Universidade de Aveiro
2021

Rafael
Teixeira

Desagregação de consumos energéticos usando Machine Learning

Energy disaggregation using Machine Learning

Universidade de Aveiro
2021

Rafael Teixeira

Desagregação de consumos energéticos usando Machine Learning

Energy disaggregation using Machine Learning

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Informática, realizada sob a orientação científica do Doutor Diogo Nunes Pereira Gomes, Professor auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro, e do Doutor Mário Luís Pinto Antunes, Professor adjunto convidado do Instituto Superior de Contabilidade e Administração de Aveiro da Universidade de Aveiro.

Dedico este trabalho à minha mãe por acreditar em mim mesmo nos momentos mais difíceis e me incentivar a lutar pelo que eu quero.

o júri / the jury

presidente / president	Prof. Doutor Luís Filipe de Seabra Lopes Professor Associado da Universidade de Aveiro
vogais / examiners committee	Prof. Doutor Paulo Jorge Freitas de Oliveira Novais Professor Associado com Agregação da Univerisadde do Minho
	Prof. Doutor Mário Luís Pinto Antunes Professor Adjunto Convidado da Universidade de Aveiro

agradecimentos / acknowledgements

Quero começar por agradecer à minha mãe que sempre me apoiou durante o meu percurso académico e ao meu irmão que mesmo com todas as nossas discussões e divergências esteve sempre lá quando precisei.
Agradeço ao professor Diogo e Mário pela paciência e disponibilidade que sempre demonstraram para responder às diversas questões durante o desenvolvimento da dissertação e à Withus por me deixarem desenvolve-la juntamente com eles e por todos os recursos disponibilizados.
Quero ainda agradecer a todos os meus amigos que direta ou indiretamente tornaram isto possível. Com um especial agradecimento ao Márcio que embora não saiba nada sobre informática me ajudou com tudo o resto, ao Direito que me acompanhou estes 5 anos e que me incentivou a ser uma melhor pessoa, ao Edu e Bastos, que me mostraram que a universidade não é só fazer cadeiras e com os quais me diverti bastante e ao Daniel e Pedro que foram a minha família na UA e são como irmãos.
Para finalizar quero ainda agradecer à minha restante família que me acompanhou durante estes 5 anos ajudando das mais diversas formas, agradecendo particularmente aos meus primos Ricardo, Diogo, Catarina e Daniela pelo apoio prestado durante este percurso que nem sempre foi fácil e por todas as experiências partilhadas.
Embora este documento tenha sido escrito por mim, sem a ajuda de todos vós não seria possível e por isso queria agradecer-vos.

Palavras Chave

Desagregação de Consumos Elétricos, Monitorização de Carga Não Intrusiva,Redes Neuronais Convolucionais, Redes Neuronais Recorrentes, Multilayer Perceptron, Redes Residuais, Transferência de Aprendizagem

Resumo

Hoje em dia estamos rodeados de dispositivos elétricos. Quer seja em casa, pela máquina de lavar, o microondas ou o forno ou no emprego pelo computador, o telemóvel ou a impressora. Estes dispositivos ajudam-nos diariamente, mas com a sua popularização o consumo energético atingiu valores preocupantes. Numa tentativa de reduzir o consumo energético, os governos começaram a introduzir políticas de educação energética para ensinar os consumidores a reduzir o desperdício energético. Uma das medidas foi a implementação generalizada de smart meters, que permitem ao consumidor saber quanta energia está a ser consumida a qualquer altura através de um ecrã no contador da casa. Mesmo sendo bem recebida, esta medida não é suficiente uma vez que os estudos indicam que os melhores resultados são obtidos através de feedback ao nível do dispositivo em tempo real. Para obtermos este feedback existem duas formas, podemos medir cada tomada numa dada casa, o que é inviável para uma implementação em larga escala, ou desagregar a energia registrada pelo smart meter que já está presente na casa. NILM ou Non-Intrusive Load Monitoring é o nome dado à segunda opção onde a energia agregada da casa é usada para descobrirmos a energia consumida por cada dispositivo elétrico. Para resolver este problema foram propostas várias alternativas, desde HMMs a GSP, onde os modelos de deep learning obtiveram resultados notáveis sendo agora o estado da arte. Com o objetivo de produzir um sistema NILM completo, a Withus juntou-se à Universidade de Aveiro e juntos propuseram esta dissertação. O objetivo inicial era o desenvolvimento de um modelo de machine learning para desagregar consumos elétricos. Contudo, durante análise do estado da arte, deparamo-nos com a necessidade de criar um novo dataset, o que levou à extensão da proposta inicial para incluir também o pré-processamento e conversão do dataset. Para desagregação de consumos elétricos propusemos três modelos: uma rede neuronal convolucional com blocos residuais, uma rede neuronal recorrente e um multilayer perceptron que usa discrete wavelet transforms como features. Estes modelos passaram por diversas iterações, sendo avaliados primeiro na tarefa de classificação ON/OFF e depois modificados e avaliados para desagregação. Os modelos foram ainda comparados com os do estado da arte presentes no NILMTK, onde apresentaram melhores resultados que a alternativa real-time, dAE, diminuindo o NRMSE em média 49% ficando próximos da melhor alternativa que classifica com atraso, Seq2Point, apresentando um erro pior, em média, de 17%. Para além disso, também analisamos os melhores modelos da experiência anterior no benefício de usar transfer learning entre datasets, onde os resultados mostram uma melhoria marginal quando usamos transfer learning. Este documento apresenta a definição do esboço da solução, as múltiplas opções consideradas para processamento de dataset e qual a melhor, a evolução dos modelos, os seus resultados e a comparação com os modelos do estado da arte na capacidade de generalização entre diferentes casas e de transfer learning entre datasets.

Keywords

Non-Intrusive Load Monitoring, NILM, NIALM, Convolutional Neural Networks, Recurrent Neural Networks, Multilayer Perceptron, Residual Network, Transfer Learning

Abstract

Nowadays, we are surrounded by electric appliances. Either at home by the washing machine, kettle, or oven, or work by the computer, cellphone, or printer. Such devices help us daily, but their popularization increased the energy consumption to concerning values. In an attempt to reduce energy consumption, governments started enforcing policies regarding energy education to teach homeowners how to reduce energy wastage on the demand side. One of those policies was the deployment of smart meters, which allow the consumer to know how much energy is being consumed at any given time through a display on the household energy meter. Even though this measure was well received, the studies show that the best results in energy conservation are obtained through real-time appliance level feedback. To get such feedback, one can either measure every outlet in a household, which is unviable for a broad deployment solution, or disaggregate the energy recorded by the smart meter. NILM or Non-Intrusive Load Monitoring is the name we give to the second option where we use the aggregated readings of a household to find the energy consumed by each appliance. There were many proposals to solve NILM ranging from HMMs to GSP, where deep learning models showed remarkable results, obtaining state-of-the-art results. With the intent to create a complete NILM solution, Withus partnered with the University of Aveiro and proposed this dissertation. The initial objective was to develop a machine learning model to solve NILM. Still, during the background analysis, we found the need to create a new dataset which led to the expansion of the initial proposal to include the dataset preprocessing and conversion. Regarding NILM, we proposed three new deep learning models: a convolutional neural network with residual blocks, a recurrent neural network, and a multilayer perceptron that uses discrete wavelet transforms as features. These models went through multiple iterations, being evaluated first in the simpler ON/OFF classification task and later modified and evaluated for the disaggregation task. We compared our models to the state-of-the-art ones proposed in NILMTK, where they presented better results than the real-time alternative, dAE, reducing the NRMSE on average by 49%. We also got close to the best option that classified with a 30 min delay, Seq2Point, increasing the error on average by 17%. Besides that, we also analyze the best models from the previous comparison on the benefit of transfer learning between datasets, where the results show a marginal performance improvement when using transfer learning. This document presents the solution outline definition, the multiple options considered for dataset processing and the best solution, the models' evolution and results, and the comparison with the state-of-the-art models regarding generalization to different houses and under transfer learning.

Contents

List of Figures

List of Tables

Glossary

AC	Air Conditioner
dAE	Denoising AutoEncoder
AFHMM	Additive Factorial Hidden Markov Model
API	Application Programming Interface
CFHMM	Conditional Factorial Hidden Markov Model
CFHSMM	Conditional Factorial Hidden Semi-Markov Model
CNN	Convolutional Neural Network
CSV	Comma Separated Value
DHMM	Difference Hidden Markov Model
DNN	Deep Neural Network
DTW	Dynamic Time Warping
DWT	Discrete Wavelet Transform
EMI	electromagnetic inference
EWMA	Exponential Weighted Moving Average
FFT	Fast Fourier Transform
FHMM	Factorial Hidden Markov Model
FHSMM	Factorial Hidden Semi-Markov Model
GRU	Gated Recurrent Unit
GSP	Graph Signal Processing
HEMS	Home Energy Management System
HIVE-COTE	Hierarchical Vote Collective of Transformation-Based Ensembles
HMM	Hidden Markov Model
ILM	Intrusive Load Monitoring
K-NN	K-Nearest Neighbour
LSTM	Long Short-Term Memory
MAE	Mean Absolute Error
MCC	Mathews Correlation Coefficient
ML	Machine Learning
MLP	Multilayer Perceptron
NILM	Non-Intrusive Load Monitoring
NILMTK	Non-Intrusive Load Monitoring Toolkit
NN	Neural Network
NRMSE	Normalized Root Mean Square Error
RF	Random Forest
RMS	Root Mean Square
RMSE	Root Mean Square Error
RNN	Recurrent Neural Network
SMA	Simple Moving Average
SMPS	Switch Mode Power Supply
STPD	Spike Timing Dependent Plasticity
SNN	Spiking Neural Network
SOA	state-of-the-art
STFT	Short-Time Fourier Transform
SVM	Support Vector Machine
YAML	Yet Another Markup Language

CHAPTER 1

Introduction

"Your fridge has been consuming 20% more energy since last month". Soon, this message will be a notification on our smartphones as the increase in energy consumption in the past decades made energy conservation and consumer education pressing topics.

1.1 Motivation

Since we still rely heavily on fossil fuels to produce electricity, the growth in energy consumption creates pressing environmental and economic issues. Given that the residential and commercial buildings account for approximately 60% of the energy consumed [1], several measures were proposed to reduce demand-side energy consumption. The roll-out of smart meters is one of them. These meters allow us to see the amount of electricity consumed by the whole house in real-time to help consumers understand how they utilize energy.

Although smart meters help educate consumers, the studies show that the best feedback mechanisms are based on appliance-level energy consumption feedback in real-time, achieving savings of up to 20% [2].

To understand why this happens, let us look at the way we buy groceries. If when we went to the supermarket, we received a monthly invoice at home instead of having the price labels of each product, it would be much harder to understand how much each product influenced the final bill. Nevertheless, that is how most of us "buy" electricity. With the smart meters, instead of the monthly invoice, we would know how much we spent on each grocery run we made during that month. The additional feedback would help us understand when we bought higher/lower-cost products on average, but still, it would be hard to pinpoint the price of each one. With appliance-level feedback, we would shop knowing how much each product costs, like we usually do, which is the best way to understand how to save on groceries.

So, in order to save on our electricity bill, we need the appliance level feedback. There are two options to get it: Intrusive Load Monitoring (ILM) or Non-Intrusive Load Monitoring (NILM). The first requires implementing a metering device per appliance to obtain their consumption, making this option expensive as the implementation and maintenance costs are high. The second one consists of a single meter that obtains the aggregated readings of the whole house and a disaggregation algorithm that uses those readings to obtain the consumption of each appliance. The global deployment of smart meters makes the second option much cheaper since the hardware cost of the solution is removed, and we can focus our attention on the disaggregation algorithms.

1.2 Proposal

With the intent to develop a NILM solution, the work presented in this dissertation describes the process of implementing a disaggregation algorithm using Machine Learning (ML). Once we obtain the energy consumed by each appliance through the algorithm, we can build an application that uses it to produce actionable feedback and send notifications, as mentioned initially, and hopefully reduce the energy consumption and the associated pollutant emissions.

We proposed three models using different types of architectures, the ResNet, DeepGRU, and Multilayer Perceptron (MLP). The first model is a Convolutional Neural Network (CNN) with skip connections, also known as a residual network, which allows us to build a deeper network without worrying about the vanishing gradient problem. The DeepGRU is a network composed of particular layers called Gated Recurrent Units (GRUs) that use the outputs from the previous examples to define the output of the current one. Finally, the MLP is a network that is composed of regular dense layers. What sets it apart from the others is using Discrete Wavelet Transforms (DWTs) statistical features instead of the raw electrical features as input. With these three models, we intend to study the viability of different research directions for future development.

To train and test the different models, we needed data. Although there are several publicly available NILM datasets, there is only one recorded in Portugal. Besides that, all of the analyzed datasets lack some criteria to be directly applicable to our objectives. This motivated Withus to create a new Portuguese dataset. Before using it, a cleaning and conversion step was done, so this dissertation also presents the work developed to prepare a dataset for NILM.

The models and dataset were built/converted to work with the Non-Intrusive Load Monitoring Toolkit (NILMTK) [3], allowing easy processing and analysis of datasets and straightforward training and testing of models.

Since we created both new models and a new dataset without any previous work, this dissertation represents the lessons learned, problems faced, their possible solutions, and the mistakes to avoid when starting a new NILM project.

1.3 Outline

The document structure is divided into the different stages of the development process of the project. After this introduction, Chapter 2 formally introduces NILM, why we need it, the challenges in developing a NILM solution, some of the proposed models and datasets, and the tools used for implementation. Chapter 3 discusses the outline of the dissertation, the dataset processing, and the proposed models. Followed by Chapter 4, presenting the results obtained by each iteration of the models and comparing the proposed models with state-of-the-art (SOA) ones. Finally, Chapter 5 presents some final considerations and future work.

1.4 Contributions

This dissertation resulted not only in this document but also in a contribution to the NILMTK project and the writing of a paper published at the ICWAPR conference.

The contribution to the NILMTK project was made to change the experiment Application Programming Interface (API), allowing the use of appliances' activations to produce balanced training and testing sets. Besides these changes submitted in a pull request [1], other errors detected during development were also flagged through various issues[2,3].

The paper titled *USING DEEP LEARNING AND KNOWLEDGE TRANSFER TO DISAGGREGATE ENERGY CONSUMPTION* [4] was published at the ICWAPR 2021 and documents the results obtained in the transfer learning experiments developed during the dissertation.

[1]`https://github.com/nilmtk/nilmtk/pull/944`
[2]`https://github.com/nilmtk/nilmtk/issues/939`
[3]`https://github.com/nilmtk/nilmtk-contrib/issues/59`

CHAPTER 2

Background

The proposal of a new system or model needs to be justified based on sound evidence. We should understand the problem we face, our options, and why we need this new system/model. This chapter was conceived to answer all these questions and justify the decisions made in the models' conceptualization and testing.

This research and development project was born from the idea of creating a NILM model ready for production. To build such a model posed many challenges, starting from the data it consumes to the actual model and its architecture. This chapter presents a formal definition of the NILM problem, why we should use it, and the research that supports our decisions when overcoming the different challenges we faced, serving mainly as a starting point for the development process.

2.1 What is NILM?

Starting with a written explanation, NILM is the task of estimating the energy consumed by each household appliance using only a single energy measurement device recording the aggregated consumption. The formal mathematical definition of the problem is presented in Equation 2.1, where the aggregated readings $P(t)$ result from the summation of the individual power consumed by each appliance i at time instant t and the Gaussian noise, ϵ, of the readings. Where N is the total number of appliances in the given household and $t = 1, 2, ..., T$, being T the last sample obtained by the aggregated meter.

$$P(t) = \sum_{i=1}^{N} p_i(t) + \epsilon \tag{2.1}$$

Based on this formula, NILM proposes to find each unknown appliance consumption, $p_i(t)$, using only the aggregated readings $P(t)$. This problem is not simple as the unknown appliances $p_i(t)$ are exchangeable [5].

2.2 Why NILM?

As we mentioned before in the introduction, the studies about feedback in electricity consumption pointed that the best way to reduce energy consumption is through real-time appliance-level feedback achieving reductions of 15% to 20% [6], [2].

Moreover, every household should receive this feedback without subscribing to an optional feedback service [6]. This means we need a low implementation cost solution that does not sacrifice the amount of information provided. Nowadays, no solution respects these requirements as the cost grows according to the information provided by the system [7].

To prove it, let us look at the associated costs of the broad deployment of the NILM alternative, namely ILM. This alternative obtains appliance-level information by monitoring each electrical appliance through an individual meter called a plug meter. Considering the average household, we could quickly reach 10 to 15 meters to monitor every appliance. With each of these meters costing around 5€, the average cost per house is 50 to 75€ just in hardware. Besides the hardware cost, we need to regard the installation and maintenance cost. The installation cost comes from the need for a setup process that allows meters to communicate their readings to the outside correctly. The maintenance cost is present as the plug meters are highly likely to be accidentally unplugged or malfunction.

These difficulties are readily confirmed when we look at many publicly available datasets. They usually record both appliance and aggregated-level readings, presenting gaps in the recording of the appliances due to network malfunctions or because they were unplugged. For example, Figure 2.1 presents the power consumption of the fridge in house 1 of the UKDALE dataset [8], one of the most popular datasets in NILM, where we can see that the recordings contain five days of missing values.

