmoid layer.

$$o_t = \sigma(W_o [h_{t-1}, x_t] + b_o) \quad (\text{eq. 10})$$

$$h_t = o_t * \tanh(C_t) \quad (\text{eq. 11})$$

2.3.8 Gated Recurrent Unit (GRU)

Gated Recurrent Unit (GRU) is a featured and efficient variant of LSTM. It maintains the effect of LSTM while making the structure simpler (Macal and North, 2010, pp. 151–162). GRU combines the input and forget gate of LSTM into an update gate, and the output gate in LSTM is named as reset gate in GRU. Update gate determines how to combine the new input with the previous memory and reset gate determines how much of the

previous memory can pass through. GRU model structure is shown in Figure 11.

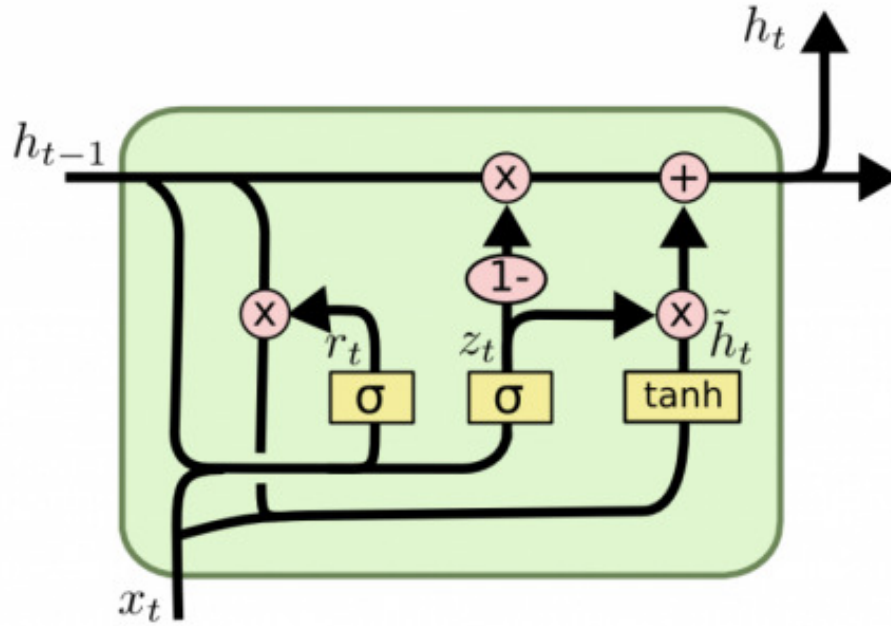


Figure 11 Gated recurrent units (oinkina et al., 2015)

Below mentioned equations represent the mathematics during feed forward:

$$Z_t = \sigma(W_z * [h_{t-1}, x_t] + b_z) \quad (\text{eq. 12})$$

$$r_t = \sigma(W_r * [h_{t-1}, x_t] + b_r) \quad (\text{eq. 13})$$

$$\tilde{h}_t = \tanh(W_h \cdot [r_t \odot h_{t-1}, x_t] + b_h) \quad (\text{eq. 14})$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \quad (\text{eq. 15})$$

where,

h_t : hidden layer vectors

x_t : input vector

b_z, b_r, b_h : bias vector

W_z, W_r, W_h : parameter matrices

σ, \tanh : activation functions

2.3.9 Bidirectional Recurrent Neural Network

Bidirectional recurrent neural networks (BDRNN) have the capability to connect the two hidden layers of opposite directions, i.e. from past(backward) and future(forward) to the one output, as shown in Figure 12. Thus, the output layer can get information from both past and future states simultaneously, which enhances the learning of the model.

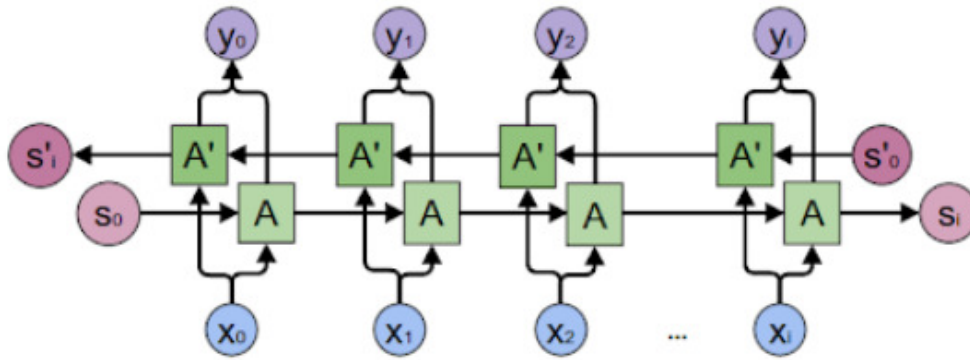


Figure 12 General structure of Bidirectional recurrent neural networks (oinkina et al., 2015)

2.3.10 Convolutional Neural Network

The convolution layer is the primary and first building block of a convolution neural network (CNN). Convolution is a linear operation between an array, called **kernel or filler** having a set of weights and the input. The filters are much smaller in size than the input data size. In general practice, it is in the range of 1*1 to 7*7 blocks. The multiplication between the filter and the same size input data is a **dot product**, as shown in Figure 15b, which occurs element-wise. The sum of this dot product value generates a single value. The filter moves over the input dataset in the left to right and then in the top to bottom direction, as shown in Figure 13, Figure 14, and Figure 16. This filter gets multiply with different patches of input data, which are same in size (Brownlee, 2020a). This process is considered one step, and the output of this step is an extracted feature, which helps in training the model. This output gets transfer into the next fully-connected hidden layer in the model, as shown in Figure 13 (Brownlee, 2020a).

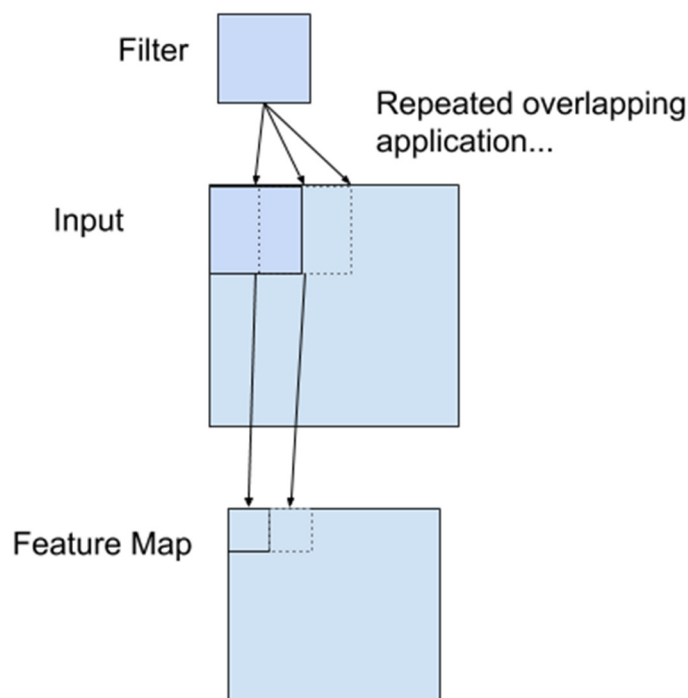


Figure 13 Example of a filter applied to a two-dimensional input to create a feature map (Brownlee, 2020a)

CNN can be categorised as 2D (two dimensional) and 1D (one dimensional) network. The main function of a convolutional layer is to extract the features from the raw data. 2D CNN is named because the kernel is two-dimensional, which slides in the third dimension, as shown in Figure 14 (Verma, 2019). Initially, 2D-CNN was designed to deliver state-of-the-art accuracy in tasks such as image recognition, speech recognition, object detection, and language translation (Krizhevsky *et al.*, 2012, pp. 1–9; Srinivasamurthy, 2018, pp. 6–15).

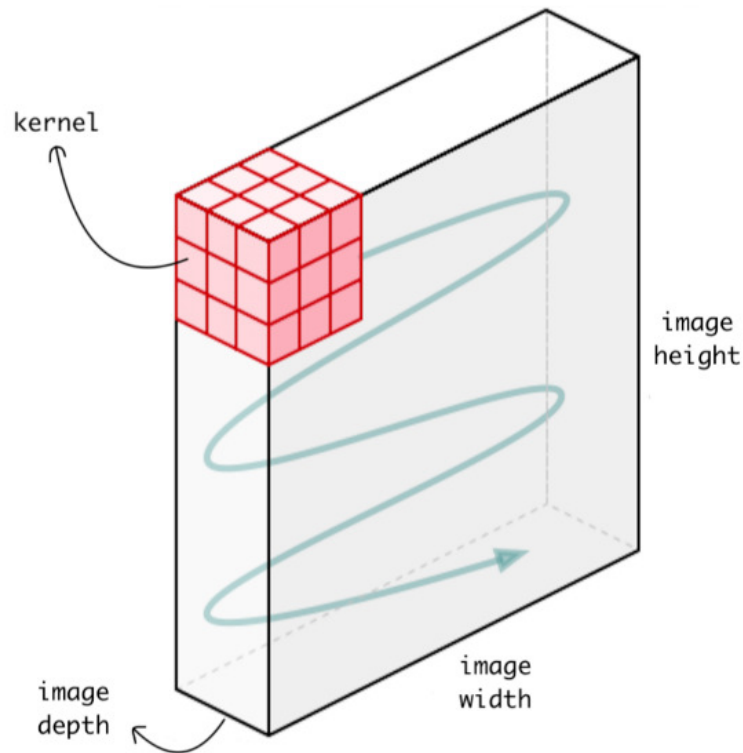


Figure 14 Kernel sliding over the Image (Verma, 2019)

1D Convolutional Neural Network was recently developed to handle the data in a 1D sequence of data, such as in signal processing, ECG classification, Cardiac Arrhythmias (Kiranyaz *et al.*, 2016, pp. 664–675; Kiranyaz *et al.*, 2015, pp. 2608–2611). There are several advantages of 1D-CNNs due to the following reasons (Brownlee, 2020b):

- 1D CNNs require simple array operations, which make the computational complexity lower. Such models run faster over GPUs but can also be trained on CPUs.
- Recent studies showed that the 1D CNNs have shallow architectures, which means that based on the quality of the database available for training, these networks need comparatively less hidden-layer (e.g., <2) and neurons (e.g., <50) for accurate results.
- Due to the low computational power requirements, it is more suitable for real-time operations, and it is also cost-effective.

The input datasets obtained for NILM algorithm are in 1D and CNNs have been an

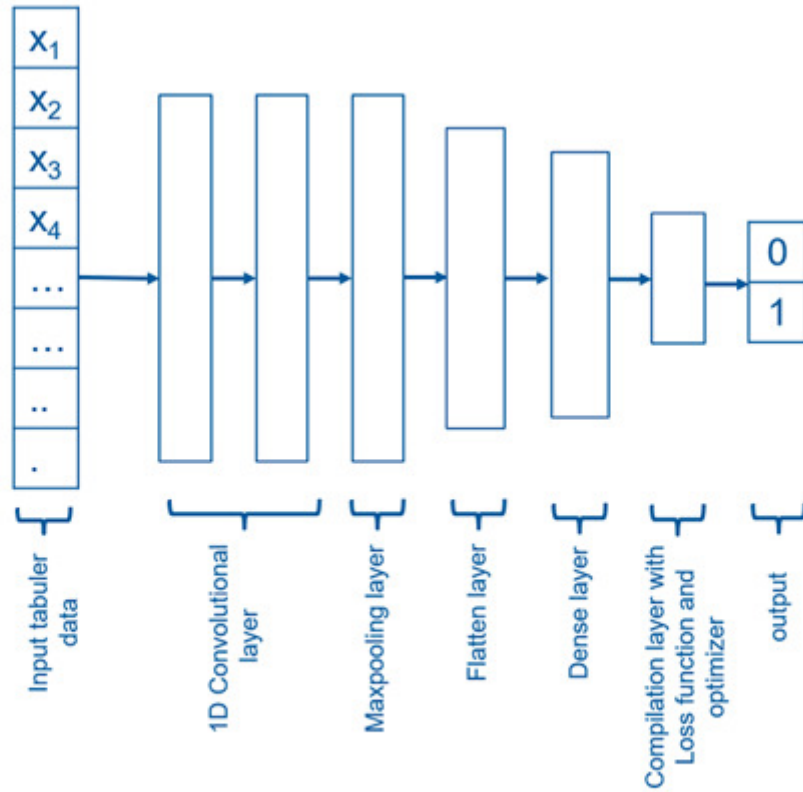
outstanding method in dealing with 1D dataset. Thus, the presented algorithms in this thesis primarily created using CNNs. The following hyper-parameters determine the configuration of a 1D-CNN model (Kiranyaz *et al.*, 2015, pp. 2608–2611):

- The number of hidden layers and neurons.
- Filter (kernel) size in each convolutional layer.
- Subsampling factor in each convolutional layer.
- The choice of pooling and activation operators

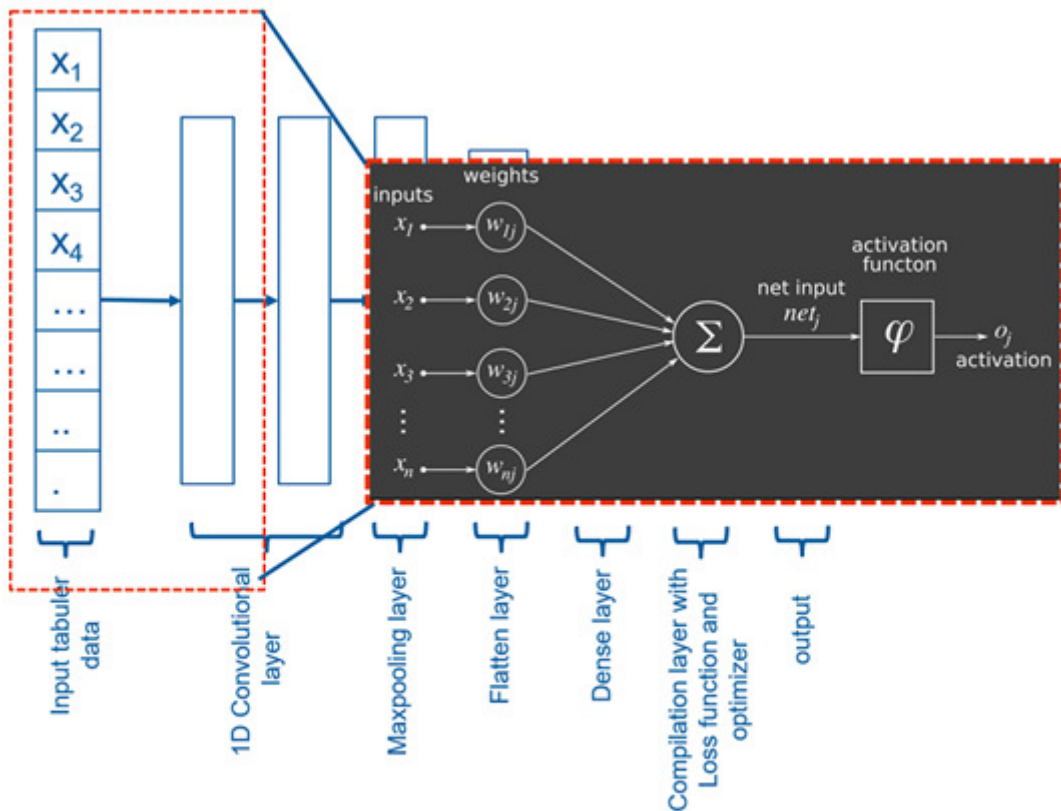
A typical 1D CNN model (see Figure 15a) has a hidden convolutional layer followed by an input layer that operates over 1D sequence data. Based on the length or the complexity of input data, sometimes the first hidden convolutional layer is connected to another hidden convolutional layer, and the next layer could be a pooling layer, e.g., ‘Max-pooling layer’ whose job is to distil the output of the previous layer to the most salient elements. Afterward, a dense layer, which interprets the extracted features by convolutional layers. A flatten layer can also be used between the dense and convolutional layers to reduce the features maps to a single 1D vector (Brownlee, 2020b).

Pooling Layer This layer is used to down-sample the data from one layer to the next layer. In general, two types of pooling are used, i.e., Maximum number pooling and average pooling. When the target is to down-sample a ten numbers array to half of it (i.e., 5). The first step is to choose a size of the window. In this case, it is two, which means if Max-pooling layer is applied, one out of two number having higher weightage will be chosen for the next layer. The main advantage is to speed up the process by reducing the amount of data (Nascimento, 2016, p. 17).

Dropout Layer is an important layer and often used to handle the problem of overfitting. Due to handling a massive set of data, deep neural networks are prone to overfit. Main function is to randomly drop out some units with some probability at the time of training. There is not any fixed rule to choose the percentage but mostly chosen 50%. It works because it is doing an ensemble of several “destroyed” versions of the main network. Thus, it also improves the capabilities of network by reducing the dependence of specific weights in the layer (Nascimento, 2016, pp. 17–18).



(a) Block representation



(b) Convolutional neural network with kernel multiplication

Figure 15 A typical convolutional neural network

Dense Layer is a fully connected layer, which performs the dot product with the previous

layer and the number of neurons or unit specified in the dense layer. A densely connected layer delivers the learning features from all the combinations of the previous layer features, whereas a convolutional layer relies on consistent features with a small repetitive field. For example, if the input shape to dense layer is (batch size,8) and the number of neurons or units in the dense layer is chosen 16, the output shape will be (batch size,16) (Nascimento, 2016, pp. 17–18).

One of the crucial parts of such a model is the **shape of input**. Each sample input should be in terms of **the number of time steps**, which is calculated based on the steps or time required to complete one full cycle by the appliance, i.e., on, operation, and off. It is not a rule to choose it like that, but this way gives better request and the **number of features**, which are considered one or two in this thesis, i.e., Active power and Reactive power. In 1D CNN, the input to the first hidden layer shall be in the shape of **(samples, time-steps, features)** (Brownlee, 2020b). Reshaping and its techniques will be discussed in detail in the next chapter of data pre-processing. The important fact to understand is that CNN does not view the data as human, but the data is treated as a sequence over which the convolutional layer performs read operation, like a 1D image, which is very similar to the 2D CNN model in case of image processing. The kernel slides in one-dimension in 1D CNNs, as shown in Figure 16 (Verma, 2019). Figure shows an example of measurement from an accelerometer, where the first dimension is time-steps, and others are the different measurements by the meter, which can also be referred to as number of features (Brownlee, 2020b).

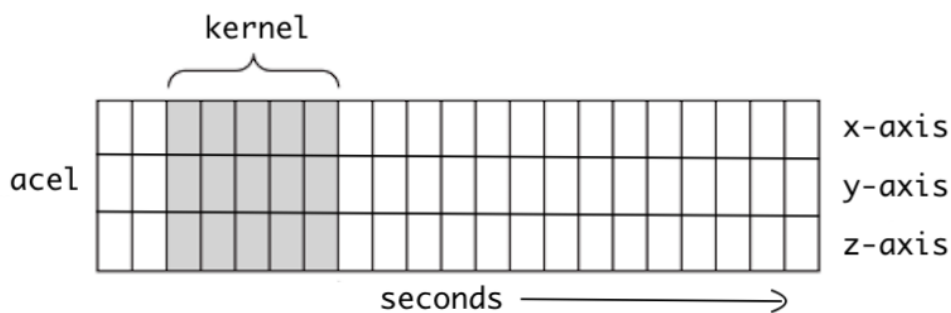


Figure 16 Kernel sliding over 1 D data (Verma, 2019)

2.4 Deep Learning Fundamentals

Deep learning has dramatically impacted image classification, speech recognition, sequence prediction, sentiment classification, image classification, and cybersecurity. Primary hyper-parameters used in deep learning methods play a crucial role, which will be discussed in this section.

2.4.1 Activation Function

An artificial neuron calculates the weighted sum of its input and adds a bias to it. Consider the following equation,

$$Y = \sum(\text{weight} * \text{input}) + \text{bias} \quad (\text{eq. 16})$$

The output Y can be any value ranging from $-\infty$ to $+\infty$. The neuron does not know the bounds of the output value and cannot decide on its own if it should or should not fire. It is where activation functions come into the picture. An activation function decides whether a neuron should be activated or not by calculating the weighted sum and adding bias to it. There are two classes of activation functions, namely:

1. **Linear Activation function:** A linear activation function happens to be a straight-line function where the activation is proportional to the input. Linear activation function has limitation with inability to capture complex relationships. It can be represented as:

$$f(x) = x \quad (\text{eq. 17})$$

2. **Non-Linear Activation functions:** These are the most utilised activation function as they enhance the neural network capability to learn complex and complicated data and generate non-linear mappings from input to output.

(a) Sigmoid: Also known as the logistic Activation Function. It is an S-shaped curve with an output value lies in the range between zero and one. It deals with a problem of vanishing gradient, i.e., there is almost no change in output for a very large and a small values of the input, which leads to no update in weights and thus no further learning of network (Chollet, 2018, pp. 178–233).

(b) Hyperbolic tangent (tanh): Tanh function is also a sigmoidal or S-shaped in nature; it outputs the value in the Range $(-1,1)$. It strongly maps the negative inputs to the negative outputs and maps only zero-valued inputs to near-zero outputs, which improves the training of networks. However, it still has a significant problem with vanishing gradient (Chollet, 2018, pp. 178–233).

(c) Rectified Linear Units (ReLU): It is a simple activation function that returns the value provided as input directly or the value zero if the input is zero or negative, as mentioned in Figure 17(c). ReLU has an advantage over the above-discussed functions that It is faster to compute and converge the network quickly. It also reduces the vanishing gradient problem compared to sigmoid, and tanh function due to the constant derivative (Nascimento, 2016, p. 17). The disadvantage is that when the inputs come to near zero or negative in value, the function gradient becomes zero. Thus, the network cannot perform backpropagation and stop learning, which is identified as the Dying ReLU problem (Chollet, 2018, pp. 178–233).

(d) Leaky ReLU: Leaky ReLU solves the “dying ReLU” problem by allowing the negative slope to be learned. This function provides the slope in negative values part; therefore, the backpropagation is possible. The functionality of Leaky ReLU is not standardized, and it depends on case to case base, which makes it

complicated (Chollet, 2018, pp. 178–233).

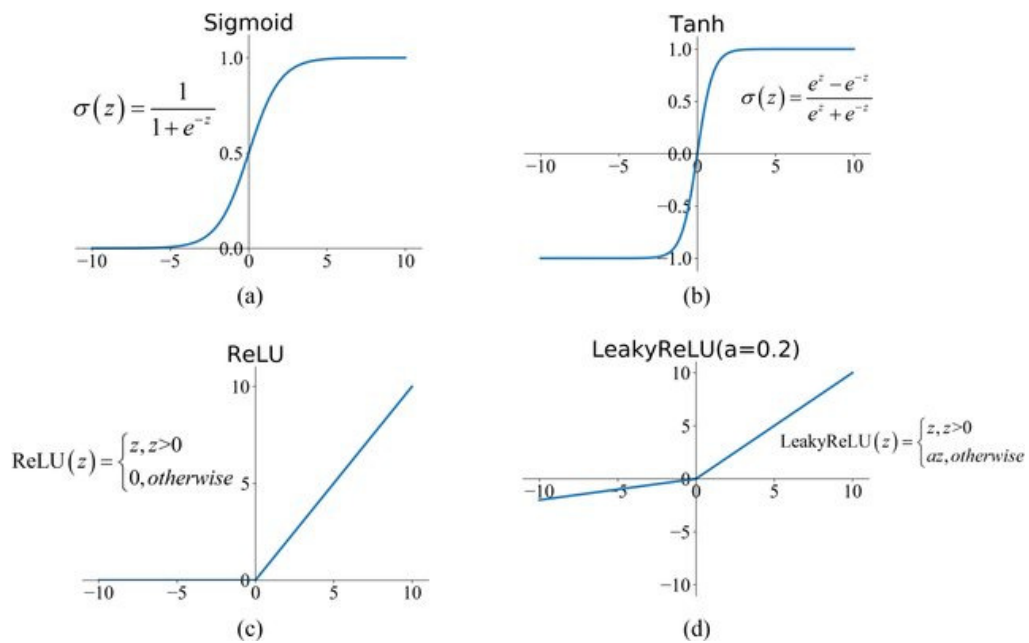


Figure 17 Commonly used activation functions: (a) Sigmoid, (b) Tanh, (c) ReLU, (d) LReLU (Feng *et al.*, 2019, p. 3)

2.4.2 Loss Functions

A deep neural network learns to map a set of inputs to a set of outputs from the training data available to it. These networks are trained using one of the optimizers, whose application is to update the weights. The aim is to move down the slope of an error where the error is a measure of how far the results are from the desired output. This error is calculated using the loss function.

The loss function should act as a metric to evaluate the performance of the model. It should represent all the aspects of the model in a single number such that it can reflect any performance degradation or improvement made by the model (Goodfellow *et al.*, 2016, pp. 271–311). A cost function is an associated term to the loss function, which is the average loss over the entire training set. In contrast, the loss function evaluates the metrics for a single training set. Regression and classification make use of different loss functions. Some of the commonly used loss functions are listed below (Mahendru, 2019).

(a) Squared Error Loss: It is the square of the difference between the actual values and the predicted values from the model. It can be expressed with the equation (Mahendru, 2019),

$$L = (y - f(x))^2 \quad (\text{eq. 18})$$

where,

L is the loss,

y is the actual values,

f(x) is the predicted values.

The corresponding cost function is the **Mean of these squared Errors (MSE)**. It can be expressed as the below-mentioned equation, where the first part is the mean of the variables n with the square error of the predicted and observed values.

$$MSE = \frac{1}{n} \sum_{t=1}^n (y_t - \tilde{y}_t)^2 \quad (\text{eq. 19})$$

(b) Binary Cross Entropy Loss: Binary classification is defined as a predictive algorithm where the output is either one of the two items, indicated by zero or one. For example, if the prediction output is 0.58, which is greater than the halfway mark then the output is one. Else, if the prediction output is 0.45, then the output is zero. This is the most commonly used loss function in binary classification (Mahendru, 2019).

In the presented thesis work, the primary task of the algorithm is to detect the on-off state of appliances. This loss function is implemented in such a way that if the power consumption is below a certain pre-defined value, then the output will be zero or else one. The binary cross-entropy error aims to reduce the entropy of the predicted probability as output from the model (Mahendru, 2019).

2.4.3 Learning Rate

Learning rate defines how much alterations of the weights are necessary with respect to the loss gradient. Selection of the learning rate is quite flexible. It can be set as either a fixed value or modified in each epoch while training a model. **Epoch** defines the number of iterations of the entire training dataset. The number of epochs is selected so that the difference of the losses from one epoch to the next epoch gets minimal or zero, which means that there is no more weight update process occurring and no learning happening in the model. Learning rate regulates the speed of convergence to global minima. If the learning rate is chosen very low (in order of 10^{-3} - 10^{-4}), the convergence to global minima will take longer time, but if the learning rate value is chosen very high (in order of 10^0 - 10^1), there are possibilities of no convergence, as shown in Figure 18. Also, weights get updated slightly in one epoch, when the learning rate is low, which leads to an increase in the required number of epochs for training.

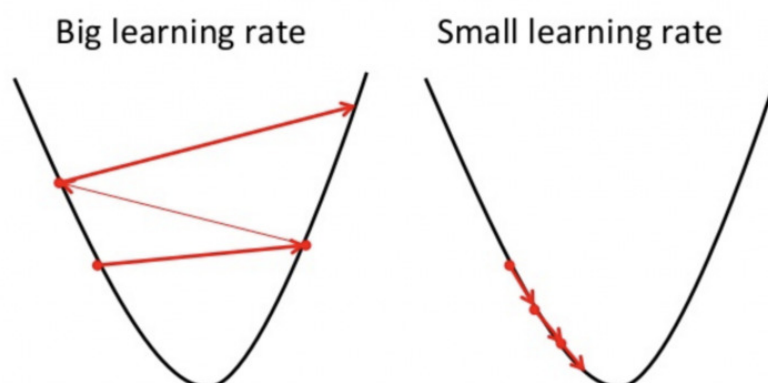


Figure 18 Learning rate (Donges, 2020)

'Global minima' is the minimum point of the entire domain, and 'local minima' is a sub optimal point, as shown in Figure 19. Local minima offer the comparatively minimum point where the losses are minimal but not the least (Khandelwal, 2018). The goal of a model is to train until it reaches the global minima. Optimizers are designed to focus on global minima while weights update in any layer (Khandelwal, 2018).

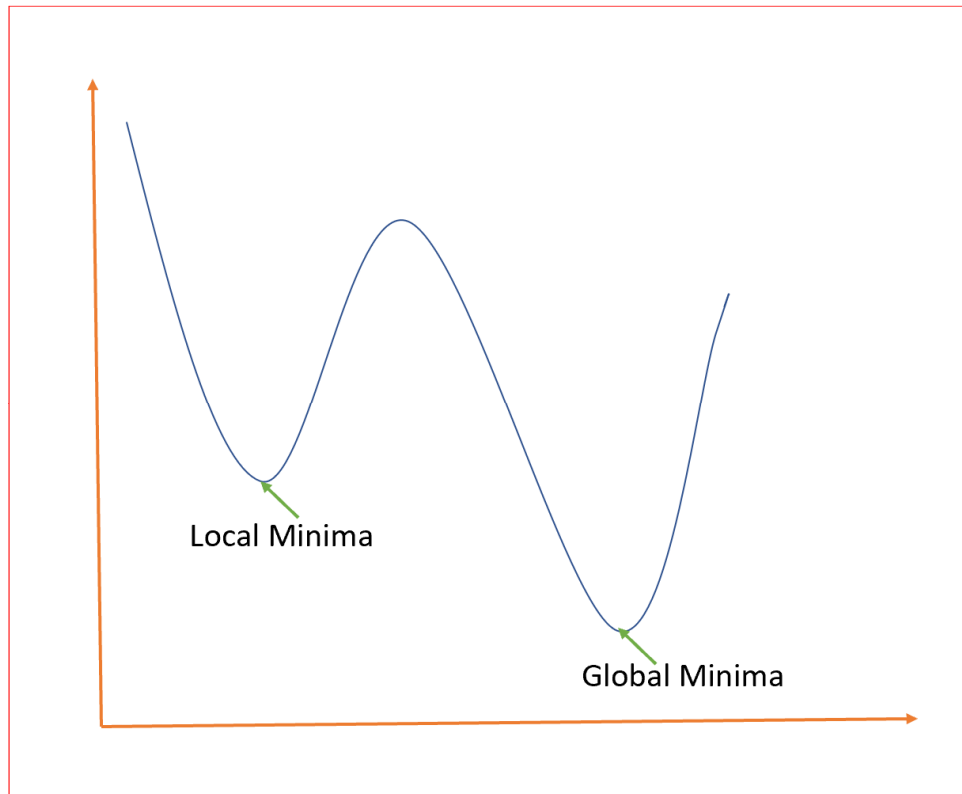


Figure 19 Local minima and Global minima (Khandelwal, 2018)

2.4.4 Optimization

Selection of the optimization algorithm plays a critical role in the convergence of a deep learning algorithm. An optimizer can reduce the losses by changing network attributes such as weights and learning rate. Advantages of the most used optimizers are discussed as follows.

- (a) **Adagrad:** The Adaptive Gradient Algorithm has an adaptive learning rate capability, which allows a network to regulate the learning rate for individual features. **Adagrad** is a very efficient method if the input dataset has lots of missing values. This method has a drawback that the learning rate keeps reducing after every iteration. This problem was taken care of by some optimizers discussed below (Algorithmia, 2018).
- (b) **RMSprop:** It stands for the Root Mean Square Propagation, which is an updated version of Adagrad. It only accumulates gradients in a fixed size window, unlike Adagrad, and deals with the radically diminishing problem of Adagrad.
- (c) **Adam:** Adam is derived from adaptive moment estimation, and it is based on the concept of momentum. The adaptive learning rate of each parameter is calculated from the estimate of the first and the second moments of the gradient. Adam deals

with the radically diminishing problem of the Adagrad and performs efficiently with faster computational power(Algorithmia, 2018).

2.4.5 Parameter Initialization

Weights initialization plays an essential role in the training of a neural network. If all the weights are assigned as zero initially, there will be no learning in the network. Because, even after the gradient update, all the values will be zero. Wrong initialization of weights may lead the network to take more time and computational power to reach global minima. There are some ways for parameter initialization, which are discussed as follows:

(a) Xavier Initialization: It is always good to have the same variance while moving from one layer to another. It supports to keep the signal from exploding to a considerable big value or zero at the end. It is based on Gaussian distribution and represented for a fully-connected layer having m inputs as (Stewart, 2019):

$$W_{ij} \sim N(0, \frac{1}{m}) \quad (\text{eq. 20})$$

where,

W is the weights

N is the neuron in the layer

The value of m is known as fan-in, i.e., the number of incoming neurons or units (Stewart, 2019).

(b) He Normal Initialization: It is a modified Xavier method with a multiplication factor of 2, as shown in the equation below.

$$W_{ij} \sim N(0, \frac{2}{m}) \quad (\text{eq. 21})$$

The initial weights are assigned, keeping in mind the size of the previous layer. It helps to attain the global minimum faster and more efficiently. This method is recommended to use with ReLU layer.

(c) Pre-initialization: It is designed to import the weights of the already-trained network. A model can be saved along with the weights in a particular file format known as **h5py**. Keras provides a feature to load the stored model with predefined weights for further actions. The only limitation is that the datasets used while saving the model must have the same properties as the new dataset required for testing or additional training(Stewart, 2019).

2.5 Evaluation Matrix

F-Measure, also known as F1-score, is a single measurement that contains both recall and precision. A perfect F-measure score would be one; thus, the closer the model score, the better the performance (Brownlee, 2020d). It can be determined as,

$$F - Measure = \frac{2*Precision*Recall}{Precision+Recall} \quad (\text{eq. 22})$$

The **confusion matrix** is one of the most recognised techniques for evaluating results. It is not enough to calculate only classification accuracy of results when a vast dataset is utilised for training a model for several appliances. The confusion matrix calculation provides more inside of the outcomes (Brownlee, 2020d).

$$Classification\ accuracy\ (\%) = \frac{total\ correct\ predictions}{total\ predictions\ made} * 100 \quad (\text{eq. 23})$$

whereas, the confusion matrix has more parameters and can be encapsulated as,

Table 1 Confusion matrix (Bernard, 2018, p. 119)

	Positive Prediction	Negative Prediction
Positive Class	True Positive (TP)	False Positive (FP)
Negative Class	False Negative (FN)	True Negative (TN)

where (Brownlee, 2020d),

true positive: correctly predicted event values.

false positive: incorrectly predicted event values.

true negative: correctly predicted no-event values.

false negative: incorrectly predicted no-event values.