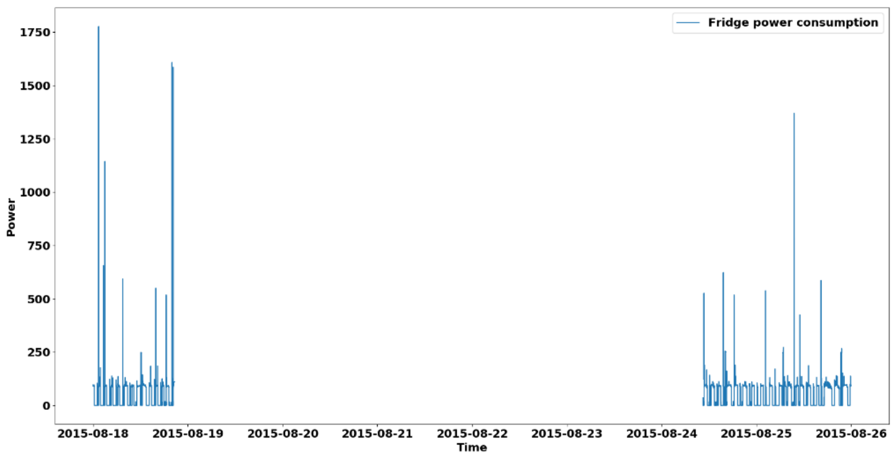


Figure 2.1: Missing values in UKDALE's house 1 fridge.

All of the obstacles we just mentioned render the broad deployment of an ILM solution unfeasible, leaving NILM as the preferred solution to obtain appliance-level feedback.

A successful NILM implementation will help develop a Home Energy Management System (HEMS) that will give real-time appliance-level feedback at a low and fixed implementation cost. The detailed feedback will help in energy conservation through behavior modifications, such as deferring loads, like washing machines or dishwashers, to off-peak hours or suggesting appliance replacements to more energy-efficient alternatives [7].

2.3 Features

After understanding what NILM proposes, we know that we are pretty limited in the information we can use in the disaggregation algorithm. It seems that we can only use the features of the aggregated electrical readings obtained by the smart meter. As we will see in this section, the number of features we can extract depends on several constraints, and they are not all obtained through the smart meter. They are divided into three categories: steady-state, transient-state, and non-traditional features, depending on when or how they are extracted. Both the steady and transient features are deduced from the electrical readings of the smart meter but at different periods and with different resolutions. The non-traditional features are obtained from external factors.

The diagram in Figure 2.2 presents the schematic division of the features used in the disaggregation algorithms.

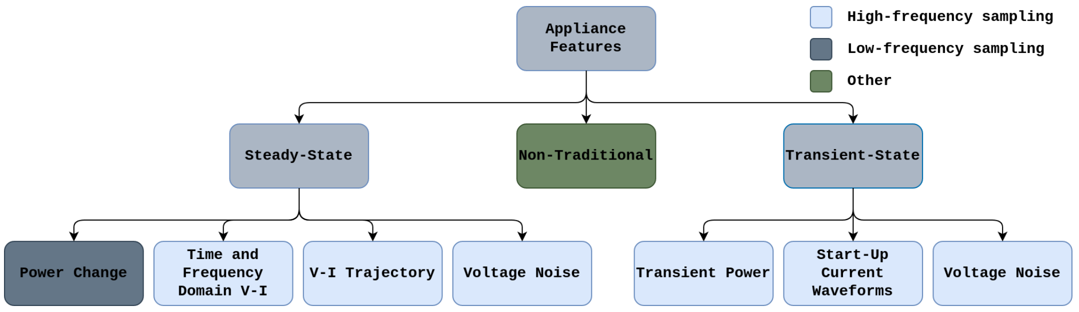


Figure 2.2: Schematic organization of the features used in NILM.

2.3.1 Appliance Signature

However, before we talk about the features, we need to understand one elemental concept that we will use throughout the dissertation.

Appliance signature is the unique consumption pattern intrinsic to each electrical appliance [9]. This consumption pattern is observed through the features we obtain from the electrical readings, and depending on them, the consumption pattern can be easier or harder to identify.

In Figure 2.3, the active and reactive power consumptions of the fridge are presented. In this case, the fridge's load signature combines the active and reactive power changes it makes when switching states.

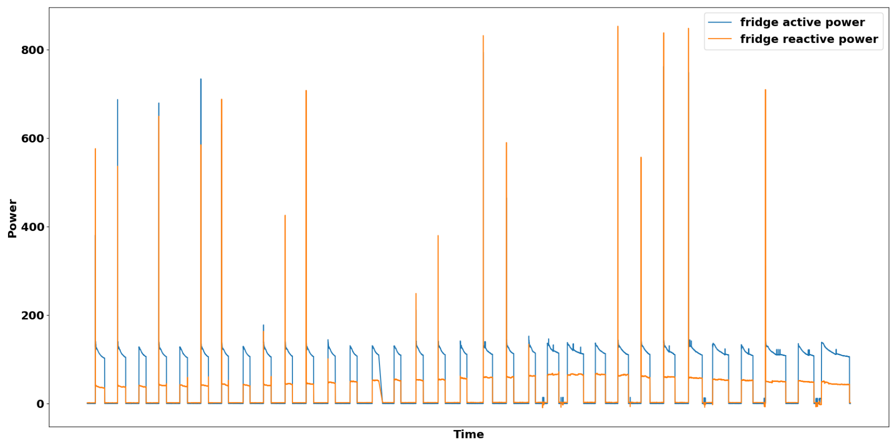


Figure 2.3: Fridge active and reactive power consumption over one day.

2.3.2 Steady-state features

The steady-state features focus on the stable states of the appliance operation, which are characterized by a window of consecutive samples where the difference between any two of them does not exceed a given tolerance value [10]. The size of the window depends on the sampling frequency used to record the samples [1]. Figure 2.4 presents the typical aggregated power consumption of a household with the steady-states identified. Each steady-state is preceded and followed by an edge in power consumption, and depending on the threshold used, we can or not consider the smaller edges. In this case, every edge was considered.

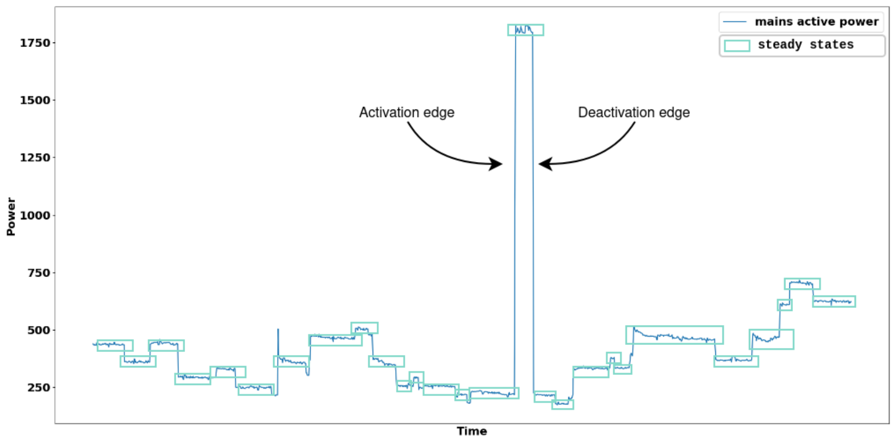


Figure 2.4: Aggregate power consumption with steady-state identification.

The features we can extract are divided into four categories: power changes, time and frequency domain V-I features, V-I trajectory analysis, and voltage noise analysis, where the first category uses low-frequency sampling while the remaining categories need high-frequency sampling.

Power change features use the differences in the energy consumed between steady-states to identify the appliance's signature. This approach is the most frequently used option in

NILM as it utilizes similar readings to what an actual smart meter provides. Researchers usually combine multiple measurements such as active and reactive power to implement this approach because active power is insufficient to distinguish between appliances with similar power consumption.

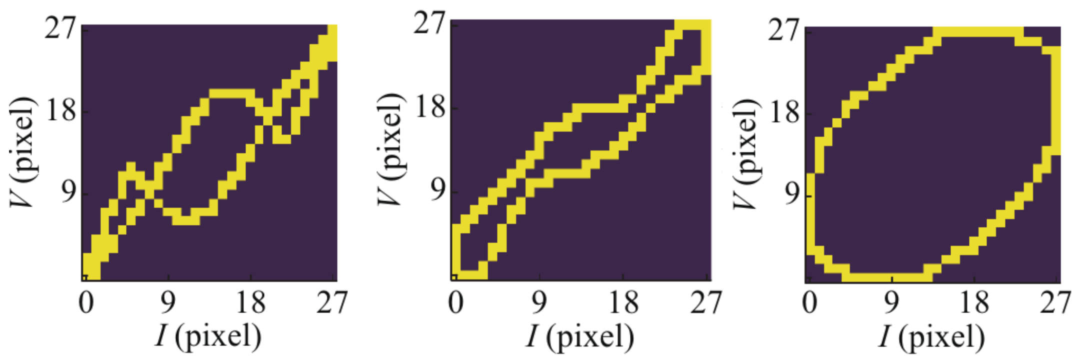
While the active power is the actual power consumption of the appliance, the reactive power represents the power that the device stores in itself and later returns to the house circuit. If the appliance's load is purely resistive, no reactive energy is consumed as the voltage, and current are in phase. When the load is purely reactive, no active power is consumed, and the waves are ninety degrees out of phase. Moreover, due to inductive and capacitive elements of the load, there is always a phase shift between current and voltage waveforms that generates or consumes reactive power [7].

Even though the combination of active and reactive power improves the distinguishability of the appliances' signatures, they are not enough to identify the lower consumption appliances [11]. To solve this, we can record more features, such as power factor, line current and voltage, frequency, and cumulative energy.

The high-frequency features, although less applied, have their benefits as they tend to create better appliance signatures.

The time and frequency domain characteristics from the voltage and current waveforms allow us to obtain features like current harmonics, current and voltage Root Mean Square (RMS), or the current peak and average. These features uniquely identify non-linear loads that draw non-sinusoidal current. The use of harmonics requires long recording periods to obtain a unique set of harmonic signatures, one for each combination of appliances. This is a disadvantage since we cannot gather all the appliance combinations in a training set [11], [12].

On the other hand, V-I trajectories create two-dimensional shapes from the plot of one voltage cycle by one current cycle, as shown in Figure 2.5. They allow us to distinguish between resistive, inductive, and electronic loads by extracting features from those shapes, such as asymmetry, looping direction, or area. This method is computationally intensive and cannot distinguish between smaller loads [11], [12].



(a) Microwave **(b)** Air conditioner **(c)** Washing machine

Figure 2.5: Binary V-I trajectory image of different appliances using data from plug load appliance identification dataset (PLAID). Adapted from [13]

Finally, voltage noise is used to extract the different appliances' electromagnetic inference (EMI) signatures. These signatures can easily distinguish appliances equipped with a Switch Mode Power Supply (SMPS) or based on motors, with the disadvantage of being sensitive to the households wiring architecture and not every appliance containing an SMPS or a motor [11], [12].

2.3.3 Transient-state features

When an appliance turns on or off, it produces a transient signature, which is a short-term fluctuation in power before settling into a steady-state [1]. To capture them, we need high-frequency sampling, above 1000 Hz, as these last just a few milliseconds. However, they can uniquely identify appliances' state transition by extracting features like shape, size, duration, and harmonics of the transient waveforms [11]. They are divided into transient power, start-up current transients, and voltage noise depending on the features studied from the transient states.

The transient power studies changes in power consumption during the transient state through the analysis of the energy consumed during the transient state [14] or the waveform spectral envelope based on Short-Time Fourier Transform (STFT) [15] or Wavelet Transforms [11], even distinguishing appliances with similar power drawn. The Start-Up current transients are extracted using only the current, obtaining the current spikes and duration, the shape of switching transients, and the transient response time. These features are sensitive to the wiring architecture and have poor performance in detecting simultaneous appliance activation/ deactivation. Lastly, the switch of electrical loads produces broadband electrical noise that is either continuous, as we have seen in the steady-state features, or transient that we are addressing now. Researchers extracted these features through Fast Fourier Transform (FFT) analysis [16], [12], obtaining similar results to the ones in the steady-state features, correctly identifying appliances with SMPS.

2.3.4 Non-traditional features

It is easy to think that the only features we can use in NILM are the electrical readings obtained by the smart meter. Any other sensor or sensing structure would bring up the cost and invalidate the NILM's proposal in the first place. Even so, there are features we can extract without needing any other sensor.

These are called non-traditional features and include the time of day, on and off duration distribution, the correlation between the usage of different appliances, and outside temperature [11]. Using them increases the feature plane with valuable information. For example, many appliances present seasonal usages, such as the Air Conditioner (AC) used mainly during summer or the heater used during the winter. The AC and heater are also influenced by the outside temperature, as the colder/hotter it is, the greater the usage of the heater/AC will be. Finally, other appliances present weekly or daily usage patterns, such as the TV or microwave.

These features help the algorithm narrow the range of appliances generating the loads by analyzing factors besides the appliance characteristics, such as the owner's typical usage or the outside temperature.

2.4 Sampling rate

While we were talking about the features, we mentioned several times low/high-frequency sampling. What we were referring to was the sampling rate of the smart meter.

The distinction between sampling frequencies is necessary as different frequencies provide access to different features [7] and require different hardware, with varying costs associated.

2.4.1 Low-frequency sampling

Most smart meters tend to report information at a very low rate, and as we explained before, the viable NILM solution must use them to keep the implementation cost low. This constraint directs much of the research towards low-frequency sampling, which contains frequencies equal to or below 1 Hz.

This sampling frequency can only extract steady-state features, and given the Nyquist-Shannon theorem, does not allow the analysis of the signal in terms of frequency.

The Nyquist-Shannon theorem states that to capture the highest frequency component of interest in a signal, one must sample at more than twice that frequency. With a fundamental frequency of 60 Hz, we needed a 120 Hz sampling frequency or above to do frequency analysis [7].

If we wanted to analyze the signal in the time domain, we would need to sample it at more than five times the fundamental frequency: a rule of thumb to correctly reproduce any waveform [7].

Given these restrictions, we can only use power changes when using low-frequency sampling. To overcome the limitations of the power changes, researchers increased the number of features recorded, as we mentioned when talking about these features, and combined them with non-traditional features.

To obtain the features in this category, such as active power, cumulative energy, or power factor, the smart meters use custom algorithms to extract features from the high-sampling frequency data and produce a down-sampled summary [7].

2.4.2 High-frequency sampling

When using low-frequency sampling, we cannot investigate some steady-state and transient-state features. To do so, we need high-frequency sampling that groups all the frequencies above 1000 Hz.

The motivation to use such high sampling is the benefit we get in terms of features as they provide the most distinct appliance signatures, even distinguishing appliances of the same model.

One example of such features is the signal's harmonics, which allows us to analyze the signal in the frequency domain. The harmonics are higher frequency components, multiples of the fundamental frequency, introduced by non-linear loads such as power supplies in the current waveform [7]. Figure 2.6 presents the effect of harmonics created by different appliances in the current waveform.

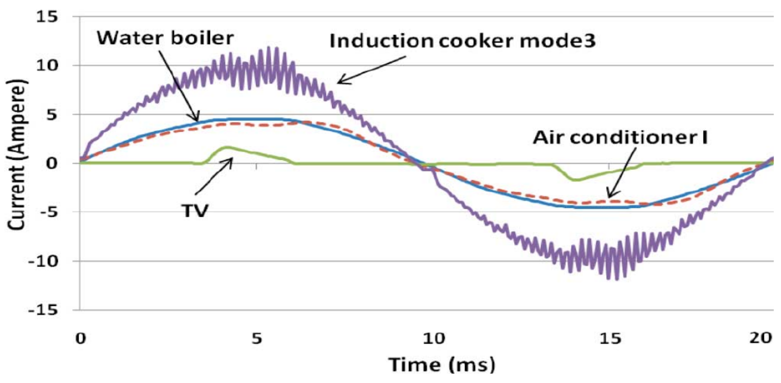


Figure 2.6: Effect of higher frequency components in the current waveform (adapted from [9]).

Although it has excellent benefits, it also has a significant cost associated since we need specialized and expensive hardware to sample at such frequencies in a smart meter. Besides the expensive hardware, the remaining infrastructure responsible for storing and processing the data is also more expensive as the bandwidth, processing power, and storage, for example, significantly increase.

2.5 Event detection

Depending on the features or model type used, there might be interest in different events in the data.

For example, when using transient features, we are only interested in extracting features from the transient states. Even so, the high-frequency meter collects the values at every timestep. To identify the occurrence of the state transitions in the signal and process only the corresponding period, researchers use the event-based approach.

On the other hand, the event-based approach does not make sense when using a Hidden Markov Model (HMM), which needs to see every step of data to update its internal states. The solution was the proposal of the state-based approach, which allows HMMs to update their states.

Finally, a new approach emerged with the popularization of ML and Deep Neural Networks (DNNs), the eventless approach, which obtains the energy consumed by all the appliances in each timestep regardless of any event using a sliding window approach.

2.5.1 Event-based approach

In order to obtain the energy consumed by an appliance, some NILM models need to know when a state transition occurs.

The task of state transition detection or event detection is not easy as different appliance types have different operational states to detect, and the state transitions can overlap. To find the events, the algorithm focuses on the edges generated by state transitions in the power consumption. After detecting the edges, a change detection algorithm finds the beginning and end of an event, matching the increasing edges with the decreasing edges [1].

Knowing the beginning and end of an event, we can extract the features supplied by the meter from that given period and use them to obtain the responsible appliance and the energy consumed. In Figure 2.4, we see the different steady-states found in the aggregated data and the identification of two edges that generate one of them.

The main disadvantage of these approaches is the need for a detection algorithm. Even if the disaggregation algorithm performs well, the system is limited by the detection algorithm's performance, where significant measurement noise can lead to false detection, or a high detection threshold can lead to miss detection. Making the overall system have poor performance.

2.5.2 State-based approach

As we mentioned before, not all models fit in the event-based approach. Models like HMMs and their variants, for example, cannot use it. Instead, they need to represent each appliance as a state machine with distinct state transitions based on the appliance usage pattern [1]. Since different states represent different consumption edges, we can map the probability of an edge corresponding to a state transition of a given appliance.

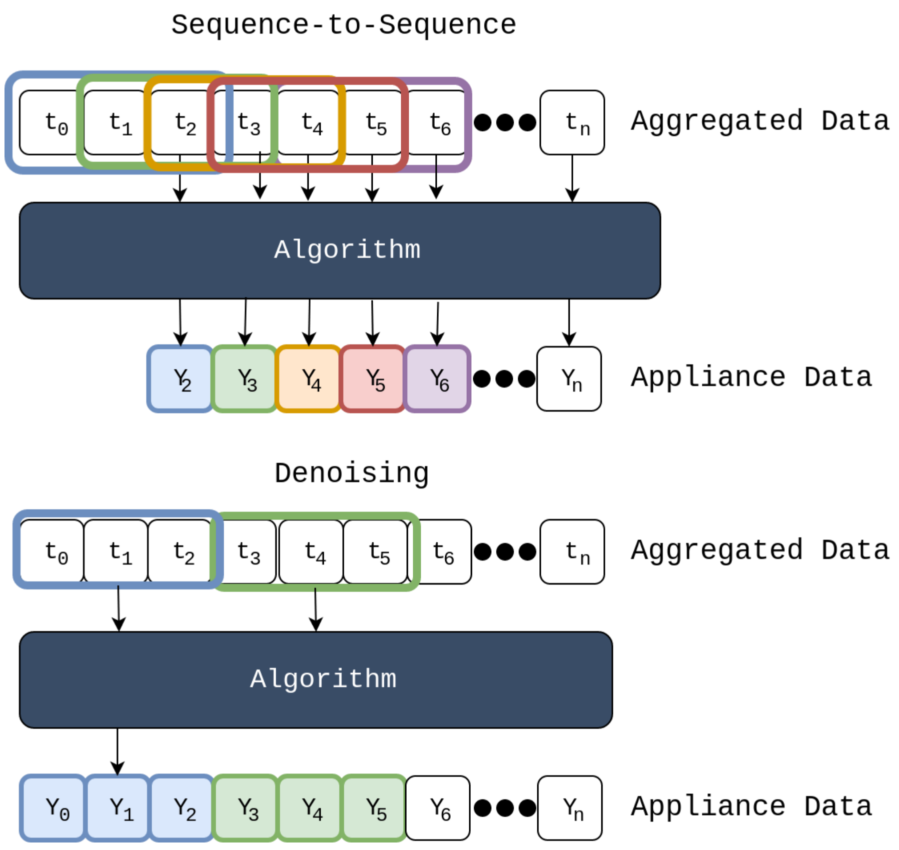
The main disadvantages of these approaches are the need for expert knowledge to create each appliance's initial probabilities and states and the inability to represent continuously variable appliances correctly.

2.5.3 Eventless approach

Recently ML models started applying a new way to disaggregate energy by considering NILM a time series [17], denoising [17], sequence-to-sequence[18], or sequence-to-point [18] task.

In these tasks, the algorithm analyzes each signal sample through a sliding window instead of examining the appliances' activations. For example, sequence-to-point or time series use a stride of one, using the whole window to obtain the energy consumed at a single timestep. In contrast, the denoising or sequence-to-sequence can use a stride equal to the size of the window as they predict the energy consumed at every timestep in the input time window. An example of these approaches is presented in Figure 2.7, where the time windows and predictions made for each one are color-coded.

Even though it can be computationally expensive since the algorithm analyzes each sample, SOA models use this approach.



Sequence-to-Sequence
t_0 t_1 t_2 t_3 t_4 t_5 t_6 ●●● t_n
Aggregated Data
Algorithm
Y_2 Y_3 Y_4 Y_5 Y_6 ●●● Y_n
Appliance Data
Denoising
t_0 t_1 t_2 t_3 t_4 t_5 t_6 ●●● t_n
Aggregated Data
Algorithm
Y_0 Y_1 Y_2 Y_3 Y_4 Y_5 Y_6 ●●● Y_n
Appliance Data

Figure 2.7: Different types of eventless approaches using different window strides.

2.6 NILM MODELS

Now that we understand the basis of NILM and what we can use to solve it, let us look at the types of models adopted. Hart proposed the first model in 1992 [19]. The model used edge detection to detect appliance activations and extracted from them the duration of the activation and the change in reactive and active power. Then using clustering, it grouped the appliances' signatures. After the training, the clusters were manually labeled, and when a new event was detected, it would fit a cluster and be classified as its appliance activation.

Nowadays, there are a variety of proposed models. The main approaches consider ML models, probabilistic models, Graph Signal Processing (GSP) models, and optimization-based models [11].

2.6.1 Ideal NILM model

Before we dive into the proposed models, we must understand the characteristics needed for a viable NILM model ready for production as it should respect some restrictions.

As presented in [20]:

- The signatures of appliances may only be based on the active power sampled at 1 Hz since most of the current smart meters are not all able to produce higher frequency measurements;

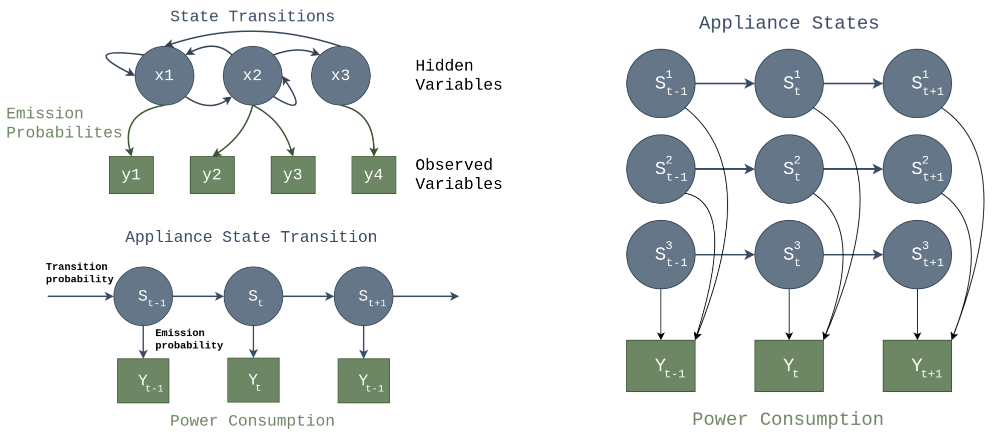
- The minimum acceptable accuracy of the NILM algorithm should be 80%, and the algorithm should perform in real-time;
- The method should be scalable in the sense of robustness, and the number of appliances tracked efficiently should be up to 20,
- The algorithm should work with various types of appliances and without prior training.

Depending on the objectives, some requirements can be alleviated, as proposing a NILM solution respecting all of them is an arduous task.

2.6.2 Probabilistic models

In their majority HMMs and variants, the probabilistic models represent the appliance based on its predefined states and state transition probabilities [20]. Besides HMMs, multiple Naive Bayes models were also proposed. To better understand their limitations, let us take a deeper look at the HMMs.

An HMM is a Markov model whose states are not directly observed; instead, each state is characterized by a probability distribution function modeling the observation corresponding to that state. There are two types of variables in HMMs: observed and hidden variables, where the sequences of hidden variables form a Markov process. In the context of NILM, the hidden variables are used to model appliances states (ON, OFF, standby ...) of individual appliances, and the observed variables are used to model electricity usage [1]. The number of hidden variables does not need to be equal to the appliance number of states as it might be easier for the model to create a transition hidden variable between two states. Instead, the number of states is the lower bound of the number of hidden variables. In Figure 2.8a, we see the schematic representation of an HMM. The model has three hidden states (x_1, x_2, x_3) and four observable variables (y_1, y_2, y_3, y_4). Besides the model, the figure also contains the time series process wherein each timestep t the model predicts a state s_t of the appliance and the corresponding power consumption.



(a) Hidden Markov Model **(b)** Factorial Hidden Markov Model

Figure 2.8: Hidden Markov Model and Factorial Hidden Markov Model schematic representations

One of the most frequent variants of HMM used is the Factorial Hidden Markov Model (FHMM), as they are preferred when modeling time series generated by the interaction of several independent processes. Figure 2.8b presents the graphical representation of an FHMM where we have multiple independent hidden state sequences, instead of a single one, representing the independent processes and where the observations depend on the hidden state of each process on that timestep [1]. In NILM, the multiple appliance loads are the independent hidden state sequences, and the observations are the energy they consumed. This means that to use FHMM, we assume that each load can be represented as one HMM [21].

Although widely researched, both HMMs and FHMMs have some disadvantages [1], [21]:

- They require expert knowledge about the appliances to define the number of states and the base state transition probabilities of the model;
- They cannot model continuously variable appliances and permanent consumption appliances;
- They present exponential growth linked to the number of appliances being disaggregated;
- They need model adaptation to apply a model across different houses;
- Even though FHMMs obtain better results than traditional HMMs, the computational complexity of both learning and inference is greater, and their inference techniques are highly susceptible to local optima.

The limitations mentioned above and the results obtained by the recent machine learning models deemed them unviable solutions. In any case, for an in-depth review of HMMs and variants in NILM, please refer to [1] and [21].

2.6.3 Graph signal processing models

Unlike the remaining approaches to NILM, GSP does not need feature extraction or event detection to perform disaggregation. Instead, GSP focuses on load waveform analysis using a graph [20].

GSP represents a dataset using a graph signal defined by a set of nodes and a weighted adjacency matrix. Each node in the graph corresponds to an element in the dataset. At the same time, the adjacency matrix defines all directed edges in the graph and their weights, where assigned weights reflect the degree of similarity or correlation between the nodes [1].

These models are suitable for data classification problems in which training periods are short and inefficient to build appropriate class models [1]. These characteristics make GSP ideal for NILM, as algorithms tend to overfit specific appliance models because they do not have enough training data. Even so, the proposed models to date need a few constraints to obtain good results.

That said, further investigation is needed to validate the viability of GSP models, as there are only a few proposed models.

2.6.4 Optimization-based models

If we see NILM as a composite disaggregation problem and we have a database of appliance signatures, we can use optimization-based models to solve it [20].

The optimization-based approach tries to match the signature recorded from an event detection algorithm with a combination of know signatures stored in a database. The combination of appliances chosen is the one that minimizes the error between the generated combination and the aggregated signature [22].

Even though different optimization approaches were proposed, such as integer programming and genetic algorithms, they had some common disadvantages [11]:

- They need to have an appliance signature recorded for each appliance type in a given household;
- They are computationally expensive;
- They are not able to discern appliances with similar load signatures.

The disadvantages presented deem the models unfeasible for a wide-scale deployment as we cannot record a signature for every single appliance in a household, and at the same time, deal with the other two disadvantages given the associated costs.

2.6.5 Machine learning models

In the past decade, machine learning has made substantial improvements in several fields. If we talk about deep learning, the improvements are even more evident, as it has become SOA in areas such as computer vision, speech recognition, and machine translation [1].

The results obtained in those areas motivated researchers to create machine learning models for NILM. Nowadays, a wide variety of models are proposed, both supervised and unsupervised, ranging from simple Decision Trees or K-means to deep learning models such as CNNs.

Supervised

The supervised models are the most popular approach for ML models and can either be event-based or eventless. If they are eventless, they can be sequence-to-sequence, sequence-to-point, or denoising approaches.

The first models in this category were K-Nearest Neighbours (K-NNs) and Support Vector Machines (SVMs). Due to the simplicity of the models, they did not obtain good results when using low-frequency sampling. On the other hand, when using high-frequency sampling, the results were good, getting close to the ones obtained using variations of HMMs.

Later, seeing the success of deep learning in other areas, Recurrent Neural Networks (RNNs), Denoising AutoEncoders (dAEs), and CNNs were also proposed to solve NILM, suppressing the SOA in both the low and high-frequency settings.

These three types of networks differ on the type of nodes they use.

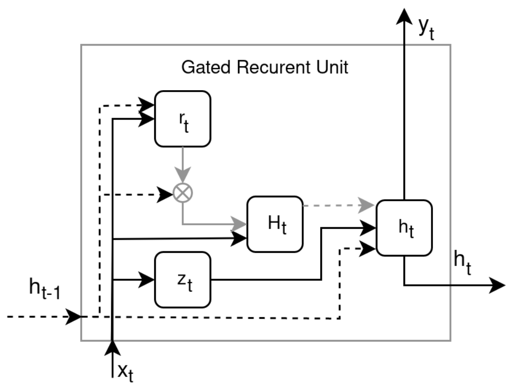
The RNNs use a particular type of neuron that considers the previous neuron outputs when obtaining the current output. This is valuable in NILM, given the temporal information of the appliances. For example, if the fridge was just turned on in the previous timestep (an activation occurred), it is most likely that the fridge is still turned on in the next timestep. On the other hand, if the fridge activation occurred more than five minutes ago, the fridge will likely turn on again soon, so the subsequent activation with a similar power draw is easier to identify as the fridge just on a probability basis.

The models proposed in NILM use either GRU or Long Short-Term Memory (LSTM) neurons to avoid the vanishing and exploding gradient problem of the first recurrent units created, allowing the analysis of longer input sequences. In Figure 2.9, the detailed structure of both types of neurons is exhibited.

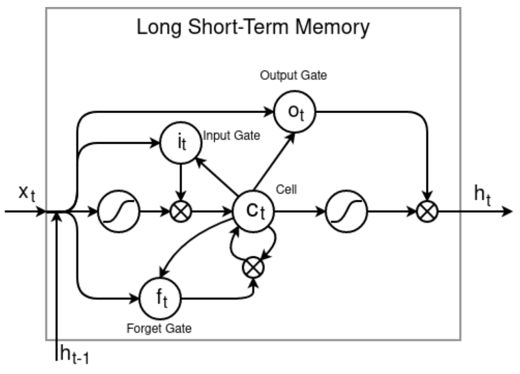
The LSTM key is the cell, C_t, which contains the information regarding the previous examples. When a new example needs to be processed by an LSTM, the first step is to decide what information the neuron will throw away. This is controlled by the forget gate, f_t, which combines the new example, x_t, and the previous output, h_{t-1}, to obtain a value between zero and one where one means forget nothing and zero means forget everything. After that, the neuron chooses the new information to add through the input gate, i_t, and the *tahn* function. The first decides which values we are adding, and the second obtains the actual values to add. Having all this calculated is time to update the cell state by forgetting the information defined by the forget gate and adding the information defined by the input gate. With the cell updated, we are ready to generate the output regarding the new example. To do so, the output gate, o_t, is applied to obtain the information we will use regarding the new example. The output gate's result is then multiplied by the *tahn* of the cell state to generate the neuron's new output.

Regarding the GRU layer, the first step is to calculate the update gate, z_t, which determines the inpact of past information in the to new example. After that, the neuron calculates the reset gate, r_t, which indicates how much information the neuron shall forget. With both of these gates calculated, the neuron updates the current memory content, H_t, by using the output of the reset gate and the previous neuron output, h_{t-1}, to remove the information that needed to be forgotten and the new value, x_t, to the add new information. At this point, the only thing left is to calculate h_t using the z_t and h_{t-1} to decide the relevance of the previous examples and $(1 - z_t)$ and H_t to decide the new example's relevance.

When analyzing the neurons' architectures, their similarity is apparent, and unfortunately, their performance depends on the task they solve. This means that to understand the better option in most problems, we need to implement both types of layers to understand which is better.



(a) Gated Recurent Unit



(b) Long Short-Term Memory

Figure 2.9: Gated Recurrent Unit and Long Short-Term Memory schematic representations.

As for the CNNs, they were first introduced to do image processing. They achieve this using two new types of neurons: the convolutional and pooling neurons. The convolutional layers act as filters, sliding through the image extracting relevant features such as vertical or horizontal lines. The filters, also known as kernels, are $m \times m$ matrixes of weights, where m is usually an odd number. Using Figure 2.10 as a reference, the kernel is applied to a section of the image, the light blue highlight, also known as receptive field, to extract relevant features by calculating the dot-product between the section pixels and the filter weights. After calculating the dot product, in this case, -3, we store it in the output matrix, and the kernel shifts a given stride value.