Precision for Binary Classification is one of the metrics required to calculate the F1-score. It can be computed as,

$$Precision = \frac{TP}{TP+FP} \quad (\text{eq. 24})$$

Recall for Binary Classification is another metric that quantifies the number of correct positives made out of all positive predictions and it can be estimated as,

$$Recall = \frac{TP}{TP+FN} \quad (\text{eq. 25})$$

2.6 Save and load model

One of the advantages of a neural network, which is a necessity as well, is that it must train and validate the model only once. It is a tedious procedure that takes a lot of time, computational power, and datasets. TensorFlow provides a feature that once the model starts giving the desired outcomes, it is feasible to save the weights.

This process also helps to create checkpoints during the training. If the availability of computational power is not adequate to execute the training with entire dataset at once, it is possible to breakdown training into several chunks. Once a model is saved, its initial weights assigned will not be random when loading it; rather, the weights will presume their properties. The saving points of a model are known as **checkpoints**. A model is

saved in **HDF5 format**, which only contains the trained weights. It is essential to call all the libraries again, used while training the model, before loading it (Agarwal *et al.*, 2015).

2.7 Development Tools

The primary programming language used for model development in this research is Python, which is the most commonly used programming language in Data science research communities. Python is a **high-level**, interpreted programming language with a strong focus on readability and understandability. Its broad selection of libraries and features that could help streamline the development process and heavily machine learning-oriented toolkit resulted in Python being the primary choice for this task.

Recent advances in the AI field have eased the learning curve significantly, and especially, deep learning has historically been a rather tricky subject to break. Google released its Python-based software library TensorFlow to the public (Agarwal *et al.*, 2015). TensorFlow has several uses, but its primary domain is machine learning by simplifying the development process. Besides, this research employs the usage of Keras, a neural network library that is implemented on top of TensorFlow, streamlining the process even further.

Additional Python packages used in this project are Stats-models, SciKit Learn, Numpy, Pandas, and Matplotlib. SciKit Learn includes several tools for statistical modelling and analysis in machine learning, while others simplify the development.

Table 2 Software information

Software	Version
Python	3.7.0
Microsoft 365	365 ProPlus
Adobe	20.13.20064.405839
Pandas	0.23.4
Numpy	1.16.2
TensorFlow	2.3.0
Keras	2.4.3
Scikit-learn	0.24
matplotlib	2.2.3
Statsmodels	0.12.0

3. Data Collection and Pre-processing

This thesis work is based on real load profile data, not on the simulated database. This chapter provides the details of the project partner, data collection process, and data pre-processing techniques. The typical structure of the whole process is shown in Figure 20.

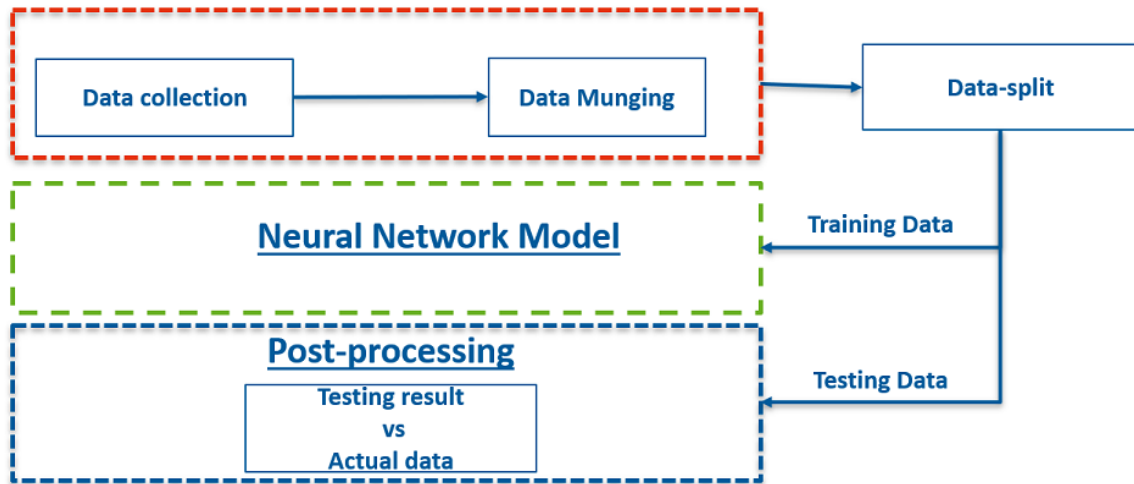


Figure 20 Typical structure of data pre-processing

3.1 Data Collection Process

3.1.1 Background Information of Project Partners

The load profile measurement and data collection process from four dairy farms based in the Bayern state of Germany are executed by the team of **Technische Hochschule Ingolstadt (THI)**, Ingolstadt, Germany, along with their project partner company named **Maschinenringe Germany GmbH (MR)**, Neuburg an-der Donau, Germany. Those four farms are named as Farm1, Farm2, Farm3, and Farm4 in this document. The individual measurements of four electrical appliances (having significant power consumption in a dairy farm) were taken care of by THI. These four appliances are **MK**, **VP**, **MP**, and **SA**. The milking process in all the selected farms occurs twice every day. MK is used for maintaining the temperature of the milk. SA is used for cleaning the udder of cow and pipelines, mainly used twice a day. MP is used to pump the milk to storage. VP is used for the milk suction process. MR provided the aggregated load profiles of the respective farms. MR performed these measurements incorporation with **Lackmann Metergesellschaft mbH & Co. KG**, Münster, Germany.

3.1.2 Measurement Devices

All the measurements performed by THI are measured with a 3-Phase Power Logger named **EMONIO P3**, as shown in Figure 21, manufactured by **Berliner Energieinstitut GmbH, Berlin, Germany**. EMONIO is ISO 50001 and DIN EN 601010-1 certified class 1 (i.e., accuracy ± 1) power logger and capable of measuring phase-wise current, voltage, power factor, active power, and apparent power with time index. The uncertainty for reactive power is less than three percent. The input voltage range is 400 Volts (U) phase-

phase and 240 V Neutral; also, it can withstand up to 80 Ampere (A) alternative current (AC), which are in accordance with the dairy farm appliances.

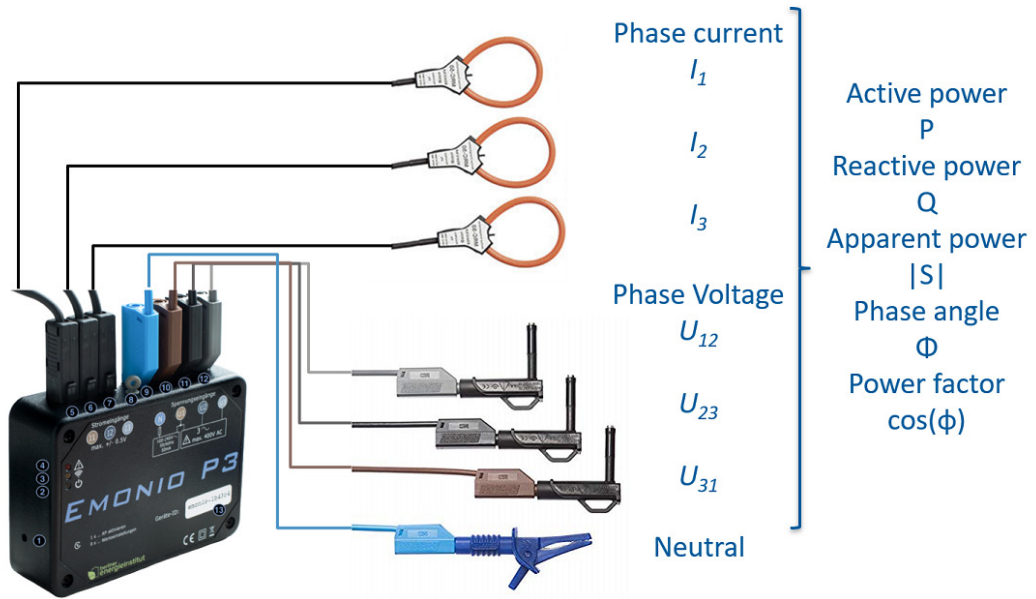


Figure 21 EMONIO P3 (Berliner Energieinstitut GmbH, 2019)

EMONIO has a measurement category of CAT III, which means this device is safe to be used in a control panel or switchboard, as shown in Figure 22.

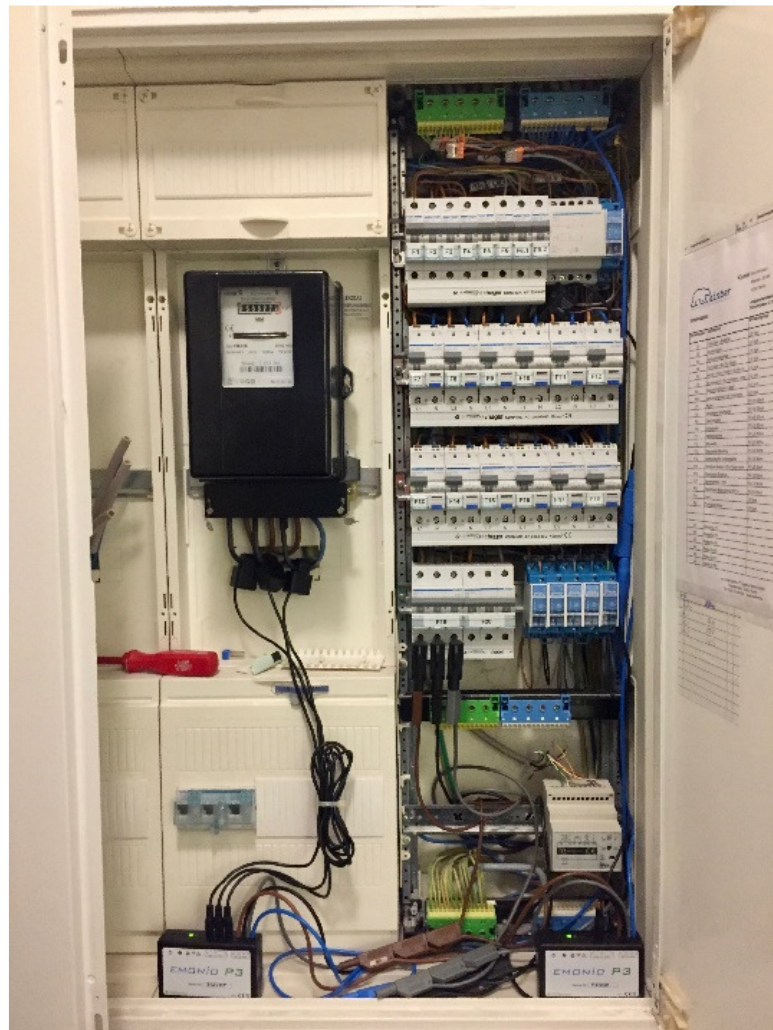


Figure 22 EMONIO Installation in switchboard (Berliner Energieinstitut GmbH, 2019)

EMONIO has communication interfaces as an intern and extern WLAN, MQTT, and Bluetooth, which provided the option to control and monitor it from a remote location (Berliner Energieinstitut GmbH, 2019).

On the other hand, the aggregated load profile provided by MR was measured using a smart meter from Lackmann, as shown in Figure 23. The smart meter from Lackmann is designed for industrial applications and can measure various electric parameters. It also has vast communication support (Lackmann GmbH & Co. KG, 2020). For the presented work, only the aggregated active and reactive power with the time index was acquired.



Figure 23 Lackmann smart meter (Lackmann GmbH & Co. KG, 2020)

3.1.3 Overview of the Data

For the development of NILM algorithm, one or more features were required. Two **features** i.e., active, and reactive power are considered for the developed algorithm. Studies say that the performance of model become better if trained with more features (Li *et al.*, 2019, p. 27). The measurements from EMONIO were captured **one's every second** and from Lackmann meter, **one's every two second**. The frequency of measurement plays an important role in neural network training. It is essential not to miss any load profile variations as input to the model to train it accurately. When the time frame is short (in seconds), the load profile has more probability of capturing all the variations. Measurement frequency also depends on measuring device capability. The measurements from both the devices are delivered in **'csv'** file format, as shown in

Figure 24 (b). Figure 24 (a) shows a sample of available raw data.

```

C:\Users\sinhaa.ESPL_001\Desktop\Projekt_übergabe\Measurements\Meitinger\TH\Scmidt\Vakuumpumpe\Data\Vakuumpumpe 14.12.19.csv - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
Vakuumpumpe 14.12.19.csv
1 timestamp;localtime;vrms;irms;watt;var;va;freq;kw;pf;connected_a;vrms_a;irms_a;watt_a;var_a;va_a;freq_a;kw_a;
2 timestamp;localtime;vrms;irms;watt;var;va;freq;kw;pf;connected_a;vrms_a;irms_a;watt_a;var_a;va_a;freq_a;kw_a;
3 1575375300;3.12.2019 13:15:00;227,4;0,1;24,9;-38,0;26,8;49,9;133,5019;0,55;1;227,3;0,1;12,8;-10,5;12,8;50,0;46,
4 1575376200;3.12.2019 13:30:00;226,9;0,1;24,8;-37,7;27,4;50,0;133,5081;0,52;1;227,1;0,1;13,1;-10,7;12,8;50,0;46,
5 1575377100;3.12.2019 13:45:00;226,4;0,1;24,7;-37,6;27,2;50,0;133,5142;0,53;1;226,5;0,1;13,0;-10,7;12,7;50,0;46,
6 1575378000;3.12.2019 14:00:00;227,7;0,1;25,1;-38,0;28,7;50,0;133,5205;0,51;1;228,0;0,1;13,4;-10,7;13,5;50,0;46,
7 1575378900;3.12.2019 14:15:00;227,2;0,1;25,0;-37,8;27,6;50,0;133,5267;0,52;1;227,3;0,1;13,3;-10,7;13,1;50,0;46,
8 1575379800;3.12.2019 14:30:00;226,8;0,1;24,9;-37,6;27,4;50,0;133,5330;0,52;1;226,8;0,1;13,1;-10,6;13,0;50,0;46,

```

(a)

Name	Änderungsdatum	Typ	Größe
3P310041.csv	20.07.2019 05:52	CSV-Datei	5.948 KB
3P310042.csv	20.07.2019 22:32	CSV-Datei	5.948 KB
3P310043.csv	21.07.2019 15:12	CSV-Datei	5.948 KB
3P310044.csv	22.07.2019 07:52	CSV-Datei	5.948 KB
3P310045.csv	23.07.2019 00:32	CSV-Datei	5.948 KB
3P310046.csv	23.07.2019 17:13	CSV-Datei	5.948 KB
3P310047.csv	24.07.2019 09:53	CSV-Datei	5.948 KB
3P310048.csv	25.07.2019 02:33	CSV-Datei	5.948 KB
3P310049.csv	25.07.2019 10:20	CSV-Datei	2.781 KB

(b)

Figure 24 Exported files from measurement devices

Figure 25 shows a selection of load profiles for aggregated load in blue and individual appliances in red, orange, and green. The y-axis shows the power in kilowatt (kW), and the x-axis is the time axis.

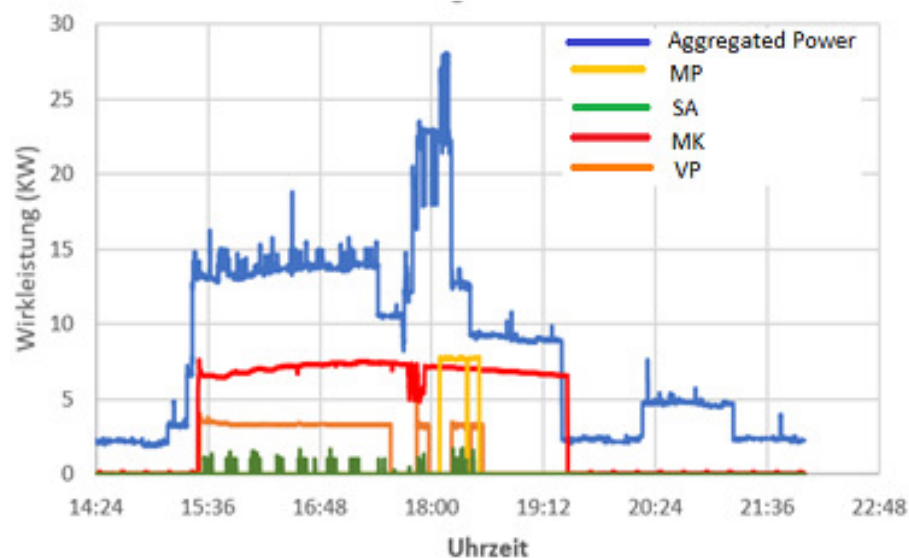


Figure 25 Load profiles (Raw Data Sample)

Initially, it was planned to collect the individual appliance power consumption load profile data from all four farms for a minimum period of seven to ten days, and the aggregated load profile data for the respective farm in same time frame supposed to be received from MR. There were some challenges faced at the time of extracting the data from the measuring devices, which will be discussed in this section. The availability of the overall datasets is listed in Table 3, which concludes that reactive power is only available for the Farm1. A total of three farms can contribute to modelling. Variations in MP and SA datasets are limited to only one farm, and VP datasets are available from two farms. MK has the maximum data availability from three farms for training and testing.