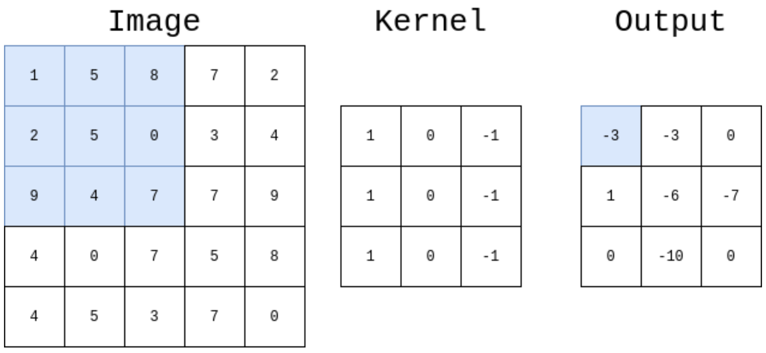


Figure 2.10: Application of a convolutional neuron with a 3×3 kernel with stride 1 to an 5×5 image.

The pooling layers are very similar to the convolutional layers, but they do not have filter weights. Instead of calculating the dot product between the filter values and the pixels, the filter uses an aggregation function. The most common aggregation functions are the max or average pooling, where we obtain the receptive field's maximum or average value, respectively. In Figure 2.11, we have the result of applying a max-pooling neuron with stride one.

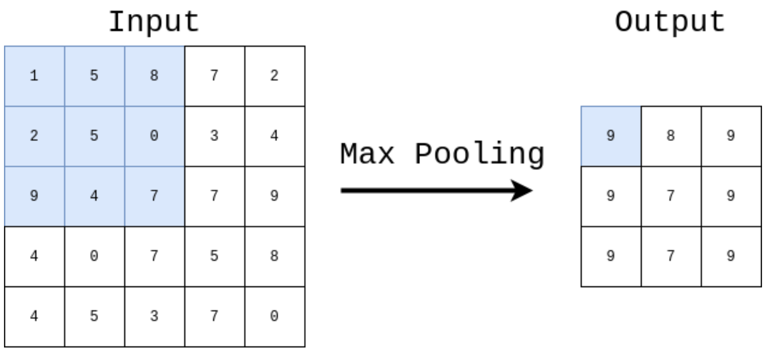


Figure 2.11: Application of a max pooling neuron with a 3×3 kernel with stride 1 to an 5×5 input.

Since a time series can be seen as a one-dimensional vector, if instead of using an $m \times m$ filter, we use an $m \times 1$, we can apply the CNNs used in image classification to time series problems, as seen in Figure 2.12. However, why would we do that? Since CNNs can learn spatially invariant features in a two-dimensional space such as an image. It is only natural to think that discovering patterns in a one-dimensional space (time) should be easier [23].

For example, suppose a kernel is responsible for detecting positive steps in power and the time series has three activations. In that case, the kernel can detect the three positive steps of the activations and preserve the order by which they were detected.

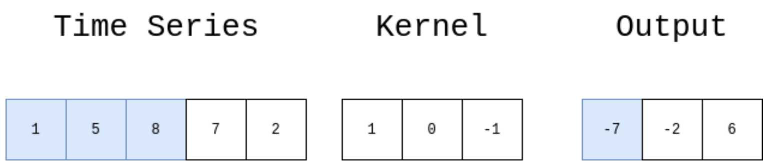


Figure 2.12: Application of a convolutional neuron with a 1×3 kernel with stride 1 to an 1×5 time series.

The main advantage of CNNs over RNNs is the independence of the input size in the features extracted. RNNs tend to forget the older inputs as they move forward, while CNNs extract the same features independently of the input size. This makes CNNs better at detecting long-term dependencies between appliances activations than RNNs.

Both the RNNs and CNNs, when applied in NILM, see it as a time series task. The dAEs treat NILM as a denoising problem [24], where the aggregated readings are treated as a single appliance consumption with noise.

The dAE reconstructs the input signal to remove the noise by dividing the network into two parts: the encode and decode. In the encoding part, the input vector is encoded into a smaller vector. This is achieved by having a layer with a smaller dimension than the input. The decoding part reconstructs the input vector without the noise. We do it by progressively increasing the number of nodes in each layer.

A visual representation of this concept is seen in Figure 2.13.

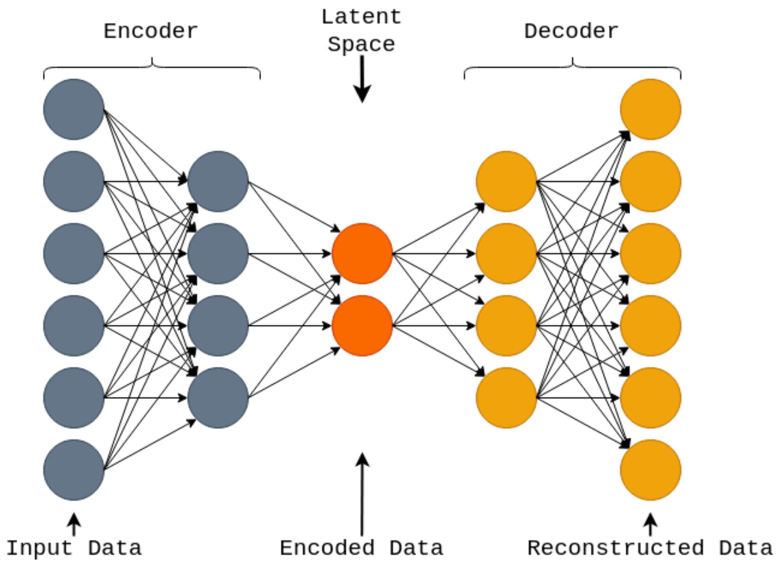


Figure 2.13: Structure of an Denoising AutoEncoder.

The dAE can be deep or shallow, and the neuron activations can either be linear or non-linear. They are the fastest networks of the three and, when combined with a convolutional layer in the input and output, achieve competitive results with the other two types of networks.

A more in-depth explanation of these different models can be found in [25] for LSTM, [26] for GRU, [24] for dAE, and [27] for CNN.

Unsupervised

Even though supervised learning achieves good results and is SOA, there is a particular interest in unsupervised alternatives mainly because the data collection for training is much cheaper.

Similar to Hart's first proposal [19], the unsupervised alternatives rely mostly on clustering algorithms, such as K-means and hierarchical clustering, that group different appliance signatures in different clusters.

The difference between clustering algorithms is how they obtain the clusters. Like most clustering algorithms, the K-means algorithm needs to have a predefined number of clusters. Once the number of clusters, k, is defined, the algorithm picks k training examples and defines them as cluster centroids, which are the average distance between all the elements in the cluster. After that, the algorithm assigns every example to the closest cluster, given a distance function like euclidean, manhattan, or Dynamic Time Warping (DTW). At this point, the cluster centroids are again calculated, and the algorithm starts a loop of reassigning examples and recalculating centroids that stops once the difference between the new and previous centroid is below a certain threshold or the loop reaches the maximum number of iterations.

As for hierarchical clustering, there are two approaches, agglomerative and divisive. The first is a bottom-up approach where we start with every example as a cluster and merge the clusters until we reach the desired number of clusters. The second does the opposite, starting from one cluster and dividing it until we reach the desired number of clusters. The algorithm uses an appropriate metric to calculate the distance between pairs of observations and a linkage criterion that specifies the difference between two sets of examples to decide how to divide or merge the clusters.

Once the clusters are obtained, they are labeled as representing one appliance, and with this structure, each following signature is identified by the appliance with the closest cluster.

However, not all approaches consist in clustering algorithms, as some recent proposals provide unsupervised alternatives based on neural networks like the Spiking Neural Network (SNN). This network uses a different type of neuron inspired in the biological brain neurons that communicate through electric spikes. The Spike Timing Dependent Plasticity (STPD) is used instead of the regular back-propagation to train this network, as it allows the update of neuron synapse without a reference output guidance. In trained SNN, a specific neuron set is excited by a corresponding type of appliance. To identify the appliance represented by each set of neurons, the user must present one labeled input of each appliance [13].

Although there is interest in having an unsupervised solution, the results show that they only have good performance when using high-frequency sampling and can only beat the supervised alternatives when considering a small dataset.

Semi-supervised

Recently, researchers in diverse areas like natural language processing started pretraining their models on a large corpus of unlabeled data to improve the results in specific tasks with small datasets. With this in mind, some researchers believe that the same could be applied in NILM by either pretraining on a large dataset of unlabelled data or pretraining on unlabelled data from the house we wish to disaggregate [17].

Using the first option would lead to a much cheaper data collection system for training, as obtaining appliance data is the most expensive part of data collection. With this setting, we could record the aggregated readings from 100 houses and the appliance readings of 10 of those instead of only recording 20 houses with the appliance data. On the other hand, pretraining on the target house would improve the system's performance without increasing the costs as the models would be trained on the target appliances, and we already had to record the aggregated data of the target house.

Given the apparent benefit of either of those options, some models using semi-supervised learning in NILM were proposed. The models used only simple algorithms such as K-NN [28], Random Forest (RF) [29], or SVM [30] that were compared with their supervised alternatives where the better results of the semi-supervised models proved the benefit of using it in NILM. Even so, the lack of comparison with SOA models or their application as semi-supervised models makes the viability of this area unclear.

2.7 Related work

The previous sections explained the features used, how they are extracted and which models were tried to solve NILM. However, it is not clear how we can combine them into a complete system. This section introduces some of the proposed models to solve NILM, explaining how they combine the different features and models and analyzing their performance.

Starting with the probabilistic models, Zhong et al. [5] implemented an Additive Factorial Hidden Markov Model (AFHMM) with signal aggregate constraints to improve the base AFHMM. The model used the active power recorded at a low rate (every 2 or 10 min) and showed an improvement of over 16% compared to the original AFHMM.

Using FHMM as a base model, H. Kim et al. [31] proposed three new models: a Factorial Hidden Semi-Markov Model (FHSMM), a Conditional Factorial Hidden Markov Model (CFHMM), and a Conditional Factorial Hidden Semi-Markov Model (CFHSMM). The first model was proposed since the ON durations of appliances are better modeled when using gamma distributions. The second one was proposed to consider additional features such as the hour of usage, day of the week, or input from other sensors. Moreover, the third and final model is a combination of the other two models. The results were obtained under low-frequency sampling (3s) in a proprietary dataset and show the benefit of adding extra features and choosing a better distribution. Given that, all the proposed models achieved better results than the standard FHMM, and the CFHSMM delivered the best results, demonstrating that the modified models can be further improved by combining strategies.

With an innovative idea, O. Parson et al. [32] implemented a Difference Hidden Markov Model (DHMM) that did not need appliance data for training. Instead, they start with generic models, capture clean appliance activations from the house aggregated data by expectation-maximization, and adapt the generic models to the specific appliances of the house. Besides that, they also proposed removing the predicted appliance power from the aggregated readings in sequence, facilitating the recognition of the lower power devices. The idea is to gradually clean the aggregated signature and allow the remaining low power signatures to be more prominent. The comparison between the models using submetered adaptation, aggregated adaptation, and no adaptation shows that the aggregated adaptation has a slight decrease in performance compared with the submetered adaptation but a significant increase compared to no adaptation, which validates this approach.

The results obtained in these papers are not directly comparable with the SOA models or between them. They use different error measurements, different datasets, or different training conditions in the same datasets. This reality is present in most papers as we can only extract conclusions from the comparisons presented in the given paper.

Looking at the GSP models, K. He et al. [33] introduced an algorithm scoring better results than decision trees and HMMs. However, the overall F1-Score obtained was below the SOA models. B. Zhang et al. [34] presented a model that improved the results obtained by the previous one achieving as good or better results depending on the appliance. These models used active power and low-sampling frequency (1 min) to achieve their results.

On the other hand, B. Zhao et al. [35] used GSP with a different purpose. Instead of developing a NILM model, they proposed two signal processing methods that improve any low-frequency supervised and unsupervised event-based NILM model's accuracy. In particular, they propose a graph-based filtering approach to clean the power signal before classification and a post-classification refinement method, which improves NILM by mitigating the effect of misclassification of loads with a similar operational range.

As for the optimization approaches, the proposed models vary a lot. For example, using models based on active power and low-frequency sampling, F. Wittmann et al. [36] propose an algorithm using Mixed-Integer Linear Programming. In contrast, R. Machlev et al. [37] introduced a modified cross-entropy method where NILM is a constrained optimization problem. The first model obtained high accuracy on its testing setup but was only compared with combinatorial optimization. The second one obtained the overall best performance compared with other optimization-based approaches, such as particle filtering and segmented integer quadratic programming with and without constraint programming.

Similar to the HMMs, the results obtained cannot be compared with the SOA ones.

To finalize the review of the proposed models, we need to talk about ML. One of the first analyses of deep learning applicability in NILM was performed by J. Kelly et al. with NeuralNILM [17]. In the paper, three DNNs are proposed: an RNN using LSTM layers, an dAE, and a network called rectangles which precited the beginning, end, and average consumption of an activation. The dAE, RNN, and rectangles networks outperformed the baseline combinatorial optimization, and FHMM models in every metric proposed, and the

RNN showed the worst performance of the three.

After this paper, other researchers proposed models that further improved the obtained results. For example, M. Kaselimi et al. [38] proposed a Context-Aware Bidirectional LSTM that improved the RNN results. And O. Krystalakos et al. [39] implemented the OnlineGRU, an RNN based on GRUs that reduces the number of trainable parameters of the initial RNN while maintaining its performance.

Meanwhile, the popularity of CNNs was increasing, and with it, some models were proposed in NILM. C. Zhang et al. [18] presented two CNNs, one solving NILM as a sequence-to-point (Seq2Point) problem and the other as a sequence-to-sequence (Seq2Seq). The overall results significantly improved the SOA performance as the Seq2point method reduces the Mean Absolute Error (MAE) by 84% compared to the dAE proposed in NeuralNILM.

Although Seq2Point obtained good results, it was not able to be implemented in embedded devices. To solve this, E. Santos et al. [40] lowered the complexity of the Seq2Point by reducing the number of convolutional layers and increasing the resolution of the filters. These modifications reduce the training and predicting time while maintaining the performance.

Even though Seq2Point obtained good results, there were problems it did not address. One of the main problems in the proposed architecture is that there is no ideal time window that achieves the best results in every appliance type. In an attempt to solve it, G. Zhou et al. [41] proposed a CNN with four different convolution blocks, each receiving a different time resolution from 150 up to 1554 seconds. This algorithm beat every model with which it was compared, although it was not compared to the SOA Seq2Point or Seq2Seq models. Similarly, W. He et al. [42] proposed a network with multiple parallel convolutions with different filter sizes at the input, followed by LSTM layers obtaining good results compared to the NeuralNILM models.

Nevertheless, the most critical problem was that Seq2Point did not obtain perfect scores. In an attempt to solve it, H. Chen et al. [43] implemented a Convolutional AutoEncoder. The model applied BatchNormalization and used the Hill-Climbing algorithm for parameter optimization achieving much better results than the RNN proposed in NeuralNILM. Similarly, T. Sirojan et al. [44] implemented a Convolutional Variational Auto Encoder that obtained slightly better results than the Seq2Point and Seq2Seq.

All of these models used low-frequency sampling and active power to solve the NILM task. Besides that, all of them were supervised models. The unsupervised models proposed rely on high-frequency sampling to obtain good results.

For example, Y. Quek [45] used K-means to group raw steady-state waveforms into several distinct clusters and K-NN to classify each cluster as an appliance type. Simultaneously, G. Jacobs et al. [46] used the DBSCAN algorithm to cluster ON/OFF pairs of appliance activations. This approach obtained good results, identifying 13 out of 17 appliances with a precision greater than 90%. The main disadvantage of this approach is the manual labeling of each cluster in the house. Furthermore, H. Lam et al. proved that the V-I trajectory was a good feature for characterizing the typical household appliances using hierarchical clustering [47].

In these approaches, some clusters end up grouping multiple appliances as their load signatures are very similar. To reduce the overlapping load signatures effect on the model's performance M. Aiad and P. Lee [48] provide an additional step intended to split the clusters when this happens. The extra step improved the clustering algorithms' performance when the original cluster was loose, and the resulting clusters had a noticeable distance between their centers.

However, not all unsupervised machine learning models are based on clustering algorithms. For example, Z. Zhou [13] implemented an SNN that showed better results than the supervised approach when the models had only a few training examples.

One other model worth mentioning that does not fit correctly in any of these categories is the Deep Neural Network-Hidden Markov Model provided by L. Mauch and B. Yang [49]. The model uses an HMM to disaggregate the appliances with bigger consumptions and an dAE that removes the smaller appliances signatures from the aggregated consumption, considered noise in this implementation. This model is worth mentioning since combining different models to improve the overall performance might be a good solution as different models tend to do better in different appliances. In other time-series tasks, different models have been combined in techniques such as ensemble classifiers or meta classifiers and proved effective. Using ensemble classifiers such as the Hierarchical Vote Collective of Transformation-Based Ensembles (HIVE-COTE), which uses the output of 37 classifiers to create a hierarchical voting system, obtains SOA results [50] in the UCR/UEA time-series classification archive. Even so, in NILM, these techniques have yet to be explored.

2.8 Datasets

Regardless of the algorithm we chose to implement, one thing is granted: we will need data to train it. The data we use to train the algorithms plays a significant role in the quality of the model, so much so that one usual saying in ML is: "The model is as good as the data in which it was trained".

Understanding the available datasets, what they offer, and the essential characteristics for a good dataset are vital steps for any model implementation. In NILM, the datasets are composed of electrical readings representing either the aggregated consumption of the house, the appliance-level consumption, or both. This seems simple enough, but in reality, we need to care about the frequency at which the data was recorded, the electrical readings collected, the type of appliances recorded, the number of houses considered, the recording period, and the country in which the dataset was collected.

2.8.1 Appliances

Besides the number of appliances recorded in a given dataset, it is vital to understand what type of appliances were recorded. Even though we mentioned that each appliance has its distinct load signature, the loads can be grouped by their usual form. A good algorithm shall be capable of disaggregating the appliances from every appliance group, and the only way to test it is through a dataset containing all of them.

According to the states of an appliance, we can divide them into four categories:

Type I appliances

The type I appliances are the simpler devices as they only have two states, ON or OFF. Examples of these devices are toasters, table lamps, and kettles.

Type II appliances

These devices are a bit more complex as they contain multiple states of stable consumption. Since they have a finite number of states, they are seen as Finite State Machines. Examples of these appliances are fridges, electric fans, and hair dryers.

Type III appliances

On the third group of appliances, we find the hardest ones for the models, the continuously variable devices. They are similar to type II appliances, as they have multiple states. However, they have an infinite number of states, which makes them much harder to disaggregate. Examples of these appliances are drills, dimmable lights, and induction motors.

Type IV appliances

Lastly, we have constant consumption devices. These devices have long periods of stable consumption (weeks or even months). Some of these devices can be tricky for the models as their readings can be mistaken by noise. Examples of these devices are routers, smoke detectors, and home alarms.

2.8.2 Electrical readings

The electrical readings are one of the most critical characteristics of a dataset. In Section 2.3, we explain the different features we could extract and their importance. Nevertheless, when choosing a dataset, we need to remember a couple of things that we did not mention before.

Suppose the readings are recorded with low-frequency sampling. In that case, we are limited to power change analysis of the low-frequency readings, and we cannot extract new features from the ones supplied. Besides that, when choosing a dataset, it is essential to understand the recording setup since the metering device used influences the recorded signatures. With inconsistent measurements, the generated signatures will not only be characteristic of the appliance but also the metering technology, making the generalization much more difficult. Finally, we should also keep in mind how the data was stored as it might or not be preprocessed to smooth reading errors or fill missing values.

These characteristics directly influence the model's performance and are sometimes overlooked as they are not mentioned in most dataset reviews and are not apparent when we use the dataset.

2.8.3 Number of houses

When we say that each appliance has its load signature, we are including even appliances from the same brand and model. The differences might not be as evident, but they do exist.

To avoid the model overfitting a specific appliance model, we need various appliances from the same type, and the easiest way to do so is by recording multiple houses.

Besides helping models generalize to different appliance brands and models, recording multiple houses will help the models generalize to different appliance usage patterns. For example, a family household with two infants and two adults will most likely use the washing machine more frequently than a single person's household.

2.8.4 Recording period

As we mentioned previously, the appliance usage pattern changes from home to home. To identify the appliance we can use other features such as the moment of usage. Since most appliances have seasonal patterns, weekly patterns, or daily patterns as shown in Figure 2.14, we need an extended recording period greater than a year.

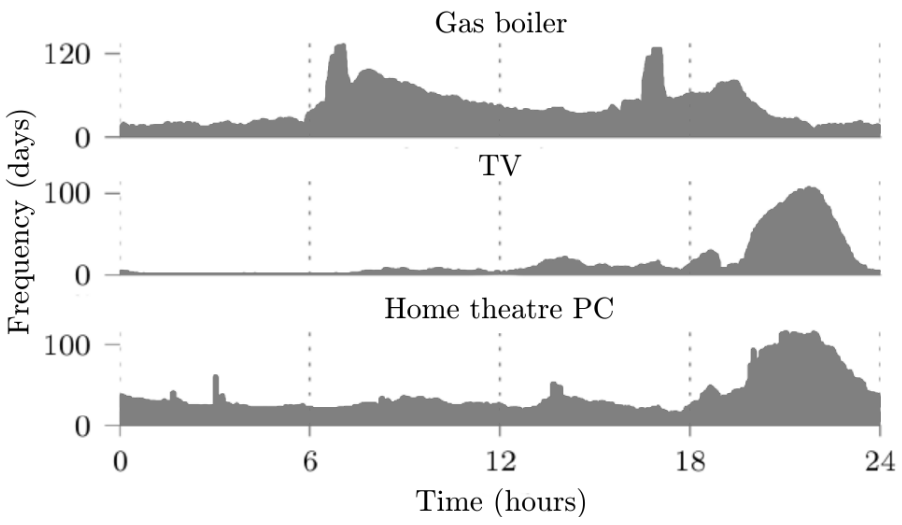


Figure 2.14: Daily appliance usage histograms of three appliances over 120 days from UK-DALE house 1 (adapted from [3]).

2.8.5 Recorded country

No matter how small it is, each country has its own culture, reflecting on the appliance sets used and their usage patterns. This means that in order to deploy a model on a specific country, we should have a dataset of that country to evaluate its performance [8].

2.8.6 Publicly available datasets

Publicly available datasets help research fields improve at a much faster pace. Image classification and natural language processing are two examples of fields that saw a notary improvement in performance once there were large publicly available datasets, and NILM is no exception.

After the analysis of the essential dataset characteristics, we can evaluate the publicly available datasets against them. Table 2.1 contains the publicly available datasets, and unfortunately, none of them fulfill all the characteristics we mentioned before, usually laking one or more.

Starting from the recorded country, we only have one Portuguese dataset SustData[51], recorded in Madeira with low-sampling features. This dataset is suboptimal as it only contains the aggregated readings. Analyzing the remaining characteristics REED[52], BLUED[53], Smart*[54], and IAWE[55] datasets present a short recording period, where sometimes the less used appliances do not have enough activations to train the models. The DRED[56] and ECO[57] datasets have a longer recording period, although insufficient, and only record active and reactive power. The AMPDs2[58] has plenty of features recorded but a low sampling rate of 1 min and from only one house. The COMBED[59] dataset records college buildings, meaning it does not contain many typical household appliances. The Dataport[60] disadvantage is the very low-frequency sampling of 1 min. The UKDALE's [8] and REFIT's[61] disadvantage is only recording the a few electrical features.

Given that all the datasets lack one or more characteristics to be considered a complete NILM dataset, most researchers opt for the one that satisfies most of their needs. With no consensus on what dataset to use, comparing models' performance becomes more arduous, stalling the progression in the field.

Dataset	Location	Duration	Nº Houses	Sensors per House	Resolution	Features
REED[52]	USA	3-19 days	6	24	120 kHz(Aggr) 0.5 and 1 Hz(sub)	V and P(Aggr) P(sub)
BLUED[53]	USA	8 days	1	Aggr	12 kHz	I, V and State transition label
UK-DALE[8]	UK	4.3, 3 or 1 years	5	5-54	16 kHz(Aggr) 1, $\frac{1}{6}$Hz(sub)	P and switch status
AMPDs2[58]	Canada	2 years	1	21	1 min	V, I, F, P, Q, S, Pf, real, reactive and apparent energy
Combed[59]	India	1 year	6	200	1 Hz	P, I and real energy
DRED[56]	Netherlands	6 months	3	12	1 Hz	P
ECO[57]	Switzerland	8 months	6		1 Hz	P and Q
GREEND[62]	Austria / Italy	1 year	9	no Aggr 9	1 Hz	P
iAWE[55]	India	73 days	10	33	1 s(Aggr) 1 or 6 s	V, I, F, P and phase
REFIT[61]	UK	2 years	20	11	8 s	P
Smart[54]	USA	3 months	3	21-26 circuit meters	1 Hz	P and S(Aggr), P(sub)
Dataport[60]	USA	up to 3.25 years	722	8 - 23	1 min	P
SustData[51]	Portugal	51 - 511 days	50	Aggr	1 min	P, Q, S
DEDDIAG[63]	Germany	up to 3.5 years	15	1-5	1 Hz	P

Table 2.1: Publicly Available Datasets. P(Active power), Aggr(Aggregate), Q(Reactive Power), Sub(Sub-metering), S(Apparent power), V(Voltage), F(Frequency) I(Current), Pf(Power phase)

2.9 Implementation Tools

The use of publicly available libraries for software development has become a standardized practice. It speeds the development process and reduces the number of bugs in the code, given that those libraries are usually actively maintained and updated. Besides using libraries, the available hardware can significantly speed up the development process, especially for ML tasks, as they tend to be computationally expensive.

Since the code developed during this dissertation will be used in future development by Withus, it was decided to be kept private. To help the reader understand the specific tools used and how our work was developed, this section will briefly present the multiple libraries used and the hardware where the experiments ran to understand the problems we faced during development.

2.9.1 Software

To gather the tools we have at our disposal, we searched for those that could help us with the NILM and time series problems, ML models' development, and data processing. We opted for Python [64] as a programming language, NILMTK as the general toolkit for developing our system, and TensorFlow [65] for building our ML models.

Python

The Python programming language is one of the first options when considering ML and data processing. Besides its easy syntax, it has a vast set of libraries designed to aid those two tasks.

Starting with data processing, libraries such as Pandas [66], NumPy [67], and Matplotlib [68] are essential tools. Pandas was created to provide high-level abstractions for fast and flexible data manipulation through data structures designed to make working with *relational* or *labeled* data easy and intuitive. NumPy provides powerful N-dimensional arrays and multiple numerical computing tools for data processing. Combining them with Matplotlib, a library for creating data visualizations, allows us to process the data and visualize its results.

Looking at ML, the Scikit-Learn [69], TensorFlow, and PyTorch [70] are some of the most famous libraries. Scikit-learn features various off-the-shelf classification, regression, and clustering algorithms, but it does not cover neural networks. Those algorithms can be developed using TensorFlow or Pytorch, two libraries designed for the training and inference of deep neural networks.

Combining all these libraries gives us a massive toolset that accelerates any ML/data processing project. However, that is not the only reason why we chose Python as the programming language. Even though these libraries are robust and have thorough, easy-to-understand documentation with many examples, none deals specifically with NILM. To deal with the NILM models' needs, we use the NILMTK library. The NILMTK library dramatically reduces the learning curve for NILM beginners and is the only toolkit to our knowledge in this area.

Considering all these factors, by using Python, we will have a flexible and straightforward language with data processing functions to treat the data fed to the models, a simple API to create ML models, and a complete toolkit developed to deal with NILM datasets and models.

NILMTK

One of the most significant setbacks in NILM, as we have seen in Section 2.7, is the lack of reproducible and comparable results between models. Without comparable results, it is unclear what the most viable option is, so the progress slows down.

The Non-Intrusive Load Monitoring Toolkit, proposed by N. Batra et al. [3], was created to address just that by providing a standard data format where distinct datasets and models can be used seamlessly. The proposal was good, but the toolkit was hard to use, as the workflow and structure of the models were confusing. This resulted in papers such as [17], [35], [42], [44], [59], [71], using the toolkit only as a dataset processing tool or for the benchmarking models it contained.

To solve this problem, N. Batra et al. [72] expanded NILMTK adding a high-level API that allowed easily reproducible experiments and a set of SOA models to enable meaningful comparisons. The benefits of this change can be seen in recent papers [40], [73], where the API and the models are used.

Nowadays, the toolkit is easier to use, although the documentation is not as thorough as the other libraries we mentioned and some of the functionalities are not explicit. It contains dataset converters for the most popular datasets in NILM, and new SOA models are frequently added in a separate library called NILMTK-Contrib [74].

Tensorflow

When we talked about creating neural network models, we had two options: we could either use TensorFlow or use PyTorch. We decided to use TensorFlow mainly because the implemented models in NILMTK-Contrib used it, and learning both the TensorFlow and PyTorch APIs would be unnecessary. Another reason is that most SOA models in NILM are also implemented with TensorFlow such as [18], [40], [42], [44], [75].

These models used the Keras API, which allows us to create complex models through simple building blocks, entirely abstracting the developer from the complexity behind them. In Code Block 1, we present the code to create a simple neural network with dense layers using the Keras API and, Figure 2.15 presents the visual representation of the same network. Besides simplifying the process of creating a Neural Network (NN), the API also simplifies the training and testing of models allowing us to use different pre-built trainers, evaluation metrics, and loss functions of the TensorFlow.

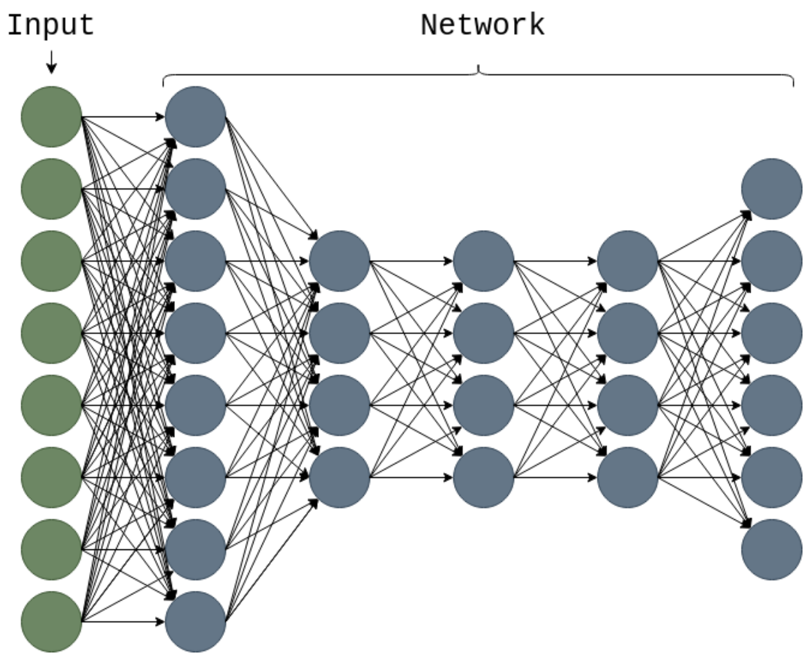


Figure 2.15: Simple Neural network representation.