Table 3 Data availability from farms

Farm Name	Date		No. Of Days	VP		MP		MK		SA		Aggregated	
	From	To		P	Q	P	Q	P	Q	P	Q	P	Q
FARM1	24.06.2020	21.07.2020	28	■	■	■	■	■	■	■	■	■	■
FARM2	23.11.2019	03.12.2019	11	■	■	■	■	■	■	■	■	■	■
FARM3	29.05.2019	30.05.2019	2	■	■	■	■	■	■	■	■	■	■
	04.06.2019	07.06.2019	4	■	■	■	■	■	■	■	■	■	■
	08.06.2019	13.06.2019	6	■	■	■	■	■	■	■	■	■	■
FARM4	21.07.2019	23.07.2019	3	■	■	■	■	■	■	■	■	■	■
	26.07.2019	26.07.2019	1	■	■	■	■	■	■	■	■	■	■
	28.07.2019	03.08.2019	7	■	■	■	■	■	■	■	■	■	■
	05.08.2019	13.08.2019	9	■	■	■	■	■	■	■	■	■	■

where,

P : Active power

Q : Reactive power

■ : Available

■ : Not available

3.2 Data Munging

The process of transforming data from a raw data form to another format to make it more appropriate and valuable for the downstream processes is called data munging.

3.2.1 Handling Missing Values

The **first error** was observed with a large percentage of missing values in the raw dataset. The timespan of the missing values was in a wide range, i.e., from a few seconds to several hours, as shown in Figure 26.

Date	Time	V12	Unit	V23	Unit	V31	Unit
14.06.2019	14:37:10	0401.0	ACV	0403.1	ACV	0402.1	ACV
14.06.2019	14:37:12	0400.5	ACV	0402.8	ACV	0402.1	ACV
14.06.2019	14:37:14	0399.7	ACV	0402.0	ACV	0400.9	ACV
14.06.2019	14:37:19	0399.6	ACV	0401.5	ACV	0400.9	ACV
14.06.2019	14:55:42	0399.1	ACV	0401.3	ACV	0400.3	ACV
14.06.2019	14:55:44	0399.4	ACV	0400.9	ACV	0400.5	ACV
14.06.2019	14:55:46	0399.2	ACV	0401.3	ACV	0400.3	ACV
14.06.2019	14:55:48	0399.3	ACV	0400.9	ACV	0399.8	ACV
14.06.2019	14:55:50	0399.3	ACV	0400.9	ACV	0400.4	ACV

(a)

1593891786;4.7.2020	21:43:06;227,0;12,0;1897,
1593891787;4.7.2020	21:43:07;227,1;12,0;1898,
1593891788;4.7.2020	21:43:08;227,1;12,0;1900,
1593891789;4.7.2020	21:43:09;227,1;12,0;1895,
1593891790;4.7.2020	21:43:10;227,0;11,9;1898,
1593891791;4.7.2020	21:43:11;226,9;12,0;1900,
1593920726;5.7.2020	05:45:26;0,0;0,0;0,0;0,0;
1593920727;5.7.2020	05:45:27;231,8;11,9;1833,
1593920728;5.7.2020	05:45:28;231,7;12,4;1855,

(b)

Figure 26 Sample of missing values dataset. (a) Farm1-VP; (b) Farm4-VP

EMONIO delivers data in both on and off condition of the appliance; thus, a possible

reason for the missing values could be power cut to the particular appliance in that timespan. This error was rectified by creating a fresh data-time data-frame for a required number of days and time. Using an API of pandas-library in Python, called **merge**, the new data-frame and raw values data-farm were merged on the basis of the date-time index of the freshly created data-frame. The column with missing values gets filled with **nan**, which were replaced with an average of **nan** and the next value using a Python function called **fill.na**. This process is known as **forward filling**. The second method is to replace the value with a zero value.

The **second error** was observed in the time index, which was not the same for the dataset measured by THI and MR in some cases, as shown in Figure 27.

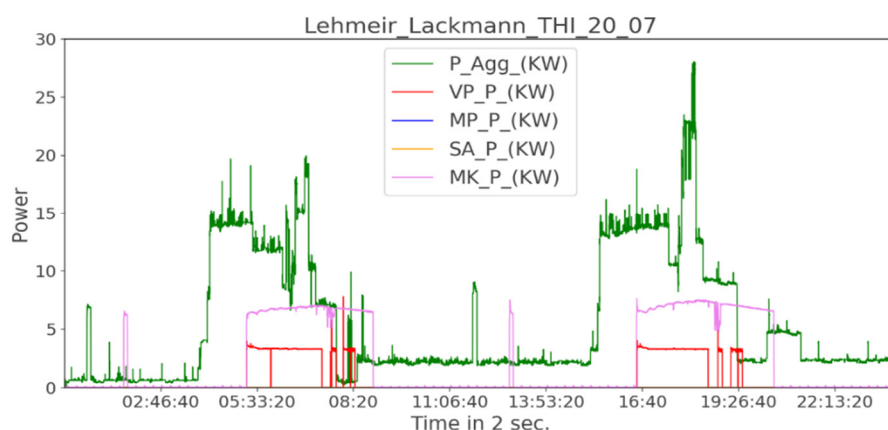


Figure 27 Sample of date-time column shift for both the devices

It is required to give input data of aggregated and individual appliances in the same index. Otherwise, the learning and prediction might also shift by the same time index difference. This error was handled by initially plotting several graphs and extracting the days with this error. Later the adjustments were performed on the time indexes using the 'pandas' and 'NumPy' libraries in python. Also, the EMONIO device was reset before the installation in the next farm.

The **third error** was determined in the dataset measured with EMONIO. Datasets had negative power values for one or more phases. EMONIO is a polarity-sensitive device, and thus, the reason was quite apparent that at the time of installation, the current coil was connected in reverse polarity. This challenge was remedied by multiplying the measured value with a factor of minus one.

The **fourth error** was also detected in the dataset measured with EMONIO. As shown in Table 4, some of the power measurement values were negative, which was identified as the reverse polarity issue of the current coil. Current measurement at the same time found zero. It is also not the self-consumption of the device, which is only 9 W. Thus, the exact reason could not be identified, but the possible cause could be a measurement error. All the values below zero were transformed to zero using the 'pandas' library in python to handle this error.

Table 4 Sample of error in polarity and accuracy

localtime	irms_a	watt_a	vrms_b	irms_b	watt_b	vrms_c	irms_c	watt_c
05.07.2020 06:02	0	-11,8	232	0	-25	232,1	0	0
05.07.2020 06:02	0	-13,2	231,8	0,4	-39,7	232	0	0
05.07.2020 06:02	0	-11,8	231,5	0,9	-129,4	232,5	0	0
05.07.2020 06:02	4,1	769,1	229,5	7,5	-1316,2	230,7	6,1	-820,6
05.07.2020 06:02	3,9	745,6	229,6	7	-1383,8	230,9	5,7	-857,4
05.07.2020 06:02	3,8	711,8	230,1	5,9	-1101,5	231,2	4,4	-727,9
05.07.2020 06:02	3,7	686,8	230,1	5,8	-1014,7	231,2	4,4	-669,1

3.2.2 Data Transforms

Data transformation from raw data to a model readable data format plays a significant role in any deep neural network modelling. All the techniques, processes, and steps taken will be discussed briefly.

Scaling of numerical inputs to a standard range enhances the performance of algorithms. Two most popular techniques for scaling numerical data prior to modelling are **Standardization** and **Normalization** (Brownlee, 2020c).

Data Normalization scales every input variable into a range of zero to one. Neural networks training are data sensitive, and their outcomes improve when trained over small weights. If a model is trained over large value datasets, model weights in hidden layers become large, which make the learning complicated (Brownlee, 2020c). Thus, the 'MinMaxScaler' operator of python based 'sklearn' library was applied to the aggregated load profile and individual appliances load profiles for normalization. This operator divides the whole dataset by its maximum value and transforms the dataset into a range of zero to one.

Transformation of datasets, from normalised power values to the **State of appliance** (i.e.,0/1 for on/off, respectively). The transformation parameters are defined in Table 5.

Table 5 Parameters for transformation to state values

Name of device	Chosen parameter for on/off
VP	if VP_ normalised power <0.02 then '0' else '1'
MP	if MP_ normalised power <0.5 then '0' else '1'
MK	if MK_ normalised power <0.07 then '0' else '1'
SA	if SA_ normalised power <0.02 then '0' else '1'

Train-Test Split is a procedure to estimate the performance of a deep learning algorithm. A neural network needs data in train and test datasets. This procedure is appropriate when dealing with a very large dataset. It helps to evaluate the performance of a model in a short period.

The percentage of split is not fixed, but the common split percentages include train: 80%, test: 20%, train: 67%, test: 33%, or train: 50%, test: 50%. The training dataset is a composition of '**X_train**' and '**Y_train**'. X_train can be defined as the data which is input to a model (e.g., aggregated power), and Y_train can be defined as the desired output from a model (e.g., Individual appliance power, or on-off state of the appliance). Similarly,

the testing dataset is also a composition of 'X_test' and 'Y_test'. X_test can be defined as the dataset, which is provided to a model as input for testing (e.g., aggregated power), and Y_test can be defined as the outcome of a model (e.g., on-off state of the appliance).

Validation Dataset is a dataset that is used to describe the evaluation of any model when tuning hyper-parameters. Train dataset can be divided into train and validation dataset, or validation dataset can be prepared separately from raw data, which is not used for training or testing. Likewise, Train-Test Split, validation data set split percentage is also not fixed. The common split percentages include train: 85%, Validation: 15%, train: 75%, Validation: 25%.

Reshaping is a very crucial procedure when dealing with 1D-dataset as discussed in 2.3.10; the model expects the input to be 3D with [sample, timesteps, features].

For this thesis work, **features** are considered as either only active power or both active and reactive power; thus, features are either one or two, respectively. The next parameter is **timesteps** also known as **window-size**, which can be defined as the number of measurements performed for any particular dataset type. If aggregated power is measured one's every two seconds for 24 hours. Then, there will be a total of 43,200 data points. This number is the timesteps for this dataset. There is no rule for the selection of a number of time steps, but selection criteria should be in such a way that at least one complete cycle (i.e., on-off) of the appliance is covered. Timesteps were preferred differently for all four appliances based on their operation cycles. The last parameter in reshaping is a **sample** also known as **batch-size**, defined as the number of sets prepared from the 43,200 data points for training. Each sample will have a defined number of timesteps. For example, 10,000 samples can be prepared, each having 10 timesteps. The variables of a sample are chosen in such a way that either they are 'unique' in each sample or 'partially repeats' in each sample (Brownlee, 2020b). Table 6 shows three cases of samples made from a univariant dataset [10,20,30,40,50,60,70,80,90] having one feature. The timesteps is considered as 3 for reshaping all the cases. Case 1 and Case 3 are partially repeating the values whereas Case 2 is having unique values.

Table 6 Example of sample datasets

Case1:	Case2:	Case3:
[10,20,30]	[10,20,30]	[10,20,30]
[20,30,40]	[40,50,60]	[30,40,50]
[30,40,50]	[70,80,90]	[50,60,70]
[40,50,60]		[70,80,90]
[60,70,80]		
[70,80,90]		

4. Model Development

This thesis demands to develop an algorithm for the classification of the on-off status of the selected appliances. Thus, the sequence-to-sequence approach was adopted for this work. The advantage of the sequence-to-sequence approach is that the size and shape of the outcome will remain the same as the input dataset. If the model is tested on one full day of operations, the classification results will be available for the whole day. Several models using CNN, LSTM, and their combinations were trained and tested over the same datasets. The architecture of models providing satisfactory results based on their F-1 scores was examined. The availability of datasets is shown in Table 3. It was observed that training of a neural network mainly requires variations in data instead of the data of a similar type. Thus, complete datasets were not used for training the models to avoid overfitting issues. Due to the unavailability of reactive power for the appliance MP and MK, they were trained and tested with one-feature (i.e., active power). Nevertheless, SA and VP were trained and tested for both one-feature (i.e., active power) as well as two-features (i.e., active and reactive power).

The following datasets were chosen for training purpose as shown in Table 7.

Table 7 Selection of datasets for training

Appliance Name	Farm1	Farm2	Farm4
MK	24.06.20 to 28.06.20	23.11.19 and 27.11.19	21.07.19 to 22.07.19
VP	24.06.20 to 28.06.20	-	21.07.19 to 22.07.19
SA	24.06.20 to 28.06.20	-	-
MP	-	-	05.08.19 to 10.08.19

The following datasets were selected for testing purpose as shown in Table 8.

Table 8 Selection of datasets for testing

Appliance Name	Farm1	Farm2	Farm4
MK	29.06.20 to 30.06.20	28.11.19 to 29.11.19	23.07.19
VP	29.06.20 to 30.06.20	-	23.07.19
SA	29.06.20 to 30.06.20	-	-
MP	-	-	11.08.19 to 12.08.19

4.1 Model-1: 1D-CNN- BDRNN

The first model is a one-dimensional convolutional bi-directional neural network with eleven layers, as shown in Figure 28. The first two hidden layers are the 1D convolutional layer having sixty-four neurons, each with a filter size of three and five, respectively.

Sixty-four neurons fitted well to handle the stated amount of data and eliminating the overfitting issues. A filter of 3*3 and 5*5, respectively, slides over the dataset to extract the unique characteristics in the first two layers. The third hidden layer is max-pooling with a filter size of two, which reduces the data-size to half by selecting one out of every two values, whichever delivers more significant weight. The fourth hidden layer is bidirectional GRU with thirty-two neurons, which supports the loss function by reducing the losses to get closer to the desired results, and the fifth hidden layer is the dropout-layer with a 50% drop. The next two hidden layers are again bidirectional GRU and dropout. The second last hidden layer is a fully connected dense layer with one-hundred neurons, which collects all the information from the previously hidden layer and puts it into neurons. The last hidden layer is a dense layer with one node, which will give only one value at a time as required in the form of either zero or one.

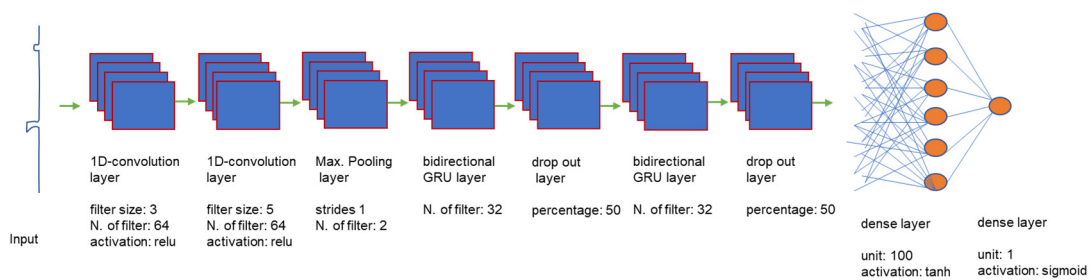


Figure 28 Architecture of Model-1 (1D-CNN-bidirectional RNN)

The training parameters nominated for Model-1 are defined in Table 9 for all four appliances.

Table 9 Training parameters of Model-1

Appliance Name	MK	MP	VP	SA
Feature	Active power			
Data Frequency	one's in 30 s	one's in 2 s	one's in 60 s	one's in 60 s
Trainable Parameters	64,841			
Samples	181	5,670	218	79
Batch-size	128	32	32	32
Window-Size	1,500	10	150	60
Number of epochs	20	10	20	5
Loss function	Binary_crossentropy			
Optimizer	Adam			
Activation function	sigmoid			
Learning rate	Initialized with 10^{-1} and reduced until 10^{-3} during epochs			

Active power was used for the training of the Model-1. The selection of data-frequency

was performed on the bases of the switching cycle of each appliance. This model creates a total of sixty-four thousand eight hundred forty-one trainable parameters based on the assortment of hidden layers. The window size chosen for each appliance is adequate for at least one full operation (on and off) cycle, respectively. The hyper-parameters play extremely critical roles, and the outcomes might get affected by slight changes. ‘Binary_crossentropy’ loss function and ‘Adam’ optimizer provided pertinent results, and they are best suitable for classification problems. It is also recommended to implement the ‘sigmoid’ activation function in the last hidden layer if the ‘Binary_crossentropy’ loss function is applied. The learning rate was initialized with 0.1 for fast learning and later reduced until 10^{-3} during epochs to avoid overshoot the global minima.

4.2 Model-2: 1D-CNN-LSTM

Model-2 is a combination of 1D-CNN and RNN consisting of fourteen layers, as shown in Figure 29. The first two hidden layers are the 1D convolutional layer, each accommodating sixty-four neurons with a filter size of three and five, respectively. The third hidden layer is an LSTM layer consisting of sixty-four neurons, which is implemented here to understand the pattern of features extracted through the convolutional layers. The fourth hidden layer is the batch-normalization layer, which automatically standardises the inputs. The fifth hidden layer is ‘LeakyReLU’, which works as an activation function for the LSTM layer, followed by the sixth hidden layer as a dropout with a 50% drop. Another set of an LSTM, batch-normalization, ‘LeakyReLU’, and dropout layers, is applied. The second last hidden layer is a fully connected dense layer with one-thousand twenty-four neurons. A large number of neurons is chosen here to minimise the losses. The final hidden layer is a dense layer with one node, which will give only one value at a time as required in the form of either zero or one.

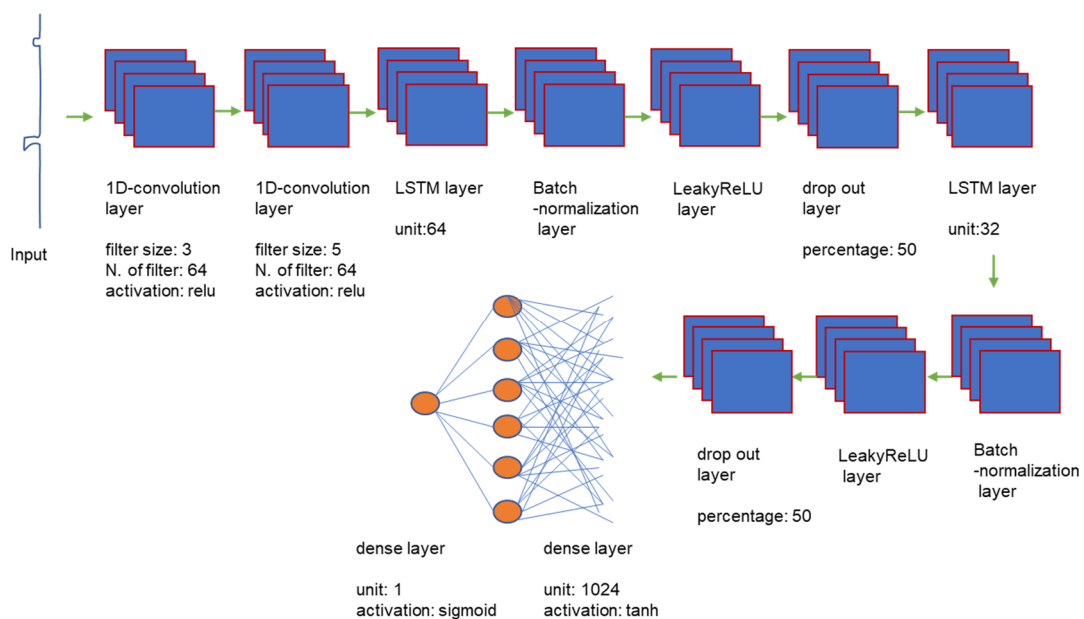


Figure 29 Architecture of Model-2 (1D-CNN_LSTM)

The training parameters selected for Model-2 are defined in Table 10.

Table 10 Training parameters of Model-2

Appliance Name	MK	MP	VP
Feature	Active power		
Data Frequency	one's in 10 s	one's in 2 s	one's in 120 s
Trainable Parameters	101,249		
Samples	559	4,725	109
Batch-size	32		
Window-Size	1,500	10	90
Number of epochs	20	10	5
Loss function	Binary_crossentropy		
Optimizer	Adam		
Activation function	sigmoid		
Learning rate	Initialized with 10^{-1} and reduced until 10^{-2} during epochs.	10^{-3}	

Model-2 was trained and tested for three appliances with active power. Data-frequencies were modified in comparison to Model-1 to make them more suitable for Model-2. The learning rate was kept constant for MP and VP.

4.3 Model-3: LSTM

Model-3 is an RNN-LSTM model, which gives good results in time series and sequential problems. The first hidden layer is an LSTM layer comprising one hundred twenty-eight neurons, followed by the batch-normalization layer, activation layer as 'LeakyReLU' and dropout layer with a 50% drop. Another set of an LSTM layer consisting of sixty-four neurons, batch-normalization, 'LeakyReLU', and dropout layers with a 50% drop, is implemented. The last two fully connected hidden layers are applied with one-thousand twenty-four and one neurons, respectively, to step down the weights update and deliver the result in either zero or one.

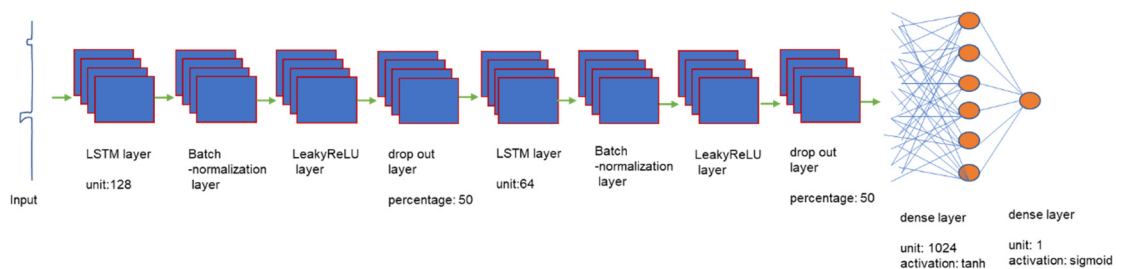


Figure 30 Architecture of Model-3 (LSTM)

The training parameters applied for Model-3 are defined in Table 11.

Table 11 Training parameters of Model-3

Appliance Name	MK	MP	VP
Feature	Active power		
Data Frequency	one's in 60 s	one's in 2 s	one's in 120 s
Trainable Parameters	184,321		
Samples	92	4,725	109
Batch-size	128	32	
Window-Size	500	10	90
Number of epochs	40	5	10
Loss function	Binary_crossentropy		
Optimizer	Adam		
Activation function	sigmoid		
Learning rate	Initialized with 10^{-1} and reduced until 10^{-3} during epochs.		

Model-3 was trained and tested for three appliances utilizing active power. The other stated parameters were modified to improve the performance of the model.

4.4 Model-4: Multi-feature 1D-CNN-BDRNN

The architect of Model-4 is comparable to Model-1: 1D-CNN- BDRNN, as shown in Figure 31. The only modification is in the input layer of Model-4, which comprises two features, i.e., active power and reactive power.

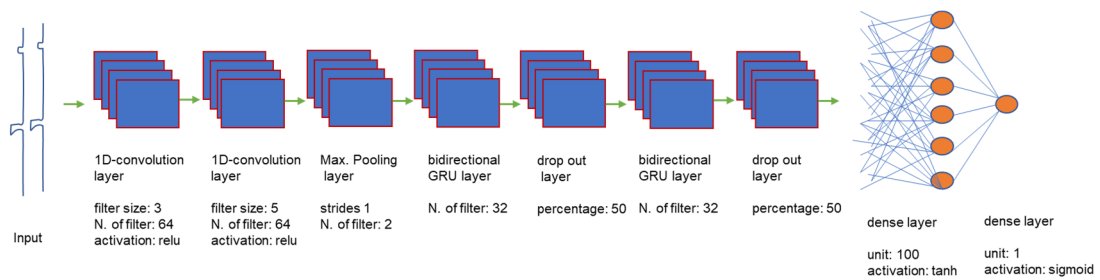


Figure 31 Architecture of Model-4 (Multi-Feature 1D-CNN-bidirectional RNN)

The training parameters selected for Model-4 are defined in Table 12.

Table 12 Training parameters of Model-4

Appliance Name	VP	SA
Feature	Active and Reactive power	
Data Frequency	one's in 60 s	one's in 120 s
Trainable Parameters	49,897	
Samples	155	77

Batch-size	32	32
Window-Size	150	90
Number of epochs	3	10
Loss function	Binary_crossentropy	
Optimizer	Adam	
Activation function	sigmoid	
Learning rate	10^{-3}	

Model-4 was trained and tested for two appliances utilizing active and reactive power. The other parameters stated above in the Table 12 were modified to achieve the best result.

4.5 Model-5: Multi-feature 1D-CNN-LSTM

The architect of Model-5 is similar to Model-2: 1D-CNN-LSTM, as shown in Figure 32. This model comprises two features as input, i.e., active power and reactive power.

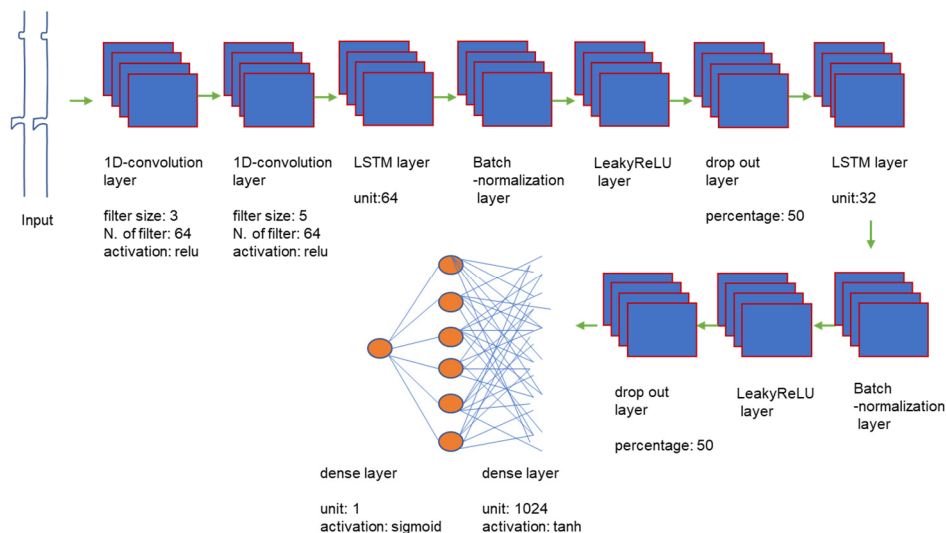


Figure 32 Architecture of Model-5 (Multi-Feature 1D-CNN_LSTM)

The training parameters carefully chosen for Model-5 are defined in Table 13.

Table 13 Training parameters of Model-5

Appliance Name	VP	SA
Feature	Active and Reactive power	
Data Frequency	one's in 60 s	
Trainable Parameters	70,217	
Samples	155	
Batch-size	32	

Window-Size	150	
Number of epochs	10	5
Loss function	Binary_crossentropy	
Optimizer	Adam	
Activation function	sigmoid	
Learning rate	10^{-3}	

Model-5 was trained and tested for two appliances with active and reactive power. It was observed, same parameters worked for both the appliances to achieve desired results.

4.6 Model-6: Multi-feature LSTM

The architect of model-6 is analogous to Model-3: LSTM, as shown in Figure 33. This model is trained with two features as input, i.e., active power and reactive power.

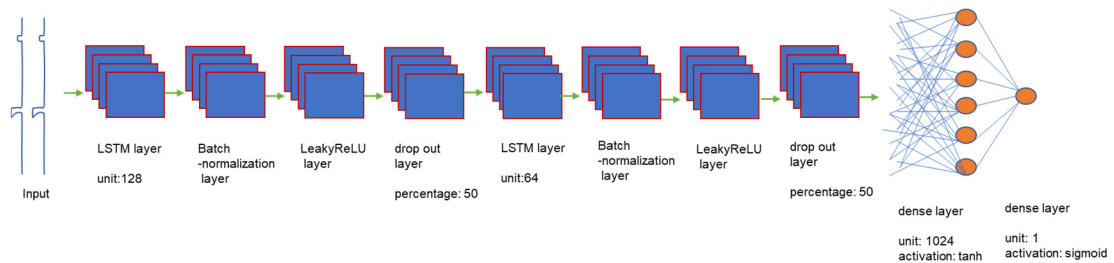


Figure 33 Architecture of Model-6 (Multi-Feature-LSTM)

The training and testing parameters selected for Model-6 are defined in Table 14.

Table 14 Training parameters of Model-6

Appliance Name	VP	SA
Feature	Active and Reactive power	
Data Frequency	one's in 60 s	one's in 120 s
Trainable Parameters	123,849	70,217
Samples	155	78
Batch-size	32	32
Window-Size	150	60
Number of epochs	5	10
Loss function	Binary_crossentropy	
Optimizer	Adam	
Activation function	sigmoid	
Learning rate	10^{-3}	

5. Results and Comparison

The performance analysis of the six models discussed in chapter 4 is presented in this chapter. The comparative evaluation of the models is performed based on their F1 score, defined in chapter 2.5. Outcomes of models are either one or zero, which indicates the state of appliance, on or off respectively. All models are tested on a full-day load profile, and their results are plotted separately for each day. The outcome of models is plotted with either normalised power of appliance or actual state (on-off) of the appliance. The data frequency parameters were chosen differently for each appliance for all six models, that caused a difference in time-indexing of the normalised value of power in plotted graphs of results.

The outcomes of the three models tested for MK on five days of datasets are shown in Figure 34. Every row represents the selected day from a farm as mentioned vertically with 'Farm_date'. It is observed that the LSTM model outperformed for a few days, but the performance is not consistent. In other two models, the convolutional layers performed well in extracting the power variation features, and bi-directional layers helped to minimise the difference between the actual state and outcome of the model. The result of the 1D-CNN-LSTM and 1D-CNN-BDRNN models are nearly identical to normalised power. The performance of 1D-CNN-BDRNN model is consistent and best out of these three models.

Note: In the following graphs, the **X-axis** is representing the time of a full day starting from 00:00:01 (HH: MM: SS), and the **Y-axis** is representing the state (on-off as one-zero) for results and the normalised power value of an appliance between zero and one. The actual state of appliances is denoted with a key as Actual _device name (MK, VP, MP or SA) _N (normalised)_AP (active power). The result of each model is displayed with orange colour, and the normalised power of the appliance is shown in purple colour.

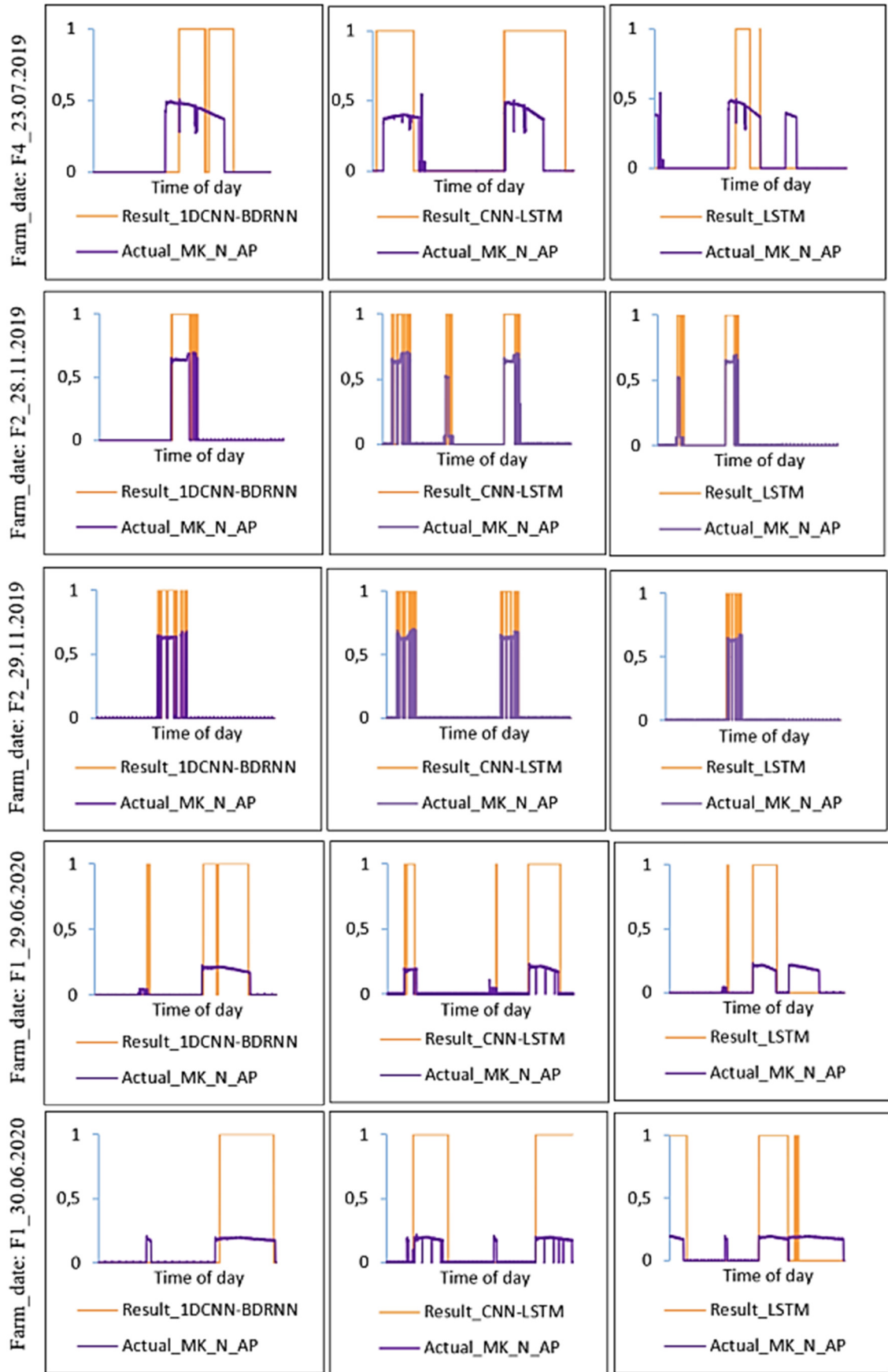


Figure 34 Comparison of result for three models tested for MK on different dates and farms

where,

Actual_MK_N_AP: ground truth of MK (normalised active power).

Result_1DCNN-BDRNN: result from Model-1.

Result_CNN_LSTM: result from Model-2.

Results_LSTM: result from Model-3.

It was observed that the performances of models were varying for the different farms. The possible reasons identified behind this behaviour were the difference in operational time and power consumption of MK for all the three farms, and limitation of variations in the dataset available for training. The graphs displayed in Figure 34 cannot quantify the results; this was achieved with an F1 score graph, as shown in Figure 35. The F1 score for the models 1D-CNN-BDRNN and 1D-CNN-LSTM are very close to one, which is the maximum possible score. The score of an LSTM model is not consistent for all the five days and relatively lower than other two stated models. 1D-CNN-LSTM model outperformed based on the F1 score and consistency in results for five days of testing.