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, InputLayer

model = Sequential()
model.add(InputLayer((8,1)))
model.add(Dense(8))
model.add(Dense(4))
model.add(Dense(4))
model.add(Dense(4))
model.add(Dense(6))
model.compile(loss='mean_squared_error',
        metrics=["MeanAbsoluteError", "RootMeanSquaredError"],
        optimizer='adam')
```

Code Block 1: Code for creating a simple neural network.

Other libraries

As we mentioned before, Python has a vast set of libraries, and during the construction of our models, we found two other libraries that eased the development. Those libraries are Traces, which helps us analyze time series with missing values, and PyWavelets, which implements the DWTs we used.

One of the main problems when dealing with time series is missing values, and NILM is no exception. To solve this problem, we tried different approaches, and Traces [76] is one of them.

This library proposes to solve this problem by seamlessly interpolating the missing values. For example, if we were using a 1Hz frequency sampling and only had the values for 12:30:15 and 12:30:17 when we queried the value for 12:30:16, the library would interpolate the value between the other two dates and return the result as if the value was there.

As we will see in Section 3.3.1, we implement a model using DWTs as features. To obtain the DWTs from the input signal, we used the PyWavelets [77] library. This library provides a fast high-level API to extract continuous and discrete wavelets, with more than 100 built-in wavelet filters.

2.9.2 Hardware

The hardware plays an essential role in developing a NILM model, not because it shortens the model's development time but because it shortens the time it takes to evaluate it. Given that our models went through multiple iterations, the time spent evaluating them plays a crucial role in how much work we could develop.

The experiments ran on a virtual machine with 32 GBs of ram, 24 vCPUs, and an RTX 2080. Having only one graphics card meant that we were somewhat limited in the resources for the experiments. So we limited the amount of data we fed to the models and the appliances we would disaggregate.

Even so, if we did not have a graphics card to do the models' training, it would take much longer as the graphics cards are much better than CPUs at parallel processing and each layer of a network can be paralleled computed. Given that the layers usually have more than 100 nodes, even with 24 vCPUs, we cannot calculate every layer's node simultaneously. While the RTX 2080, with 2944 cores, can compute all the nodes simultaneously.

To have a time comparison, when using a GPU, the network with the longest training period takes 146 seconds per epoch, while the CPU takes 1 hour. This means that the GPU model will take less than 3 days for the entire experiment with 5 appliances and 300 training epochs per appliance, and the one using the CPU will take 62 days.

The time reduction achieved by the GPU is remarkable, although having to wait 4/5 days to obtain the results of the complete experiment for all the used models slowed the development process, which is noticeable in the number of architectures experimented.

CHAPTER 3

Development

When reviewing the SOA in Chapter 2, we presented many different research directions we could take. To understand the ones we took, we will start by defining the project outline. After that, we will discuss both the dataset and models' development.

With all the information gathered in the background, there was only one thing left to define before starting the model development, the project outline. Given that we were at the beginning of the project, there was no clear vision regarding the models and dataset characteristics, so we had to establish the base guidelines on which we would operate so that we did not develop the wrong system.

After we set the project outline, the lack of a suitable dataset became evident, so Withus decided to develop a new one that fulfilled every requirement. This led to the development of a dataset preprocessing and conversion step where we transform the raw data into a NILMTK cleaned dataset.

Having the project outline defined and the dataset processing ready, we focused on the models' development, iterating multiple times over the proposed architectures to solve the issues we faced throughout the development.

3.1 Project outline

A successful project needs a clear vision. Otherwise, with constantly changing requirements, no work can be developed.

In NILM, there are several directions we could take when developing a new model. We must choose between high and low-frequency sampling, supervised and unsupervised learning, ON/OFF classification or energy disaggregation, event-based or eventless models, and the type of model we will implement.

3.1.1 Sampling frequency

One of the most limiting decisions in NILM is the sampling frequency we choose.

As seen in Section 2.3, we are limited in the features we can extract when using low-frequency sampling. In consequence, high-frequency sampling becomes very appealing, and the results presented in the research show its benefit [17], [39], [41]. Even so, we need to remember that the project's final objective is to build a viable product, and the expensive deployment of high sampling invalidates its use, leaving us with low-frequency sampling as our only option.

Even though low-frequency sampling is limiting when compared with high-frequency sampling, its results are constantly improving [18], [43], [44] and prove that we can deploy a NILM solution for real-world use with it.

3.1.2 Model type

When the project was proposed, the objective was to use ML models to solve NILM. Despite that, we needed to understand our options and if it even made sense to use ML. From the analysis of Section 2.7, we concluded that the ML models were a viable option and the supervised alternative obtained SOA results [18], [43], [44]. Since we could bear the extra cost of recording appliances for the training of our models, we decided to use supervised ML models.

Having decided the type of model, we need to decide the type of event detection. Event-based models are a frequent practice [10], [33], [34], [35], [71] but they require an event detection algorithm. Keeping in mind that we are using low-frequency sampling, we could also use the eventless approach. This approach does not require an event detection algorithm and is frequently used in ML models such as [3], [18], [44], where they achieve SOA results. Since we wanted to focus our attention on the ML models and the eventless approach showed promising results, we opted to implement our models using it.

3.1.3 Solved task

The job of a NILM model is to obtain the energy consumed by an appliance at any given time. This is an arduous task, and to simplify it, some of the proposals [10], [26], [31], [33], [46], [78], [79], see NILM as a classification problem. Here, instead of obtaining the energy consumed by the appliance, we are only concerned with classifying it as ON or OFF.

Since ON/OFF classification is a simpler task than energy disaggregation and we were taking our first steps in NILM, we concluded that the best option was to start by implementing models to solve it. When these models obtained satisfactory results, we would transition them to energy disaggregation.

3.1.4 Datasets

Having defined an outline for our models, the type of data they will consume, and what they will solve, the only thing left to start the implementation is obtaining the data. Section 2.8 concluded that all datasets lacked one or more characteristics to be considered a complete dataset, so how do we choose which one to use?

The first idea was to create a dataset with all the characteristics we needed. However, a dataset with an extended recording period cannot be built without passing the extended period. Therefore we needed a temporary solution while the dataset was being built. The only solution was to alleviate one of the dataset characteristics so that a couple of datasets respected all the requirements.

Given that most SOA algorithms and the ones available in NILMTK-Contrib disaggregate the energy consumed by the appliances using only one electrical feature, we decided that we could drop the requirement of multiple electrical features until the Withus dataset was ready. With these settings, the REFIT and UKDALE now fulfill all the requirements for a good dataset.

To support our choice, we evaluated the datasets according to the appliances they record, the number of missing values, the number of houses, the proportion of energy submetered, and usage in papers.

Dataset description

The UKDALE is a very popular dataset, where many algorithms have been tested [17], [18], [40], [41], [42], [44]. It contains readings from 5 houses in the UK, one of them recorded for over four years. The appliances are recorded under low-frequency sampling (6s) and the average proportion of energy submetered is 51%, where it is worth mentioning that house one has a coverage of 80% and house three has only coverage of 19%. As we will see in the evaluation of the models, we will only use five appliances, the washing machine, dishwasher, fridge, kettle, and microwave, which are present in the three houses we are using. The quality of the readings is not perfect, which is expected given the recording time and the lack of a preprocessing step. Even so, when we look at the average dropout rate of the aggregated readings, UKDALE only obtained 0.9%. This is reasonable given that the system recorded the aggregated data using a sampling frequency of 1s in the three houses used, which we downsampled to 6s to match the appliances recording. Since we downsampled the data, the number of missing values is much lower in those houses, given that for each sample we used, the recording device had to fail the reporting of 6 consecutive readings.

On the other hand, even though not as popular as the UKDALE, the REFIT dataset contains low-frequency readings(8s) from 20 UK houses during two years. The average proportion of energy submetered is lower than the UKDALEs (39%). However, as we mentioned, we are only evaluating five appliances, and each one is present in at least fourteen houses. The average aggregated readings dropout is higher than the UKDALEs (2%), which is expected as the aggregated readings in REFIT cannot be downsampled, making each missing value relevant. Even so, this is not critical as the dataset has a long recording period. The dataset was used to test other algorithms [33], [35], [38], [80], and considering the number of houses, it is excellent to evaluate the generalization capability of the models.

3.2 Dataset development

As we mentioned before, the available datasets lacked at least one characteristic to be considered a complete dataset. With this in mind, Withus decided to record a new one with all the characteristics of a good dataset. It recorded five houses from their employees focusing on the appliances we use on UKDALE and REFIT. The measurements collected were the active, reactive and apparent power, the power factor, current, voltage, and the active, reactive and apparent cumulative energy at a frequency of 0.5 Hz for both aggregated and appliance readings. This dataset is a proof of concept that will later be expanded to the national domain containing more houses and recording more appliances. Even so, the dataset contains readings from more than 25 devices, meaning that it is representative of the problems faced when creating a new NILM dataset.

Creating such a dataset is challenging as it comprises recording multiple long-duration time series, each representing one appliance or aggregated meter. With so many sensors, one might inevitably malfunction and create a gap of information in the recordings. Furthermore, each sensor produces distinct recording noise due to manufacturing defects that change the appliance's original signature [7]. To mitigate the effect of these two problems in the dataset's quality, we implemented a preprocessing step that fills missing values and smoothes the data while maintaining the original characteristics.

Once the dataset went through the preprocessing, it was ready to be used by the algorithms. Even so, if we wanted to take advantage of the multiple assets of NILMTK, we needed to adapt our dataset to their data format. To do so, we created the required dataset metadata and developed a conversion script that transformed our Comma Separated Value (CSV) files into the h5 file used by NILMTK.

3.2.1 Dataset Cleaning

The dataset cleaning is divided into two phases, filling missing values and smoothing the data. In the first one, we deal with the errors caused by miscommunication, accidentally unplugged meters, or any other cause that leads to the recorded time series missing one or more timesteps. The second one tries to minimize the recorded noise associated with the metering device by smoothing the data while preserving the meaningful events, such as consumption edges.

Filling missing values

Our first idea to deal with the missing values was to create a simple algorithm that, from an inconsistent time series, created an evenly spaced time series filling the missing values when needed. Later, we found a library, Traces [76], whose premise is to treat inconsistent time series. Since the library seemed to be an easier way to work with inconsistent time series, we compared it with our algorithm.

Our algorithm has a simple workflow. Let us consider O, the original time series with missing values, and N, the time series with no gaps. The algorithm starts by creating a local timestamp, lt, beginning at the first timestamp of O. The lt will be progressively incremented by the correct timestep until it reaches the last sample. While it goes through all the timestamps, it fills N using the values from O. The values from O are obtained through a separate index, i, representing the next value to be used. In each iteration, the algorithm compares its timestamp with the one corresponding to the index position. The algorithm checks three conditions to choose how to fill N :

- If the timestamp of i is within a margin of error of lt, the algorithm uses the corresponding value and moves to the following timestamp. The margin of error, e, is defined as $e = \pm^{\texttt{timestep}}/_{2}$;
- If the original time series is behind the local timestamp, the algorithm moves the original time series until it is synchronized and uses the synchronized value;
- If O is ahead of the lt, the algorithm averages the last value introduced in N and the value present in i.

To facilitate understanding the algorithm, Figure 3.1 contains a visual representation of the algorithm processing a time series with a timestep of 2 s.

The Traces library uses a similar concept. Focusing on the fact that most time series are inconsistent, the library masks the missing values by interpolating them. To do so, we have to load the data into a structure called TimeSeries, and when we want to obtain the value from a given timestamp, we query it. In the background, when queried, the TimeSeries checks if the timestamp exists, and when it does not, it interpolates the value according to a specified method. The methods we can use are either *linear*, where a linear function is computed between the previous and next sample, or *previous*, where the returned value is the last recorded sample. These methods are helpful when analyzing a small sample of data; however, we want to reformat the complete time series. In that case, Traces has a specific method to create evenly spaced time series with a given timestep using moving average.

With the three options provided by Traces and the algorithm we implemented, we have four options to fill missing values. We will evaluate each one of them and choose the one that recreates the original signal most closely.

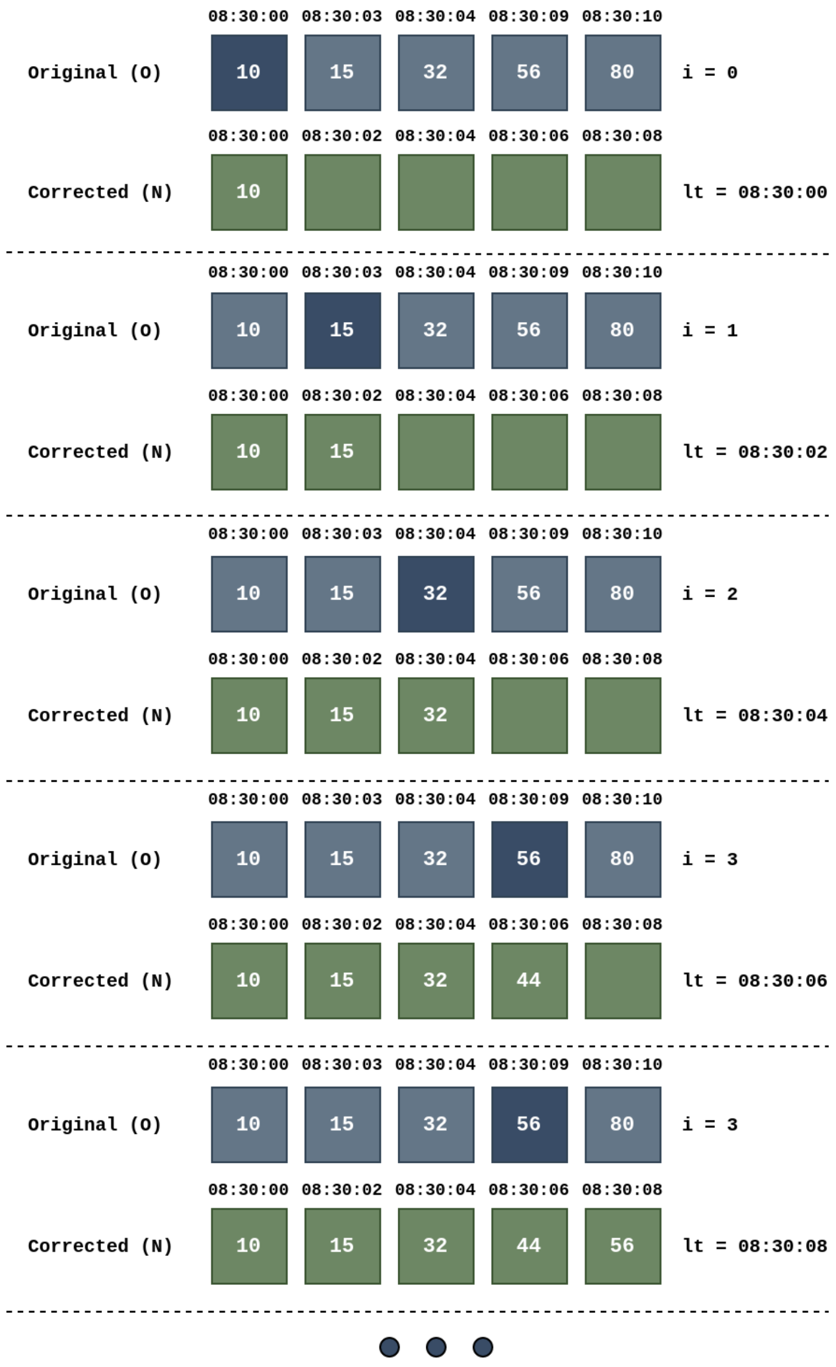


Figure 3.1: Example of our algorithm processing a time series.

Smoothing data

An appliance signature is closely related to the metering technology used, meaning that each different meter can yield different values for the same appliance [7]. In consequence, when we train our models, they will inherently learn the meter's characteristics. One way to reduce the effect of the metering noise is by smoothing the data [79].

There are multiple ways to smooth the data, and most of them consist of a sliding window that updates each value present in the time series according to a given function that uses the values in the window. We experimented with three data smoothing techniques: Simple Moving Average (SMA), Exponential Weighted Moving Average (EWMA), and Median Filter.

The SMA, as the name indicates, does a simple average between the values inside the window. This method is regularly used to smooth the data in other time-series fields such as wind energy production [81] or general noise reduction [82], but it can also help predict trends, as is the case in stocks trading [83]. The SMA's advantage is the ease of calculation as we can use Equation 3.2 instead of the complete summation (Equation 3.1) to calculate the moving average for each new example.

$$SMA = \frac{1}{M}\sum_{i=0}^{n-1} x_{M-i} \tag{3.1}$$

$$SMA = SMA_{prev} + \frac{1}{M}(X_M - X_{M-n}) \tag{3.2}$$

The median filter is a non-linear mathematical tool that reduces noise in a time series while keeping the sharp, sustained changes [79]. To do so, instead of calculating the average value of the given window, it finds its median value. This method is the most popular in NILM, being used as a preprocessing step to increase the performance of the algorithms or downsample the data [12], [48], [84], [85].

Unlike SMA and median filter, the EWMA does a weighted average where the most recent values have bigger weights. As a result, it can be better at capturing energy consumption spikes than SMA. The formula used is shown in Equation 3.3, where r_t is the observed value at time t and α is the weight defined by the user.

$$EWMA_t = \alpha \times r_t + (1-\alpha) * EWMA_{t-1} \tag{3.3}$$

We experimented with these three approaches to find the function that removes the noise from the readings without removing essential features, so our dataset closely represented the actual appliances' consumption.

3.2.2 Dataset converter

The NILMTK's main objective is to allow easily reproducible experiments with multiple datasets through a standardized interface. However, that is not all it has to offer, as it contains a diverse set of tools for analyzing the dataset and the quality of the recorded data.

These benefits led us to build a dataset converter that transformed the recorded data into the NILMTK data format. To properly use the dataset in NILMTK, we need to create some metadata files that tell it how the dataset is structured and tell the user relevant characteristics of the meters and households. After creating the needed metadata files, we created a script to convert the CSV files into the final h5 file and combine it with the metadata written.

NILMTK metadata

In NILM, the metadata of a dataset gives valuable information. Besides telling the appliance that each meter recorded, the metadata contains the characteristics of the household, the number of inhabitants, and their demographics, which are characteristics that influence how the appliances are used and consequently the disaggregation algorithm's performance. In a NILMTK dataset, the metadata is divided into at least three files written in Yet Another Markup Language (YAML).

```yaml
contact: jack.kelly@imperial.ac.uk
creators: ['Kelly, Jack']
date: '2017-04-26'
description: Appliance-by-appliance and whole-home power demand for 5 UK homes.
description_of_subjects: 4 MSc students and 1 PhD student.
funding: [Jack Kelly's PhD is funded by an EPSRC DTA, Hardware necessary for this
    project was funded from Jack Kelly's Intel EU PhD Fellowship]
geo_location: {country: GB, latitude: 51.464462, locality: London, longitude: -0.076544}
geospatial_coverage: Southern England
institution: Imperial College London
long_name: UK Domestic Appliance-Level Electricity
name: UK-DALE
number_of_buildings: 5
publisher: UK Energy Research Centre Energy Data Centre (UKERC EDC)
```

Code Block 2: Sample of UKDALE's dataset.yaml metadata file.

The first one is the *"dataset"* file, where the general information of the dataset is presented, such as name, number of buildings, and period covered. This file contains only general information, and it is the starting block for the metadata. An example of this file is presented in Code Block 2.

```yaml
CurrentCostTx:
  manufacturer: Current Cost
  max_sample_period: 120
  measurements:
  - {upper_limit: 25000, lower_limit: 0, physical_quantity: power, type: apparent}
  model: Transmitter
  model_url: http://www.currentcost.com/product-transmitter.html
  model: CurrentCost Tx
  sample_period: 6
  wireless: true
EcoManagerWholeHouseTx:
  brand: EcoManager
  brand_url: http://www.edfenergy.com/products-services/for-your-home/ecomanager
  manufacturer: Current Cost / Sailwider
  max_sample_period: 120
  measurements:
  - {upper_limit: 25000, lower_limit: 0, physical_quantity: power, type: apparent}
  model: EcoManager Whole House Transmitter
  sample_period: 6
  seller: EDF Energy
  site_meter: true
```

Code Block 3: Sample of UKDALE's meter_devices.yaml metadata file.

The second one is the *"meter_devices"* file that contains the details regarding the metering devices used. This file describes each metering device by defining characteristics such as the model, maximum sampling period, communication protocol, measurements recorded, and sampling period. The first three influence the measurement noise or missing values, and the other two tell us the features we can use in the NILM models. Code Block 3 presents a sample of the *"meter_devices"* used in UKDALE.

Lastly, we need one *"building"* file for each house in the dataset to complete the metadata. In this file, we specify the household information such as the number of inhabitants, the type of heating, the type of building, and the construction year, which help us understand the different usage patterns of certain appliances such as the AC. Besides information about the house, the file also stores the recorded appliances, the meters used, and the metering hierarchy. When we specify the appliance recorded, we can also specify its metadata by filling in information like the model, brand, year of purchase, and efficiency rating. An example of the building metadata file is shown in Code Block 4.

```yaml
instance: 1
original_name: House 1
description: House description
construction_year: 2010
m_occupants: 4
energy_improvements: ["photovoltaics"]
heating: ["electricity"]
building_type: cottage
geo_location: {country: PT, latitude: 40.6333333, longitude: -8.65, locality: Aveiro}
timezone: Europe/London
timeframe: {end: '2020-10-01T00:00:00', start: '2021-02-05T00:00:00'}
elec_meters:
  1:
    device_model: aggregate_meter
    submeter_of: 0
    submeter_of_is_uncertain: false
    site_meter: true
    timeframe: {end: '2020-10-01T00:00:00', start: '2021-02-05T00:00:00'}
    name: Aggregated meter
    disabled: false
  2:
    device_model: plug_meter
    submeter_of: 1
    site_meter: false
    timeframe: {end: '2020-10-01T00:00:00', start: '2021-02-05T00:00:00'}
    name: heatpump
    disabled: false
appliances:
  - description: Small description regarding the appliance
    type: heat pump
    instance: 1
    meters: [2]
    control: ‘thermostat’
    efficiency_rating: A+
    model: Heater_2
    manufacter: Electric Heater Corp.
    brand: Electric Appliances
    year_of_purchase: 2018
    year_of_manufacture: 2018
```

Code Block 4: Example of a building.yaml metadata file.

If all the metadata files of NILMTK are correctly filled, we can do consumption profiling of appliance brands and typical households without even using a NILM model. If we want to build a NILM model, the metadata eases the data loading process and helps us understand the possible different results obtained between houses.

Conversion script

Having the dataset metadata written, the only thing left is to combine it with our CSV files to create the h5 file respecting the NILMTK data format. To achieve this, we built a script that used the helping functions of NILMTK for dataset creation.

The first step we need to do to convert the dataset is to create the *Datastore* that saves the multiple time series. This is done through the *get_datastore()* method from NILMTK, where we specify the file destination and the viewing mode we are using. After the datastore is opened, we need to fill it with data. To place the data in the *Datastore*, we need a *Key* object that identifies the meter we are storing and to which house it belongs. The data itself needs to be in a *pandas DataFrame*, which is not a problem, as *pandas* has a method to load data from CSVs. Once the data is in a *DataFrame*, we need to ensure that the column names follow the same nomenclature used by the NILMTK-Metadata, through the *pandas MultiIndex*. Having done all these steps, we can place the data using the *put()* method.

At this point, all the CSV data should be loaded, and to finalize the conversion, the only thing left is to load the metadata into the *Datastore*. This is made available through the *save_yaml_to_datastore()* method, which receives the metadata folder path and the store that we filled with data.

The process is not that hard once we understand what each method does and how they should be combined. To prove it, Code Block 5 presents a dataset converter using only a few lines of code, and Figure 3.2 contains the folder structure that allows the converter to work properly.

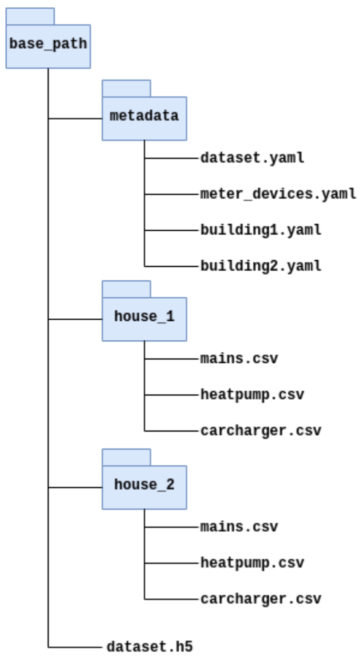


Figure 3.2: Folder structure used in the dataset converter example.

```python
from nilmtk.utils import get_datastore
from nilmtk.datastore import Key
from nilmtk.measurement import LEVEL_NAMES
from nilm_metadata import save_yaml_to_datastore
import pandas as pd

houses = [1, 2]
appliance_meter_mapping = {
    "mains" : 1,
    "heatpump" : 2,
    "carcharger" : 3
}
base_path = "./data/"
metada_path = base_path + "metadata"
output_filename = base_path + "dataset.h5"

store = get_datastore(output_filename, "HDF", mode='w')

for house_id in houses:
    for appliance, meter in appliance_meter_mapping.items():
        key = Key(building=house_id, meter=meter)

        csv_filename = base_path + "house_" + str(house_id) + "/" + appliance +".csv"
        df = pd.read_csv(csv_filename)

        df.columns = pd.MultiIndex.from_tuples(
                [column_mapping[c] for c in df.columns.values],
                names=LEVEL_NAMES)
        store.put(str(key), df)