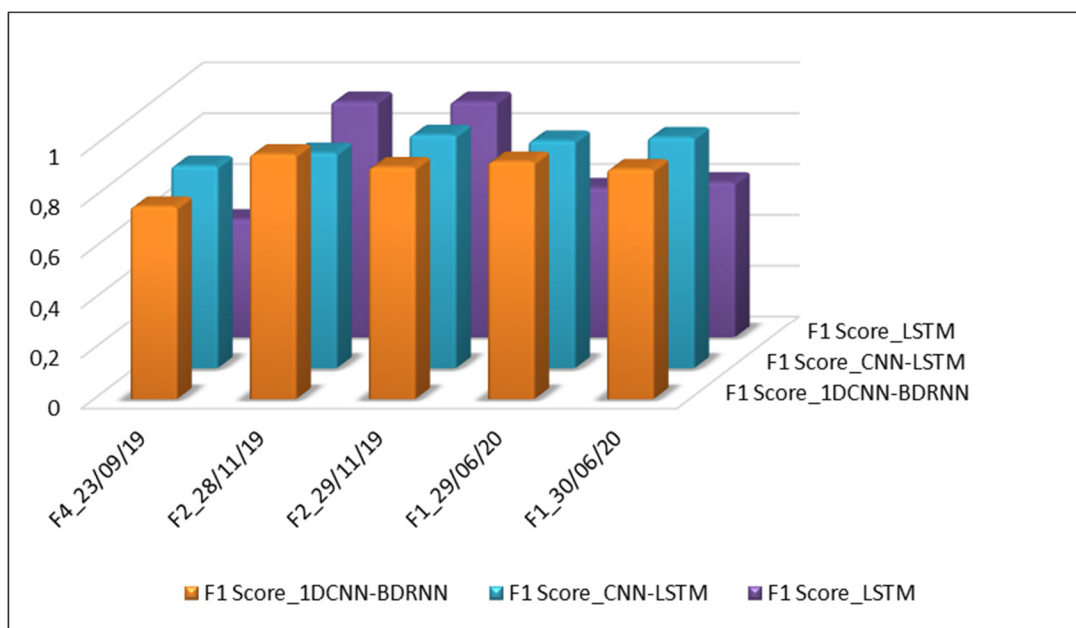


Figure 35 F1-Score comparison of three model tested for MK on different dates and farms

where,

F1, F2, and F4 are Farm1, Farm2, and Farm4 respectively.

F1 Score_1DCNN-BDRNN: result from Model-1.

F1 Score_CNN_LSTM: result from Model-2.

F1 Score_LSTM: result from Model-3.

The results of the three models tested for MP on two days of the dataset are shown in Figure 36. It was observed that the performances of all the models were virtually identical to the normalised values of power consumption and the consistent for both the days. The possible reasons identified behind this behaviour were the limitation of variations in the dataset available to only one farm. The convolution layer and LSTM layer are capable of extracting and memorizing the features and their patterns.

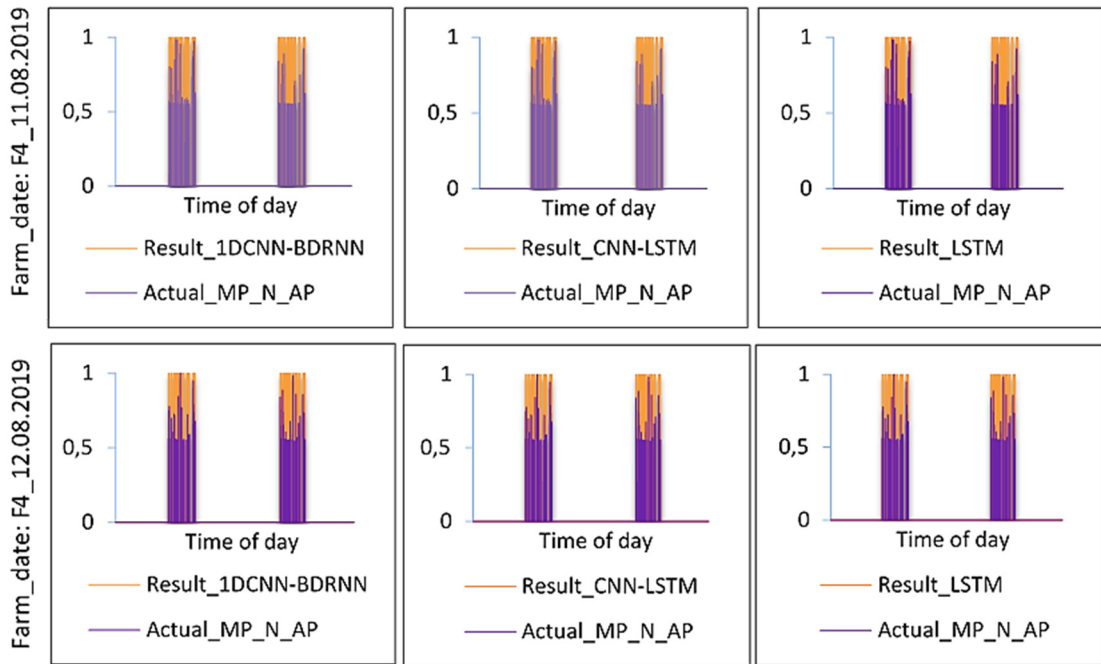


Figure 36 Comparison of the results of three models tested for MP on different dates

where,

Actual_MP_N_AP: ground truth of MP (normalised active power).

Result_1DCNN-BDRNN: result from Model-1.

Result_CNN_LSTM: result from Model-2.

Results_LSTM: result from Model-3.

Figure 37 shows that the F1 scores for all the models are one, which is only an ideal case. Therefore, the models should be tested on the dataset of different farms to identify their performances further and make them more robust.

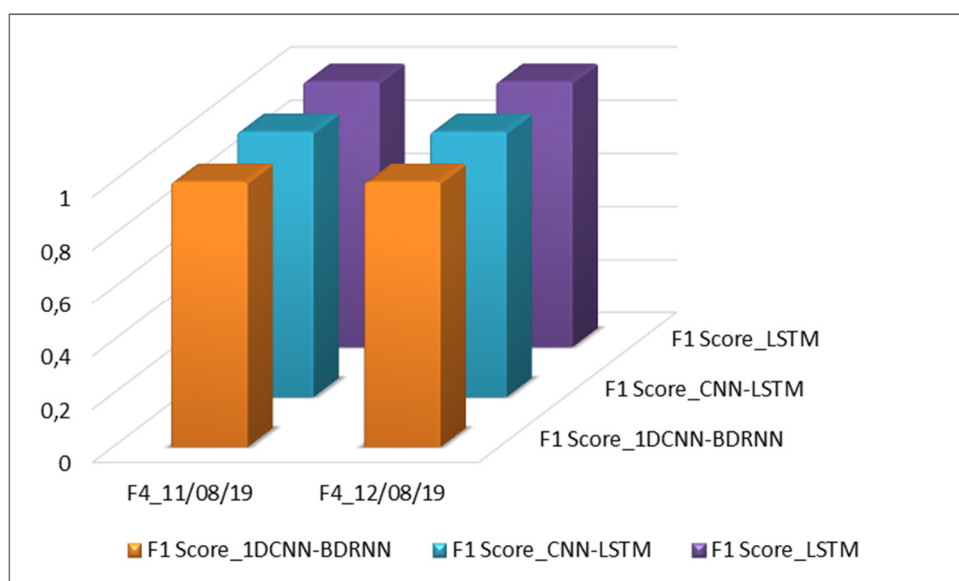


Figure 37 F1-Score comparison of three model for MP on different dates

where,

F4 is the Farm4.

F1 Score_1DCNN-BDRNN: result from Model-1.

F1 Score _CNN_LSTM: result from Model-2.

F1 Score _LSTM: result from Model-3.

The outcomes of models tested for VP on three days dataset from two farms are shown in Figure 38. It was observed that the operation cycle of VP reflects several variations throughout the day, and the load profiles are also entirely different for both the farms. The outcomes of all the three models are identical to the power consumptions for Farm4, which indicates their excellent performance. On the other hand, the convolutional layer-based models are not capable of identifying the 'on-state' of VP for Farm1 at some points. Which shows the limitation of a convolutional layer in extracting the features in small windows. Among the three models, LSTM based model outperformed as the LSTM layer works to learn the pattern from datasets.

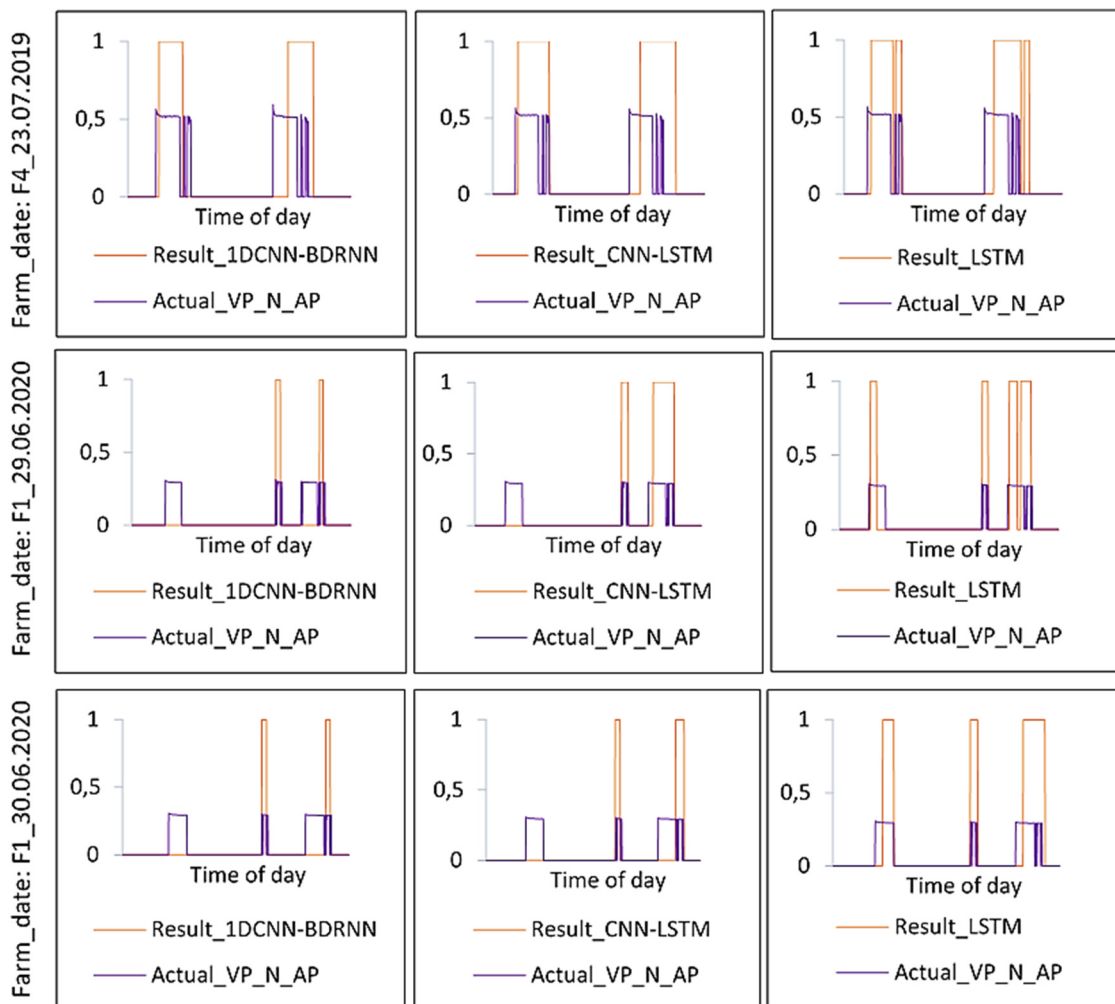


Figure 38 Comparison of results of the three models tested for VP of different dates and farms

where,

Actual_VP_N_AP: ground truth of VP (normalised active power).

Result_1DCNN-BDRNN: result from Model-1.

Result_CNN_LSTM: result from Model-2.

Results_LSTM: result from Model-3.

Figure 39 shows that the F1 scores for 1DCNN-BDRNN and CNN_LSTM are not consistent for three days. Thus, these models are not suitable for VP. The LSTM model outperformed with a score of about 0.7 for all three days as well as it shows consistency. Therefore, it is recommended to opt for the LSTM model for VP.

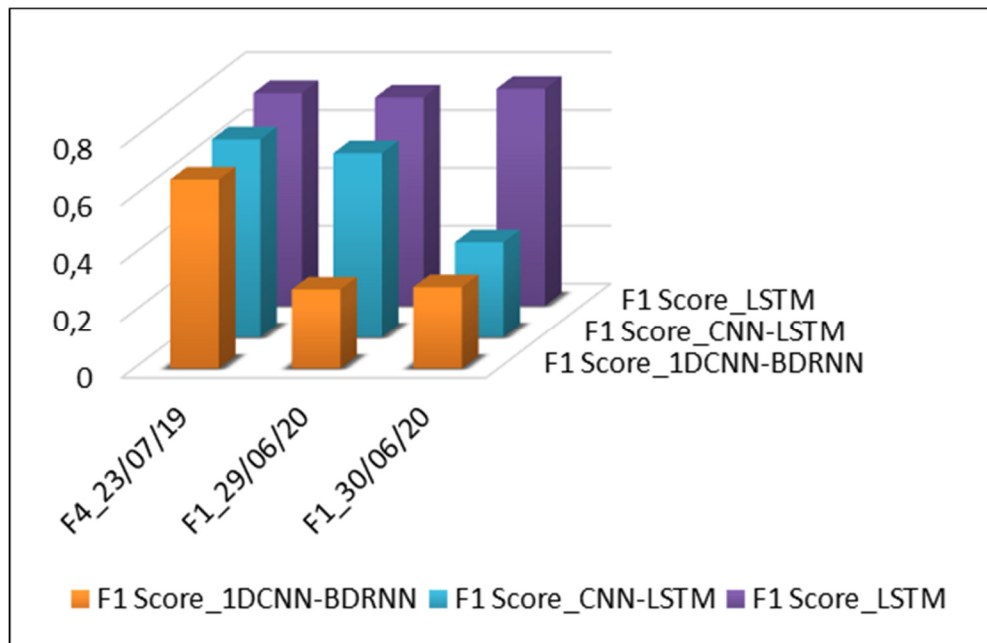


Figure 39 F1-Score comparison of three model for VP tested on different dates and farms

where,

F1, and F4 are Farm1, and Farm4 respectively.

F1 Score_1DCNN-BDRNN: result from Model-1.

F1 Score_CNN_LSTM: result from Model-2.

F1 Score_LSTM: result from Model-3.

The outcomes of the 1DCNN-BDRNN model tested for SA on a two days dataset for one farm is shown in Figure 40. It was observed that SA operates only twice a day for a short time of around thirty minutes. The availability of dataset was limited to only one farm; consequently, the LSTM based models could not identify any useful sequence in dataset. The convolutional layer successfully extracted the feature, and the result of 1DCN-BDRNN are indistinguishable to the normalised power, as shown in Figure 40.

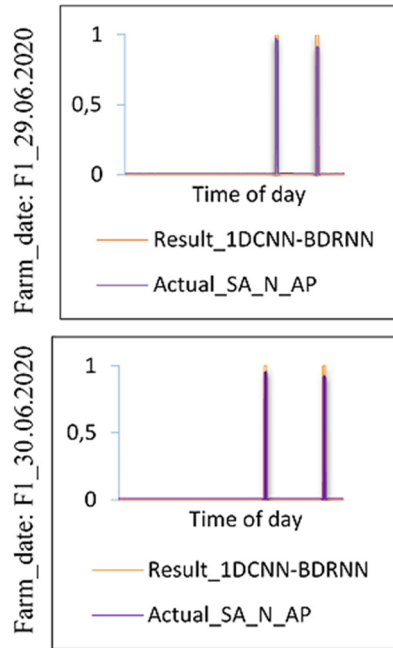


Figure 40 Comparison of results of three model for SA tested on different dates

where,

Actual_SA_N_AP: ground truth of SA (normalised active power).

Result_1DCNN-BDRNN: result from Model-1.

Result_CNN_LSTM: result from Model-2.

Results_LSTM: result from Model-3.

Figure 41 shows, the F1 scores of the 1D-CNN-BDRNN model for both the days are above 0.5 but consistent. Hence, this model needs to be trained and tested with multiple farms dataset to ensure the credibility of the performance.

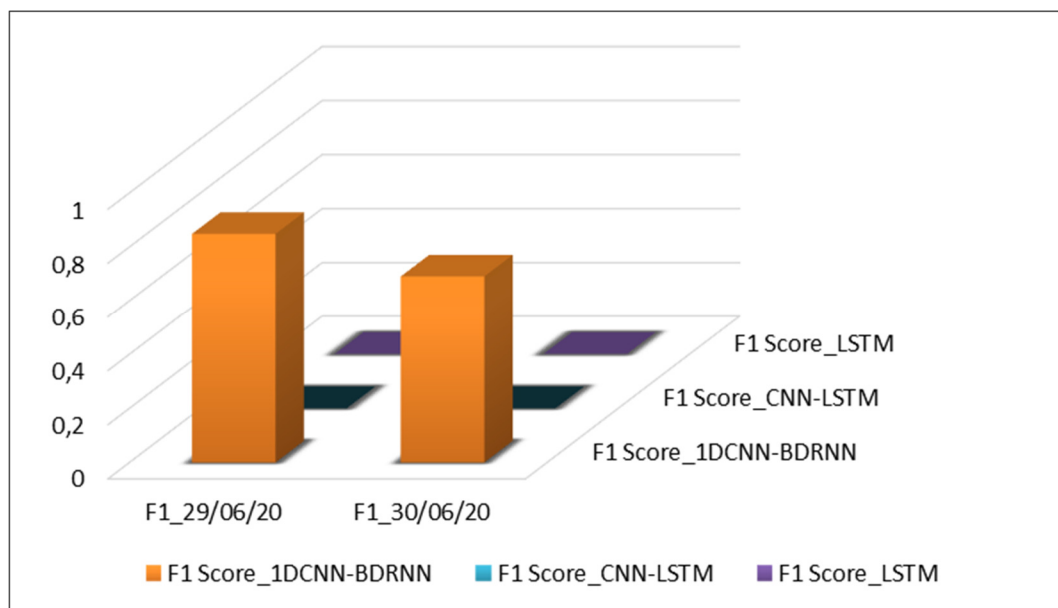


Figure 41 F1-Score comparison of three model for SA tested on different dates

where,

F1 is the Farm1.

F1 Score_1DCNN-BDRNN: result from Model-1.

F1 Score _CNN_LSTM: result from Model-2.

F1 Score _LSTM: result from Model-3.

The results discussed so far were based on only one feature, and the results of models trained and tested with two features (i.e., active power and reactive power) are discussed later in this chapter. Graphs were plotted between the **on-off state on X-axis** and **time on Y-axis** for the actual state of the appliance and the result of the model, the remaining measures are the same as the above-discussed graphs.

Figure 42 shows **the outcomes of the three multi-features models tested for SA** on two days of Farm1. The results of models overlap with the actual states, which signifies the virtuous performances of all the models. It was observed that the unique combination of active and reactive power reduced the complexity of an LSTM layer to recognise the pattern and enhance the overall performance of the model. Still, the consistency of convolutional layer-based models for SA is better than LSTM model. The enhancement in training of model certified that the overall performance improves with an increase in the number of features while training.

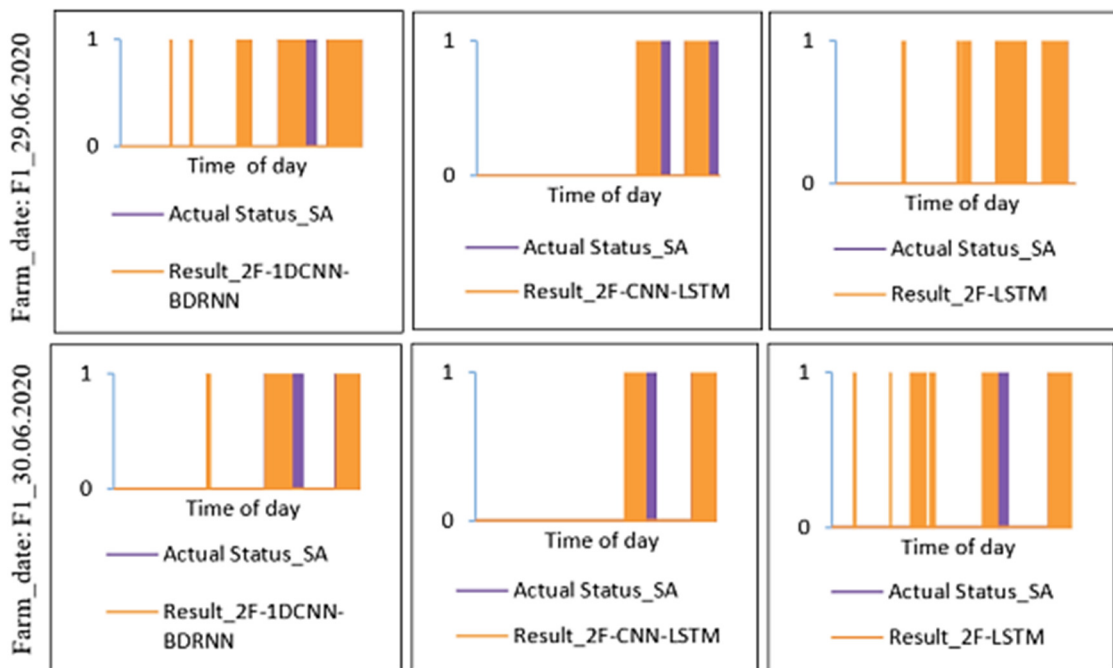


Figure 42 Comparison of the results of three multi-features model for SA tested on different dates

where,

Actual_Status_SA: ground truth state of SA (on-off).

Result_2F_1DCNN-BDRNN: result from Model-4: Multi-feature 1D-CNN-BDRNN.

Result_2F _CNN_LSTM: result from Model-5: Multi-feature 1D-CNN-LSTM.

Results_2F _LSTM: result from Model-6: Multi-feature LSTM.

Figure 43 shows, the F1 score of the LSTM model is lower compared to the other two convolutional layer-based models, which are having a score of about 0.65 and above. The consistency in the results of all the models is significantly small. The availability of dataset for training was limited to only one farm. Therefore, to ensure the performance and robustness of the models, it is essential to train and test them with datasets of the multiple farms. The 1DCNN-LSTM model outperformed among the three models with the highest F1 score.

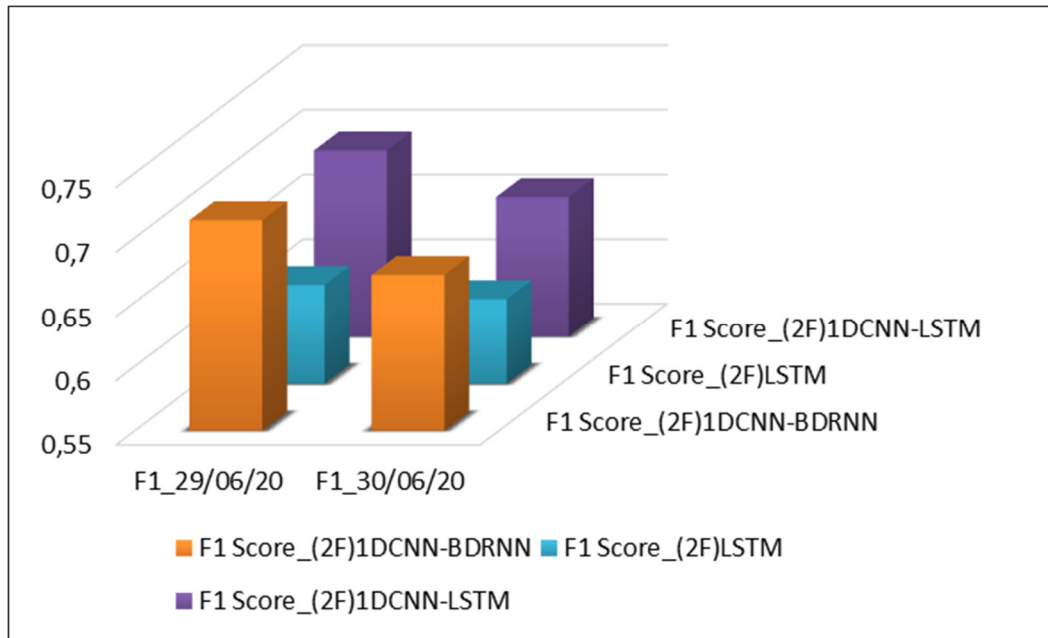


Figure 43 F1-Score comparison of three multi-features model for SA tested on different dates

where,

F1 is the Farm1.

F1 Score_(2F)1DCNN-BDRNN: result from Model-4: Multi-feature 1D-CNN-BDRNN

F1 Score_(2F)1DCNN_LSTM: result from Model-5: Multi-feature 1D-CNN-LSTM.

F1 Score_(2F)LSTM: result from Model-6: Multi-feature LSTM.

The results of the three models tested for VP on a two-days dataset from Farm1 are shown in Figure 44. It is challenging to categorise the performances of models due to the overlapping of actual states and the results. However, the consistency of the results is comparable for all the three models.

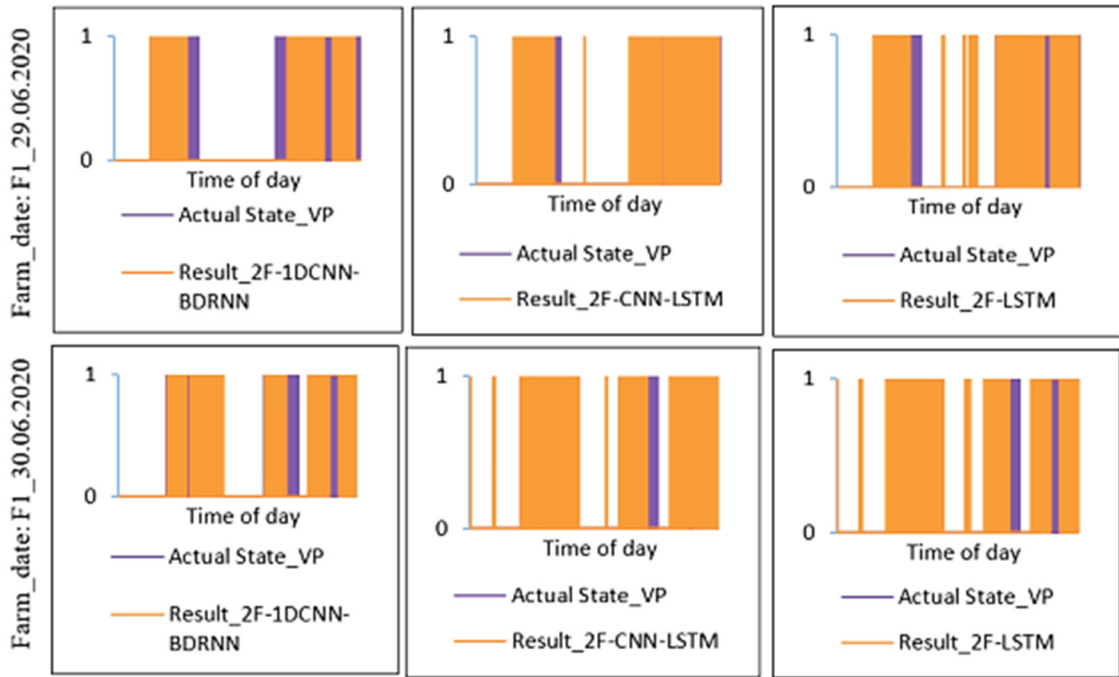


Figure 44 Comparison of the results of three multi-features model for VP tested on different dates

where,

Actual_Status_VP: ground truth state of VP (on-off).

Result_2F_1DCNN-BDRNN: result from Model-4: Multi-feature 1D-CNN-BDRNN.

Result_2F_CNN_LSTM: result from Model-5: Multi-feature 1D-CNN-LSTM.

Results_2F_LSTM: result from Model-6: Multi-feature LSTM.

F1 score of 1DCNN-BDRNN model is lower than the LSTM based models, as shown in Figure 45.

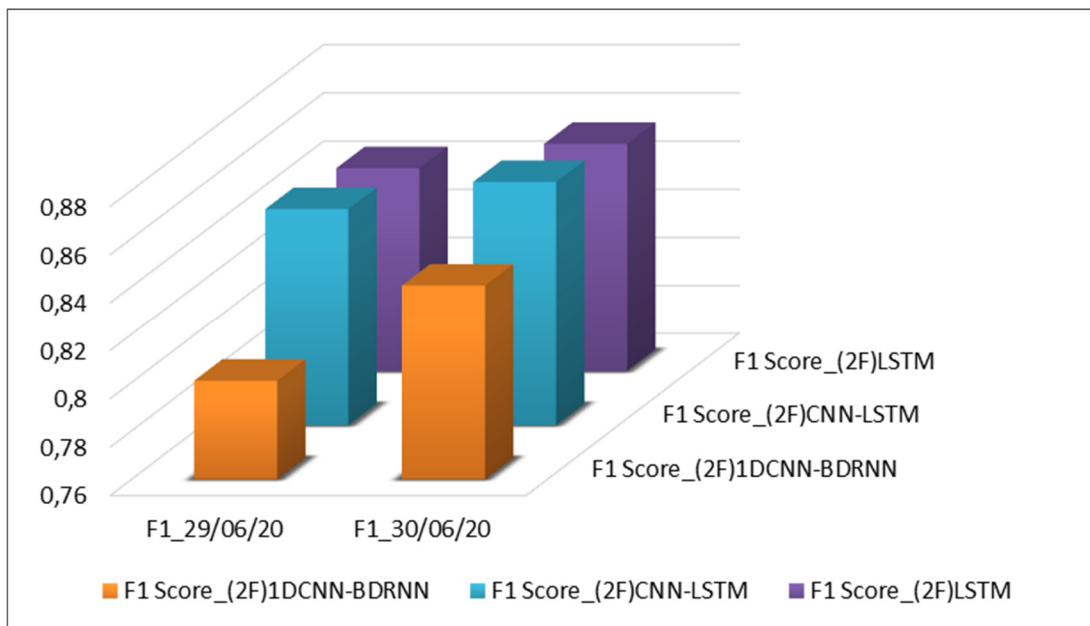


Figure 45 F1-Score comparison of three multi-features model for VP tested on different dates

where,

F1 is the Farm1.

F1 Score_(2F)1DCNN-BDRNN: result from Model-4: Multi-feature 1D-CNN-BDRNN

F1 Score_(2F)CNN_LSTM: result from Model-5: Multi-feature 1D-CNN-LSTM.

F1 Score_(2F)LSTM: result from Model-6: Multi-feature LSTM.

The F1 score of the multi-feature LSTM model is significantly higher compared to the model trained with one feature for VP. It specifies that the performance of an LSTM based model prominently improves with multi-feature input data. F1 score for both the LSTM based models are close to 0.85, and the results show consistency for both the days. Model-6 outperformed for VP with the highest F1 score.

6. Conclusion and Outlook

6.1 General Conclusion

Literature research was carried out around the deep-learning-based neural network model development, as this is the latest trend in the NILM field. This thesis presented the potential of energy saving in dairy farms and proposed a deep-learning-based power disaggregation algorithm developed in Python. The measured load profile of the four appliances and the aggregated load profile of the four dairy farms located in Bavaria, Germany, were accumulated and analysed. By analysing the collected data, four main problems were identified and later rectified (as discussed in chapter 3.2.1). Several deep-learning based models were developed, trained, and tested as the operational patterns and behaviours of appliances were utterly different from each other. It is observed that the data gathered was not adequate to train a model which imparts the desired results for all the four appliances. The results of the three most performing models developed with one feature (i.e., active power) were discussed, along with their challenges. Another observation is that the performance of a model can be enhanced by adding a number of features for training the model. This thesis also identifies that the development of the multi-feature-based model for SA and VP improves the results with better F1 score. Performances of all the models were evaluated with F1-score and presented by testing the models on several days of unseen load profiles. The 1D-CNN-BDRNN and 1D-CNN-LSTM outperformed for MK. All three models performed excellent for MP, but the datasets used for the training and testing of the model were limited to only one farm. The LSTM model performed outstandingly for VP, and only the 1D-CNN-BDRNN model performed for SA. The final observation is that it is not recommended to use only one model for all the appliances. Instead, it is suggested to combine the best performing model for each appliance and implement the final combined algorithm for the best results.

6.2 Limitation and Future Work

The backbone of any deep neural network model is the amount of data. It was initially planned to collect data from at least six to eight farms, which ended up to only four farms due to some external conditions. While dealing with a tremendous amount of data for training the neural network, the use of GPU becomes essential. The availability of GPU could accelerate the whole process and contribute to improving the overall performance. It is recommended to make the data measurement process more precise to avoid the changes in formats, which will help in data pre-processing. Secondly, more data should be collected with multiple features; this will make learning easier.

In the future, the models could be modified to work on real-time data. The existing model can also be improvised, or a new model can be developed to estimate the power consumption of the appliance along with the on-off state.

References

- Agarwal, Ashish, Barham, Paul, Brevdo, Eugene, Chen, Zhifeng, Citro, Craig, Corrado, Greg.S, Davis, Andy, Dean, Jeffrey, Devin, Matthieu, Ghemawat, Sanjay, Goodfellow, Ian, Harp, Andrew, Irving, Geoffrey, Isard, Michael, Jia, Yangqing, Jozefowicz, Rafal, Kaiser, Lukasz, Kudlur, Manjunath, Levenberg, Josh, Man, Dandelion, Monga, Rajat, Moore, Sherry, Murray, Derek, Olah, Chris, Schuster, Mike, Shlens, Jonathon, Steiner, Benoit, Sutskever, Ilya, Talwar, Kunal, Tucker, Paul, Vanhoucke, Vincent, Vasudevan, Vijay, Vi, Fernanda, Vinyals, Oriol, Warden, Pete, Wattenberg, Martin, Wicke, Martin, Yu, Yuan and Zheng, Xiaoqiang (2015), "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems", available at: <https://www.tensorflow.org/> (accessed 2 November 2020).
- Algorithmia (2018), "Introduction to Optimizers", available at: <https://algorithmia.com/blog/introduction-to-optimizers> (accessed 16 November 2020).
- Arnold, Ludovic, Rebecchi, Sébastien, Chevallier, Sylvain and Paugam-Moisy, Hélène (Eds.) (2011), *An Introduction to Deep Learning*, 19th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, Bruges, Belgium, April 27-28-29, 2011 proceedings.
- Basheer, Imad and Hajmeer, Maha (2000), "Artificial neural networks: fundamentals, computing, design, and application", *Journal of Microbiological Methods*, Vol. 43 No. 1, pp. 3–31.
- Berliner Energieinstitut GmbH (2019), "EMONIO P3- 3-Phase Power Logger", available at: http://emonio.com/images/docs/Emonio_datasheet.pdf (accessed 17 November 2020).
- Bernard, Timo (2018), "Non-Intrusive Load Monitoring (NILM): Combining multiple Non-Intrusive Load Monitoring (NILM): Combining multiple distinct Electrical Features and Unsupervised Machine Learning Techniques", Doctoral, Fraunhofer IMS, Universit at Duisburg-Essen, Von der Fakult at f ur Ingenieurwissenschaften, Abteilung Informatik und Angewandte Kognitionswissenschaft, 22 June.
- Biansoongnern, Somchai and Plangklang, Boonyang (2016 - 2016), "Nonintrusive load monitoring (NILM) using an Artificial Neural Network in embedded system with low sampling rate", in *2016 13th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON), 28/06/2016 - 01/07/2016, Chiang Mai, Thailand*, IEEE, pp. 1–4.
- Brownlee, Jason (2020a), "How Do Convolutional Layers Work in Deep Learning Neural Networks?", available at: <https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/> (accessed 16 November 2020).
- Brownlee, Jason (2020b), "How to Develop Convolutional Neural Network Models for Time Series Forecasting", available at: <https://machinelearningmastery.com/how-to-develop-convolutional-neural-network-models-for-time-series-forecasting/> (accessed 16 November 2020).
- Brownlee, Jason (2020c), "How to Use StandardScaler and MinMaxScaler Transforms in Python", available at: <https://machinelearningmastery.com/standardscaler-and-minmaxscaler-transforms-in-python/> (accessed 19 November 2020).
- Brownlee, Jason (2020d), "How to Calculate Precision, Recall, and F-Measure for Imbalanced Classification", available at: <https://machinelearningmastery.com/precision-recall-and-f-measure-for-imbalanced-classification/> (accessed 20 November 2020).

- BUNDESMINISTERIUM FÜR WIRTSCHAFT UND ENERGIE (2019), *Barometer Digitalisierung der Energiewende 2018*, available at: <https://www.bmwi.de/Redaktion/DE/Publikationen/Studien/barometer-digitalisierung-der-energiewende.html> (accessed 10 November 2020).
- Chanan, Gregory, Chintala, Soumith, Gross, Sam and Paszke, Adam (2020), "PyTorch Development", available at: <https://pytorch.org/> (accessed 28 October 2020).
- Chollet, François (2018), *Deep learning with Python*, Safari Tech Books Online, Shelter Island, NY, Manning.
- Cole, Agnim and Albicki, Alexander (2000), *Nonintrusive Identification of Electrical Loads in a Three-phase Environment Based on Harmonic Content: 2000 IEEE Instrumentation and Measurement Technology Conference*, Piscataway, IEEE.
- Dertat, Arden (2017), "Applied Deep Learning - Part 1: Artificial Neural Networks", available at: <https://towardsdatascience.com/applied-deep-learning-part-1-artificial-neural-networks-d7834f67a4f6> (accessed 6 November 2020).
- Donges, Niklas (2020), "GRADIENT DESCENT: AN INTRODUCTION TO ONE OF MACHINE LEARNING'S MOST POPULAR ALGORITHMS", available at: <https://builtin.com/data-science/gradient-descent> (accessed 16 November 2020).
- dprogrammer (2019), "Artificial Intelligence, Tutorial: RNN, LSTM & GRU", available at: <http://dprogrammer.org/rnn-lstm-gru> (accessed 6 November 2020).
- Ehrhardt-Martinez, Karen, Donnelly, Kat A. and Laitner, John A. (2010), *Advanced Metering Initiatives and Residential Feedback Programs: A Meta-Review for Household Electricity-Saving Opportunities*, available at: <https://www.aceee.org/research-report/e105> (accessed 10 November 2020).
- Federal Ministry for Economic Affairs and Energy (2016), *The Energy Transition*, Germany, available at: <https://www.bmwi.de/Redaktion/EN/Pressemitteilungen/2016/20160708-gabriel-die-naechste-phase-der-energiewende-kann-beginnen.html> (accessed 10 November 2020).
- Feng, Junxi, He, Xiaohai, Teng, Qizhi, Ren, Chao, Chen, Honggang and Li, Yang (2019), "Reconstruction of porous media from extremely limited information using conditional generative adversarial networks", *Physical review. E*, Vol. 100 No. 3-1, p. 33308.
- Figueiredo, Marisa B. (2013), "Contributions to Electrical Energy Disaggregation in a Smart Home", Doctoral, University of Coimbra, Department of Informatics Engineering, Portugal, 09/2013.
- Fiol, Albert (2016), "Algorithms for Energy Disaggregation", Master's dissertation, Universitat Politècnica de Catalunya, Departament de Ciències de la Computació, Barcelona, Spain, 2016.
- Goodfellow, Ian, Bengio, Yoshua and Courville, Aaron (2016), *Deep learning*, Cambridge, Massachusetts, London, England, MIT Press.
- Graville, Vincent (2017), "Difference between Machine Learning, Data Science, AI, Deep Learning, and Statistics", available at: <https://www.datasciencecentral.com/profiles/blogs/difference-between-machine-learning-data-science-ai-deep-learning> (accessed 6 November 2020).
- Hart, George W. (1985), *Prototype nonintrusive appliance load monitor (Tech. Rep.)*, Cambridge, available at: <http://www.georgehart.com/research/Hart1985.pdf> (accessed 11 November 2020).

- Hinton, Geoffrey E., Osindero, Simon and Teh, Yee-Whye (2006), "A fast learning algorithm for deep belief nets", *Neural computation*, Vol. 18 No. 7, pp. 1527–1554.
- Hochreiter, S. and Schmidhuber, J. (1997), "Long short-term memory", *Neural computation*, Vol. 9 No. 8, pp. 1735–1780.
- Huss, Anders (2015), "Hybrid Model Approach to Appliance Load Disaggregation. EXPRESSIVE APPLIANCE MODELLING BY COMBINING CONVOLUTIONAL NEURAL NETWORKS AND HIDDEN SEMI MARKOV MODELS.", Degree project, KTH ROYAL INSTITUTE OF TECHNOLOGY, Stockholm, Sweden, 2015.
- Jia, Yangqing and Shelhamer, Evan and Donahue, Jeff and Karayev, Sergey and Long, Jonathan and Girshick, Ross and Guadarrama, Sergio and Darrell, Trevor (2014), "Caffe: Convolutional Architecture for Fast Feature Embedding".
- Kelly, Jack and Knottenbelt, William (2015), "Neural NILM: Deep Neural Networks Applied to Energy Disaggregation", pp. 55–64.
- Kelly, Jack D. (2017), "Disaggregation of Domestic Smart Meter Energy Data", PhD, University of London, Imperial College of Science, Technology and Medicine, London, 2017.
- Kelly, Michael (2018), "Germany moving ahead with smart meter rollout plans", available at: <https://guidehouseinsights.com/search-results?search=Germany%20moving%20ahead%20with%20smart%20meter%20rollout%20plans> (accessed 10 November 2020).
- Khandelwal, Renu (2018), "Machine learning Gradient Descent", available at: <https://arshren.medium.com/gradient-descent-5a13f385d403> (accessed 16 November 2020).
- Kim, Jihyun, Le, Thi-Thu-Huong and Kim, Howon (2017), "Nonintrusive Load Monitoring Based on Advanced Deep Learning and Novel Signature", *Computational intelligence and neuroscience*, Vol. 2017, p. 4216281.
- Kiranyaz, Serkan, Ince, Turker and Gabbouj, Moncef (2016), "Real-Time Patient-Specific ECG Classification by 1-D Convolutional Neural Networks", *IEEE transactions on bio-medical engineering*, Vol. 63 No. 3, pp. 664–675.
- Kiranyaz, Serkan, Ince, Turker, Hamila, Ridha and Gabbouj, Moncef (2015), "Convolutional Neural Networks for patient-specific ECG classification", *Annual International Conference of the IEEE Engineering in Medicine and Biology Society. IEEE Engineering in Medicine and Biology Society. Annual International Conference*, Vol. 2015, pp. 2608–2611.
- Kolter, J. Z., Batra, Siddarth and Ng, Andrew Y. (2010), "Energy disaggregation via discriminative sparse coding", Vol. 1, Pages 1153–1161.
- Krizhevsky, Alex, Sutskever, Ilya and Hinton, Geoffrey (Eds.) (2012), *ImageNet Classification with Deep Convolutional Neural Networks*.
- Lackmann GmbH & Co. KG (2020), "Smart Metering MT382", available at: <https://www.lackmann.de/hardware/elektrizitaetszaehler/industriegewerbezaehler> (accessed 17 November 2020).
- Lecun, Y., Bottou, L., Bengio, Y. and Haffner, P. (1998), "Gradient-based learning applied to document recognition", *Proceedings of the IEEE*, Vol. 86 No. 11, pp. 2278–2324.
- LeCun, Yann, Bengio, Yoshua and Hinton, Geoffrey (2015), "Deep learning", *Nature*, Vol. 521 No. 7553, pp. 436–444.
- Li, Kuan-Ching, Chen, Xiaofeng and Susilo, Willy (2019), *Advances in Cyber Security: Principles, Techniques, and Applications*, Singapore, Springer Singapore.

- Lukoševičius, Mantas and Jaeger, Herbert (2009), “Reservoir computing approaches to recurrent neural network training”, *Computer Science Review*, Vol. 3 No. 3, pp. 127–149.
- Macal, Charles. M. and North, Mystic J. (2010), “Tutorial on agent-based modelling and simulation”, *Journal of Simulation*, Vol. 4 No. 3, pp. 151–162.
- MacKay, David (2010), *Sustainable energy - without the hot air*, Reprinted., Cambridge, UIT Cambridge.
- Mahendru, Khyati (2019), “A Detailed Guide to 7 Loss Functions for Machine Learning Algorithms with Python Code”, available at: <https://www.analyticsvidhya.com/blog/2019/08/detailed-guide-7-loss-functions-machine-learning-python-code/> (accessed 16 November 2020).
- Montreal Institute for Learning Algorithms (2007), “Theano 1.0.0”, available at: <http://www.deeplearning.net/software/theano/NEWS.html> (accessed 23 October 2020).
- Nascimento, Pedro P.M.d. (2016), “APPLICATIONS OF DEEP LEARNING TECHNIQUES ON NILM”, Master’s dissertation, Universidade Federal do Rio de Janeiro, Rio de Janeiro, Brazil, 04/2016.
- Neser, Dr. S., Neiber, Josef, Niedermeier, Josef, Götz, Manfred, Kraus, Ludwig and Pettinger, Prof. K.-H. (2014), “Energieverbrauch im Milchviehbetrieb Effizienz und Einsparpotential”, available at: https://www.lfl.bayern.de/mam/cms07/publikationen/daten/informationen/energieverbrauch_im_milchviehstall_065687.pdf (accessed 6 November 2020).
- Nicholson, Chris and Kokorin, Vyacheslav (2013), “Eclipse Deeplearning4j”, available at: <https://projects.eclipse.org/proposals/eclipse-deeplearning4j> (accessed 28 October 2020).
- oinkina, Gers, Felix, Cummins, Fred, Fernandez, Santiago, Bayer, Justin, Wierstra, Daan, Togelius, Julian, Gomez, Faustino, Gagliolo, Matteo and Graves, Alex (2015), “Understanding LSTM Networks”, available at: <https://colah.github.io/posts/2015-08-Understanding-LSTMs/> (accessed 6 November 2020).
- Palma, Wilfredo (2016), *Time series analysis, Wiley series in probability and statistics*.
- Parson, Oliver (2014), “Unsupervised Training Methods for Non-intrusive Appliance Load Monitoring from Smart Meter Data”, Doctoral, University of Southampton, Faculty of Engineering and Physical Sciences, 04/2014.
- Parson, Oliver, Ghosh, Siddhartha, Weal, Mark and Rogers, Alex (2014), “An unsupervised training method for non-intrusive appliance load monitoring”, *Artificial Intelligence*, Vol. 217, pp. 1–19.
- Pereira, Lucas and Nunes, Nuno (2018), “Performance evaluation in non-intrusive load monitoring: Datasets, metrics, and tools—A review”, *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, Vol. 8 No. 6, pp. 1–17.
- Rafiq, Hasan, Zhang, Hengxu, Li, Huimin and Ochani, Manesh K. (2018 - 2018), “Regularized LSTM Based Deep Learning Model: First Step towards Real-Time Non-Intrusive Load Monitoring”, in *2018 IEEE International Conference on Smart Energy Grid Engineering (SEGE), 12.08.2018 - 15.08.2018, Oshawa, ON, IEEE*, pp. 234–239.
- Raiker, Gautam A., Subba Reddy, B., Umanand, L., Yadav, Aman and Shaikh, Mujeeba M. (2018 - 2018), “Approach to Non-Intrusive Load Monitoring using Factorial Hidden Markov Model”, in *2018 IEEE 13th International Conference on Industrial and Information Systems (ICIIS), 01/12/2018 - 02/12/2018, Rupnagar, India, IEEE*, pp. 381–386.

- ROSENBLATT, Frank (1958), “The perceptron: a probabilistic model for information storage and organization in the brain”, *Psychological review*, Vol. 65 No. 6, pp. 386–408.
- Scully, Pdraig (2019), “Smart Meter Market Report 2019-2024. Smart Meter Market 2019: Global penetration reached 14% – North America, Europe ahead”, available at: <https://iot-analytics.com/smart-meter-market-2019-global-penetration-reached-14-percent/> (accessed 10 November 2020).
- Srinivasamurthy, Ravisutha S. (2018), “Understanding 1D Convolutional Neural Networks Using Multiclass Time-Varying Signals”, Clemson University, Clemson, South Carolina, 8/2018.
- Stewart, Matthew (2019), “Neural Network Optimization”, available at: <https://towardsdatascience.com/neural-network-optimization-7ca72d4db3e0> (accessed 16 November 2020).
- Valenti, Michele, Bonfigli, Roberto, Principi, Emanuele and Squartini, and S. (2018 - 2018), “Exploiting the Reactive Power in Deep Neural Models for Non-Intrusive Load Monitoring”, in *2018 International Joint Conference on Neural Networks (IJCNN), 08.07.2018 - 13.07.2018, Rio de Janeiro, IEEE*, pp. 1–8.
- Verma, Shiva (2019), “Understanding 1D and 3D Convolution Neural Network | Keras”, available at: <https://towardsdatascience.com/understanding-1d-and-3d-convolution-neural-network-keras-9d8f76e29610#:~:text=In%201D%20CNN%2C%20kernel%20moves,2D%20CNN%20is%203%20dimensional.> (accessed 16 November 2020).
- Vine, Desley, Buys, Laurie and Morris, Peter (2013), “The Effectiveness of Energy Feedback for Conservation and Peak Demand: A Literature Review”, *Open Journal of Energy Efficiency*, Vol. 02 No. 01, pp. 7–15.
- Zeifman, Michael and Roth, Kurt (2012), *Non-intrusive Appliance Load Monitoring (NIALM): Promise and Practice*, USA.
- Zerrahn, Alexander, Schill, Wolf-Peter and Kemfert, Claudia (2018), “On the economics of electrical storage for variable renewable energy sources”, *European Economic Review*, Vol. 108, pp. 259–279.
- Zhang, Zijian, He, Jialing, Zhu, Liehuang and Ren, Kui, in Li, K.-C., Chen, X. and Susilo, W. (Eds.) (2019), “Non-intrusive Load Monitoring Algorithms for Privacy Mining in Smart Grid”, *remove Advances in Cyber Security: Principles, Techniques, and Applications*, Vol. 66, Singapore, Springer Singapore, pp. 23–48.
- Zoha, Ahmed, Gluhak, Alexander, Imran, Muhammad A. and Rajasegarar, Sutharshan (2012), “Non-intrusive load monitoring approaches for disaggregated energy sensing: a survey”, *Sensors (Basel, Switzerland)*, Vol. 12 No. 12, pp. 16838–16866.

Appendix

Clarification on missing data measured from EMONIO

Q1: We are facing an issue in the measurement data. The values are missing in several measurement files. We have attached few samples. Do you know the reason?

Date	Time	V12	Unit	V23	Unit	V31	Unit
14.06.2019	14:37:10	0401.0	ACV	0403.1	ACV	0402.1	ACV
14.06.2019	14:37:12	0400.5	ACV	0402.8	ACV	0402.1	ACV
14.06.2019	14:37:14	0399.7	ACV	0402.0	ACV	0400.9	ACV
14.06.2019	14:37:19	0399.6	ACV	0401.5	ACV	0400.9	ACV
14.06.2019	14:55:42	0399.1	ACV	0401.3	ACV	0400.3	ACV
14.06.2019	14:55:44	0399.4	ACV	0400.9	ACV	0400.5	ACV
14.06.2019	14:55:46	0399.2	ACV	0401.3	ACV	0400.3	ACV
14.06.2019	14:55:48	0399.3	ACV	0400.9	ACV	0399.8	ACV
14.06.2019	14:55:50	0399.3	ACV	0400.9	ACV	0400.4	ACV

(a)

```
1593891786;4.7.2020 21:43:06;227,0;12,0;1897,  
1593891787;4.7.2020 21:43:07;227,1;12,0;1898,  
1593891788;4.7.2020 21:43:08;227,1;12,0;1900,  
1593891789;4.7.2020 21:43:09;227,1;12,0;1895,  
1593891790;4.7.2020 21:43 10;227,0;11,9;1898,  
1593891791;4.7.2020 21:43 11;226,9;12,0;1900,  
1593920726;5.7.2020 05:45 26;0,0;0,0;0,0,0;  
1593920727;5.7.2020 05:45 27;231,8;11,9;1833,  
1593920728;5.7.2020 05:45 28;231,7;12,4;1855,
```

(b)

Figure 46 Screenshot of query asked from EMONIO Team

Response:

Name: Anonymous

Company: Berliner Energieinstitut GmbH

Position: Technical Support

Answer: The missing values marked in yellow look like a reboot and/or a minor network deadlock that resulted in a hang of a few seconds. The second, missing several hours I have no theory besides power loss.