# Add metadata
save_yaml_to_datastore(metada_path, store)

store.close()
```

Code Block 5: Example for building a converter from csv to h5.

The hardest part of building both the metadata files and the conversion script was the lack of documentation, as the only thing we could use was the NILMTK source code of other dataset converters. Nevertheless, once we understand how the files are organized, the information stored in them, and how the different methods work, the process of building a dataset converter is straightforward.

3.3 Models development

At this point, the data we are going to feed the model is no longer a concern, as Withus is recording a new dataset with all the needed features, and the steps to clean and convert it were implemented. Moreover, while we wait for the dataset to have a reasonable recording period, we can use the two other datasets mentioned before in Section 3.1.4 to develop our models.

This allows us to focus on the main objective of this dissertation, the proposal of a NILM model, which can be a daunting task. One way to diminish NILM's complexity is to simplify the task by reducing the problem to ON/OFF classification, as explained in Section 3.1. To ease the familiarization with the multiple libraries, the window-based solutions, and the overall NILMTK model's structure, we started by creating models for the ON/OFF classification task. Once we got some momentum and the results in this task were satisfactory, we would transition towards disaggregating appliances consumption. Since the appliance disaggregation is a regression task and the ON/OFF identification is a classification task, we divided the models into classification or regression.

3.3.1 Classification models

The classification models were not built with the intent to obtain SOA results. Their objective was two-fold, facilitate the task at hand while we were learning the ropes and serve as an initial filter for the viability of the different proposed models. The simple models completed the first objective, as they were used to learn the libraries. The advanced models were tested under the classification task to understand if their results in this more simplistic task justified their use in the disaggregation task.

When we start developing a NILMTK model, there is a lot to take in. Just respecting the *Dissagregator* class can be difficult as the inputs and outputs of the different methods can be hard to understand, given the lack of documentation. To avoid creating overhead to the already complicated task, we started by implementing two simple models: the SVM and SimpleGRU. The SVM model was created as the base for any disaggregation model, and SimpleGRU was created to serve as the base model for any NN.

SVM

The SVM is one of the most popular ML models, being widely deployed in various research areas. It is a supervised model which can be used in both classification and regression tasks. Its main advantage against the newer machine learning models like NNs is its performance when only a few training examples are available (≤ 1000). The algorithm divides two classes using an hyperplane that maximizes the distance between the nearest example of each class. As an example, Figure 3.3 presents two possible hyperplanes for the SVM. As we can see, the hyperplane in grey has a much smaller margin than the black one making the latter the optimal hyperplane.

Some out-of-the-box SVM classifiers are already implemented in Python, so we do not need to develop our own. We are using the scikit-learn implementation of the SVM for classification, which they named *SVC*. We opted for this implementation as scikit-learn is a well know library with thorough documentation and easy-to-use models. The hyperparameters values we use are present in Table 3.1. We decided not to optimize the model's default values as its purpose was to help us implement a first NILMTK model, not to solve the task. Besides, in NILM, SVMs only obtained significant results in high-sampling frequency implementations [86], [87], and the default values from scikit-learn are a reasonable choice.

Using the *SVC* with the default values, we could implement the algorithm with just three lines of code. This allowed us to focus our attention on the methods we needed to develop, understanding the experiment API, the workflow of the model, and how the data was handled.

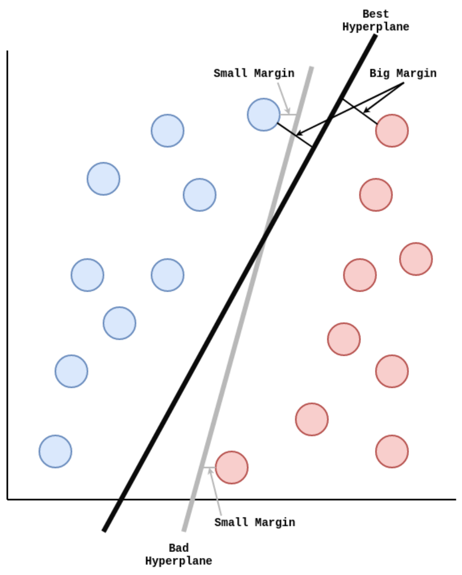


Figure 3.3: Example of how an SVM choses the optimal hyperplane.

	Values
C	1
Kernel	rbf
Gamma	scale
Tolerance	0.001

Table 3.1: SVM parameters.

Once we finished implementing this model, we had the general code base for every other model as most of the model's lifecycle remains the same.

SimpleGRU

To create NNs, we cannot use scikit-learn. Instead, we have to learn a new library, TensorFlow. The easiest way to learn how to use TensorFlow is to build a simple model using it, and that is where SimpleGRU comes to play. With this simple model, we learned how to use the TensorFlow library, how the model is built through the Keras API, how we can obtain the training history, and what are the data formats expected and returned by the different methods. This process was simplified thanks to the various examples and detailed documentation of TensorFlow and the fact that we already knew the NILMTK's workflow from the SVM implementation.

The model is composed of a GRU and a Dense layer, as seen in Figure 3.4. The GRU layer was chosen to understand how the Recurrent units worked in Tensorflow and how they processed the data. Since they maintain information from the previous examples to predict the present example, we wanted to understand when we needed to maintain the order of the data to avoid harming the algorithm's performance. The Dense layer was necessary to obtain the classification output. Using the softmax activation on the Dense layer, the model outputs the probability of the example belonging to each class. The number of classes is given by the number of nodes in the Dense layer, and the class chosen is the one with the highest probability.

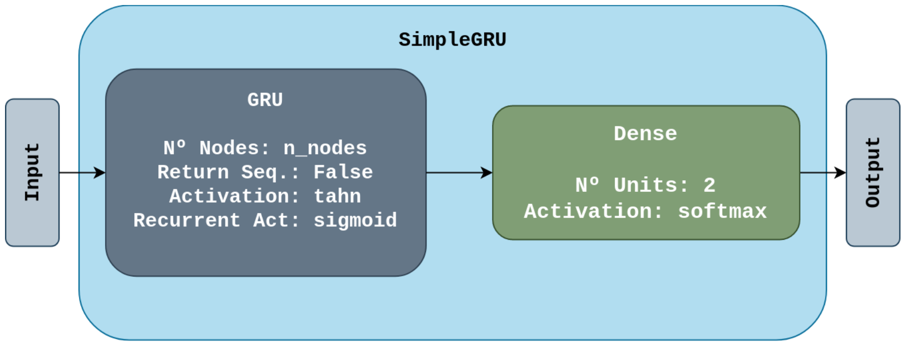


Figure 3.4: Architecture of the SimpleGRU model.

After developing this simple model, to create any other NN, the only thing we needed to worry about was its architecture and the data it received as the remaining code was developed to be reusable.

Now that we understand the workflow of the NILMTK API and how to build a NN using TensorFlow, we are ready to implement the models we want to apply to NILM. As we mentioned above in Section 3.1.2, we propose models based on the eventless approach, where each model pursues a different research direction to evaluate its viability. We proposed three models: ResNet, DeepGRU, and MLP.

ResNet

Since NILM can be seen as a time series classification task, we searched its SOA for relevant models to understand the available options. During that search, we found the ResNet [88], a network proposed by Z. Wang and T. Oates that obtained good results in diverse benchmarking datasets for time series [50]. The network's performance led us to try it in NILM and see if it was worth developing a similar architecture as we have not seen a residual network proposed for NILM.

The ResNet is a CNN composed of three blocks with convolutions. Its main characteristic is the use of skip connections where the input and output from a block are combined before being fed to the next, as seen in Figure 3.5. The skip connections bypass the vanishing gradient problem, which allows us to create much deeper networks.

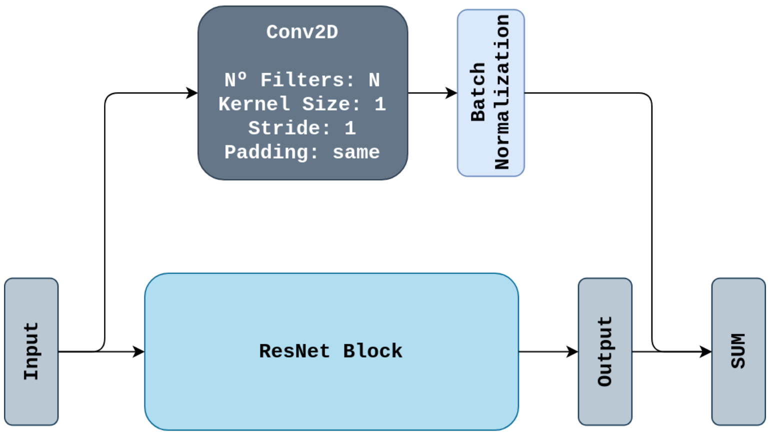


Figure 3.5: Example of the skip connection of a residual network.

The blocks in this network are composed of three convolutions with batch normalization and *ReLU* as the activation function. The convolutions have a kernel size of 8, 5, and 3, and the number of filters stays the same within each block, with the first block having 32 and the other two blocks having 64. A visual representation of a convolution block is shown in Figure 3.6.

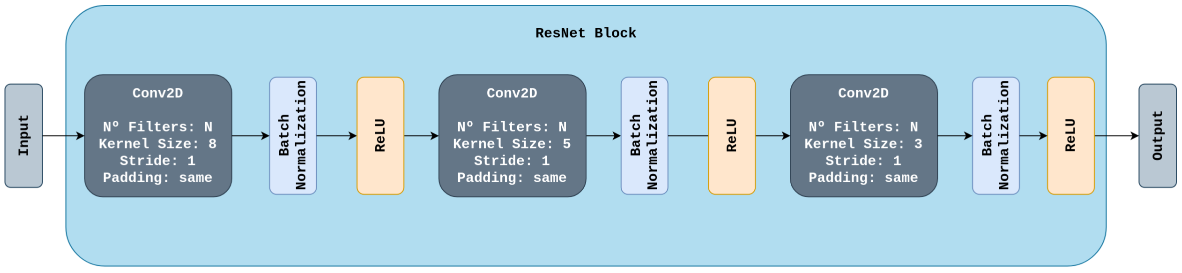


Figure 3.6: Architecture of the ResNet block.

After the three convolution blocks, the network does a global average pooling and proceeds to classify the example through a dense layer with softmax activation. The complete architecture of the network is presented in Figure 3.7, where N is 32.

Following the initial tests, it was clear that the network had overfitted the training data, so we decided to start changing the architecture to solve this problem. Our first change added a dropout layer right after the global average pooling, which would force the network to drop some irrelevant features that might cause the overfitting. This resulted in the architecture seen in Figure 3.8, maintaining N as 32.

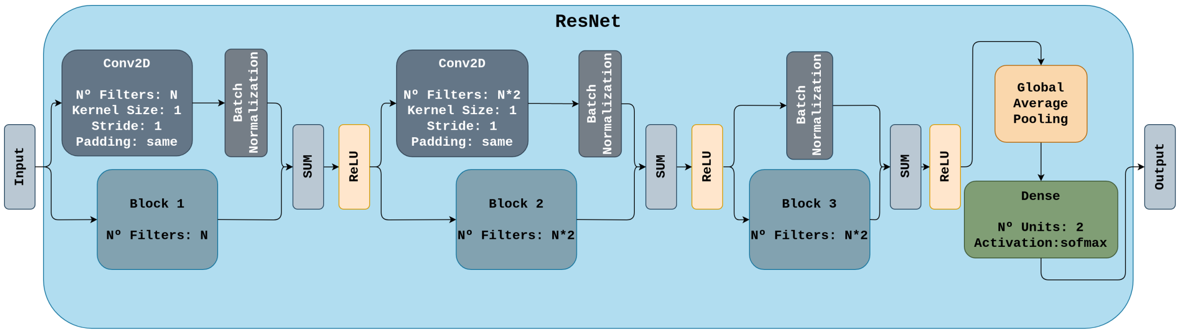


Figure 3.7: Architecture of the first proposed ResNet model for classification.

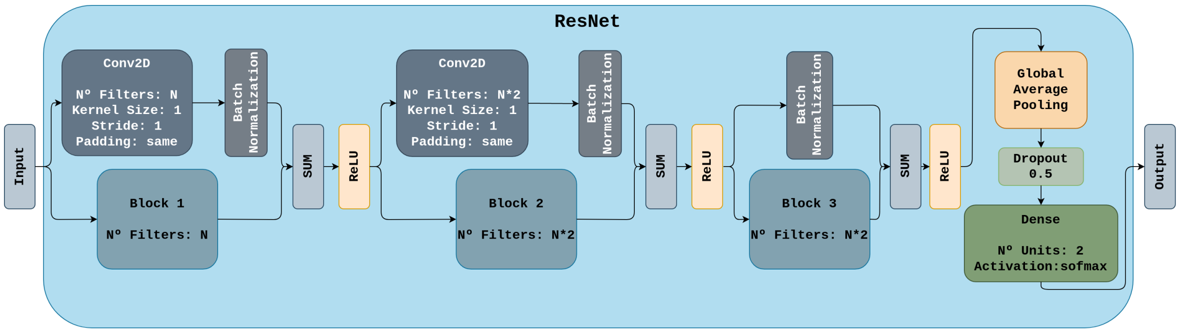


Figure 3.8: Architecture of the second proposed ResNet model for classification.

Even though the results showed a decrease in the overfitting, it was not enough as the model was not generalizing well to unseen houses. To force the model to generalize, we added a dropout layer after each block. By doing so, we removed the less essential features even further, making the model focus on the invariable characteristics of the appliance between houses. The final architecture, seen in Figure 3.9, where N is 32, still overfitted the training data. However, since we were already comfortable with the network and a schedule restrained us, we decided to transition it to the disaggregation task.

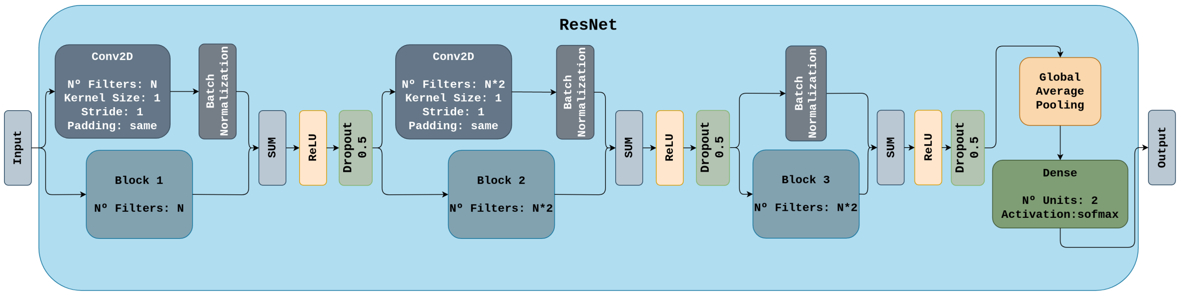


Figure 3.9: Architecture of the third proposed ResNet model for classification.

DeepGRU

While we were examining the SOA, we found multiple RNNs [17], [25], [38], [39]. Even so, all of them were shallow networks with two GRU or LSTM layers. Since there were no results with deeper networks based on recurrent units, we questioned if the RNN's performance would be improved if we built a deeper network with more GRU layers.

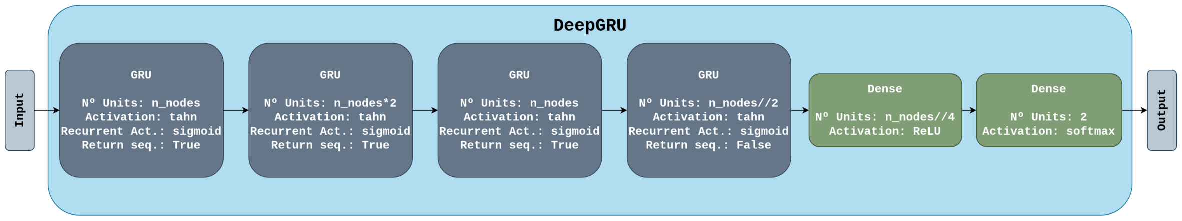
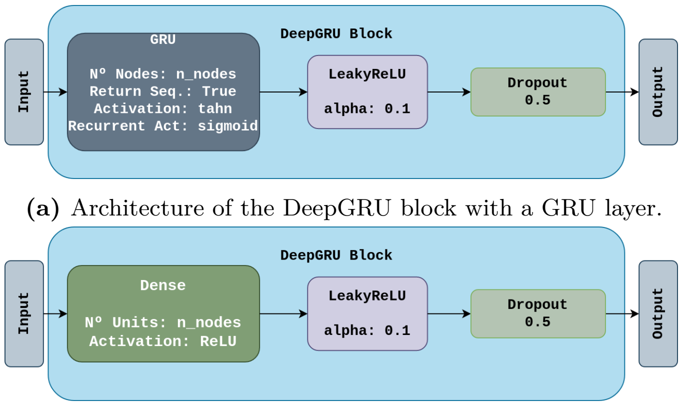


Figure 3.10: Architecture of the first proposed DeepGRU model for classification.

To evaluate this, we proposed a network with four GRU layers followed by two Dense ones. The network was designed to expand the latent feature space in the second GRU layer and then progressively reduce it to force the network to choose only the relevant features. The proposed architecture is seen in Figure 3.10, where *n_nodes* is 90.



(a) Architecture of the DeepGRU block with a GRU layer.

(b) Architecture of the DeepGRU block with a Dense layer.

Figure 3.11: Architecture of the DeepGRU blocks used for classification.

The model obtained good results, but there were some problems in the learning curves. To solve the issue, we adapted the proposed architecture using inspiration from the ResNet, creating an architecture based on blocks. The DeepGRU's blocks were composed of a GRU, LeakyReLU, and Dropout layer except for the last block, which would have a Dense layer instead of the GRU. Figure 3.11 presents both blocks used in this architecture.

As a starting point, we opted for only using three blocks followed by the Dense layer responsible for the classification, resulting in the architecture presented in Figure 3.12, where *n_nodes* is 90.

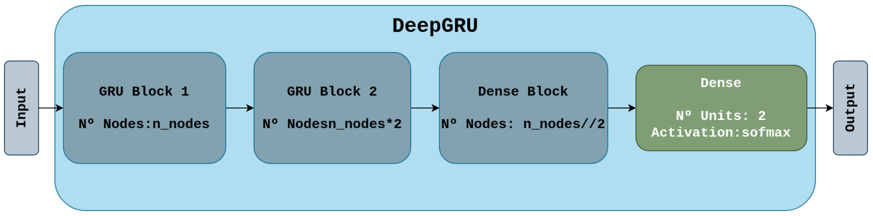


Figure 3.12: Architecture of the second proposed DeepGRU model for classification.

The results improved, but the network was composed of only two GRU layers, which meant we did not have the deeper network we proposed. Since we still had a margin for improvement, we decided to add a new block to the network and evaluate its performance. The resulting architecture is presented in Figure 3.13, where n_nodes is 90.

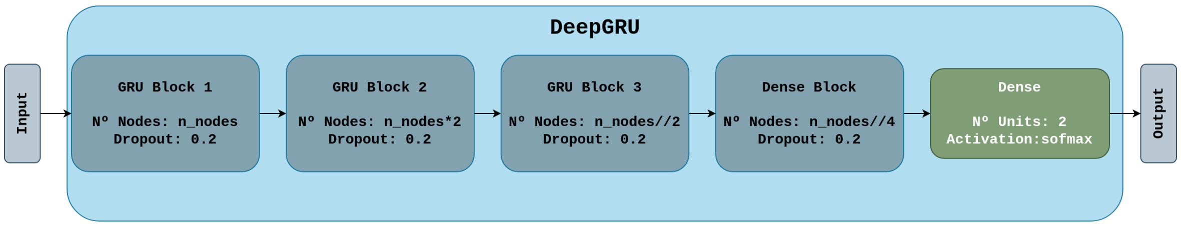


Figure 3.13: Architecture of the thrid proposed DeepGRU model for classification.

This architecture augmented even further the problems we saw in the first architecture, overfitting the data. Even so, due to time constraints, we decided not to explore this model further in classification and use the second architecture for regression.

MLP

So far, the models we implemented used the raw data as features. However, that is not the only option we have. For example, in [85], the authors study the benefits of combining multiple statistical features and electrical features. These methods are not frequent, and many feature extractors commonly used in time series are not explored. To evaluate the benefit of feature extractors against raw data, we decided to implement an MLP that would use the data obtained from a feature extractor.

The MLP was chosen for this task because it is composed of dense layers that cannot extract any temporal meaning from the information, are fast at making predictions, and usually have good performance with statistical data. As for the features used, we decided to extract a set of statistical features from the output of a DWT of the raw data. The extracted statistical features are the minimum, maximum, mean, standard deviation, entropy, mean crossings, variance, skew, RMS, mode, median, and 5, 25, 75, and 95 percentiles.

To understand why we opted for DWTs instead of FFTs or STFTs, we need to know how they work and their advantages. That said, Discrete Wavelet Transforms process a time series using a series of functions called wavelets, each with different scales. Since wavelets are localized waves with finite energy, we can use them to extract time-domain information in addition to the frequency information. This also means that when we multiply the wavelet by a signal, we only apply it to a section of the signal. To process the whole signal, we need to change the location where we apply the wavelet by moving a given stride after each multiplication. This process is called wavelet transform, and it is why we can extract time-domain information from the transformation.

The DWTs apply subband coding where the signal is passed through filters with different cutoff frequencies at different scales [89]. To compute them, we need to successively pass the signal through high and low-pass filters, which produce detail and approximation coefficients. In Figure 3.14, we have an example of a DWT applied to a chirp signal, where the signal was decomposed until level 5. As it is clear, each level of decomposition obtains information regarding different frequencies since the detailed coefficients of the first level only obtained information about high-frequency components, and the ones of the fifth level captured information regarding low-frequency components. Another crucial aspect of DWTs that is also noticeable while analyzing Figure 3.14 is that each detailed coefficient maintained the temporal information about when each frequency appeared in the signal.

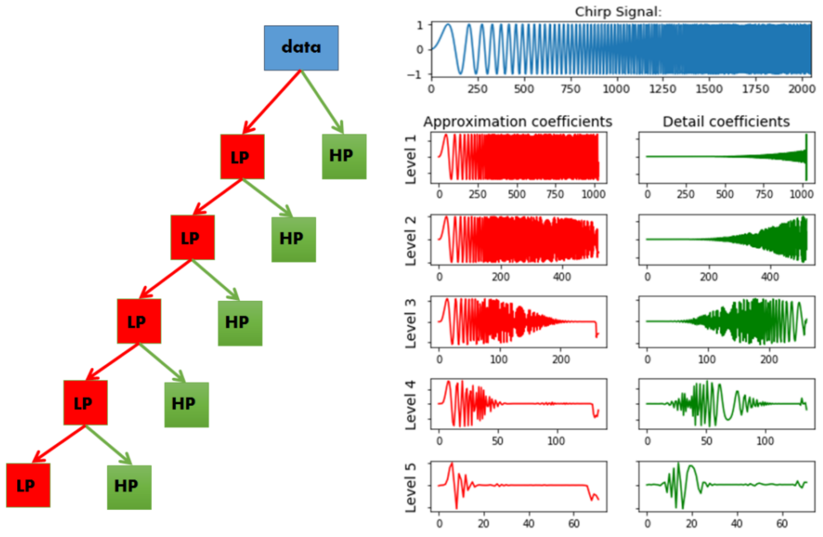


Figure 3.14: The approximation and detail coefficients of the sym5 wavelet (level 1 to 5) applied on a chirp signal, from level 1 to 5. Adapted from [90].

From a practical standpoint, DWTs are a great tool, being used for dimensionality reduction, noise filtering, and even singularity detection. In time series, its applications include similarity search, classification, clustering, pattern detection, and prediction [89].

Even so, in NILM, DWTs are primarily used for noise filtering and dimensionality reduction in high-frequency sampling solutions [91], [92], leaving out the use of DWTs as features [93]. To assess the viability of using DWTs as features in NILM, we applied a similar procedure to the one used in time series classification models. Where, for each window of raw data that the other proposed models usually process, the MLP will process the statistical features of the detailed coefficients and the last approximation coefficient extracted from that window of raw data using a wavelet, namely Daubechies 4. Figure 3.15 presents a visual representation of the feature extraction process.

We decided to use DWTs instead of FFTs or even STFTs (Short-Time Fourier Transforms) because FFTs cannot extract time-domain information, and STFTs can only extract information through fixed windows. This means that both approaches miss out on relevant frequency and time domain information. By using DWTs, we can extract both the appropriate time and frequency domain information, as they extract information through varying size windows. Figure 3.16 presents a graphical representation of how each of the three options analyzes a time-series signal. For an in-depth explanation of DWTs and their advantages, we recommend reading [89].

Now that we understand the features used by the model, we can discuss its architecture. The first proposed architecture was a simple neural network with five dense layers and one

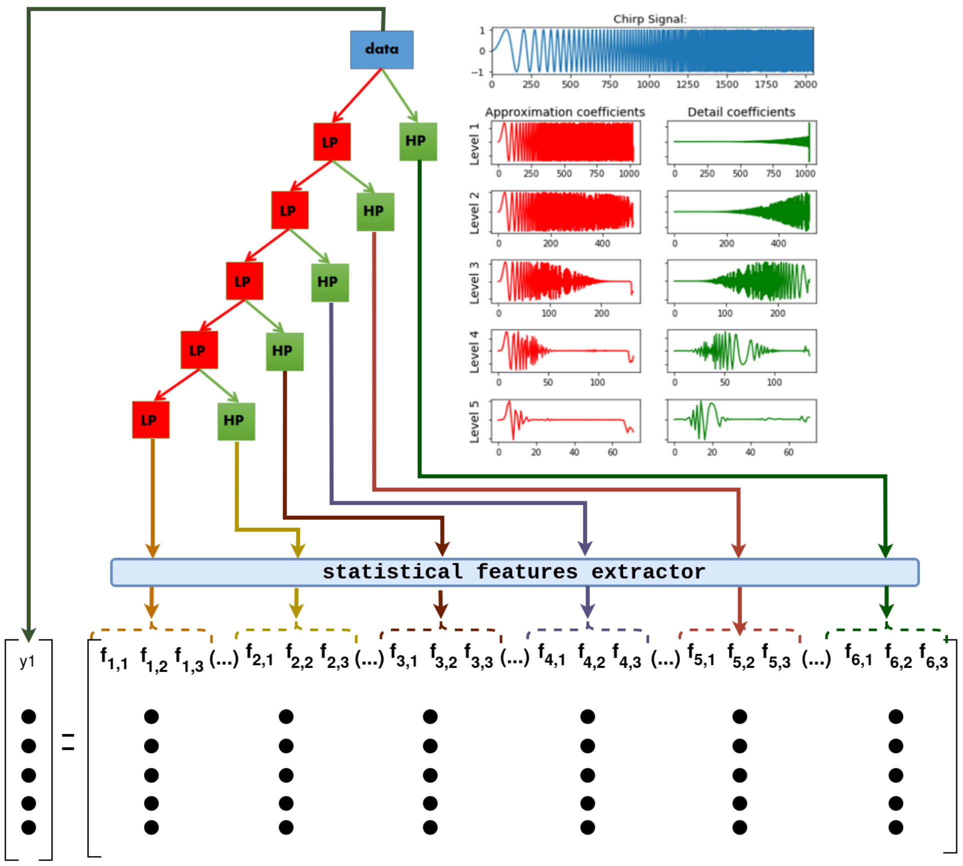

data
LP
HP
Chirp Signal:
Approximation coefficients
Detail coefficients
Level 1
Level 2
Level 3
Level 4
Level 5
statistical features extractor
y1
f1,1 f1,2 f1,3 (...) f2,1 f2,2 f2,3 (...) f3,1 f3,2 f3,3 (...) f4,1 f4,2 f4,3 (...) f5,1 f5,2 f5,3 (...) f6,1 f6,2 f6,3
=

Figure 3.15: A schematic overview of the feature extraction process using DWTs.

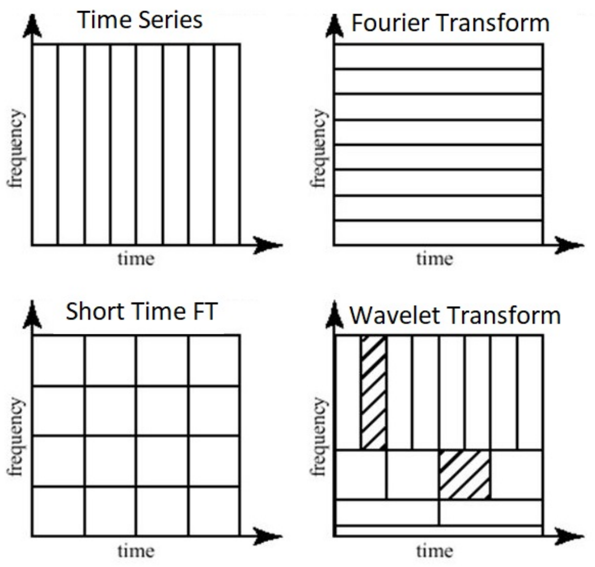


Figure 3.16: A schematic overview of the time and frequency resolutions of the different transformations in comparison with the original time-series dataset. The size and orientations of the block gives an indication of the resolution size. Adapted from [90].

dropout. It was designed to force the model to choose the most relevant features by reducing the number of nodes in the second and fourth layers and combine them by increasing the number of nodes in the third layer. Since we were experiencing overfitting in the remaining models, we also decided to place a small dropout before the classification layer. Figure 3.17 presents the described architecture where *n_nodes* is 256.

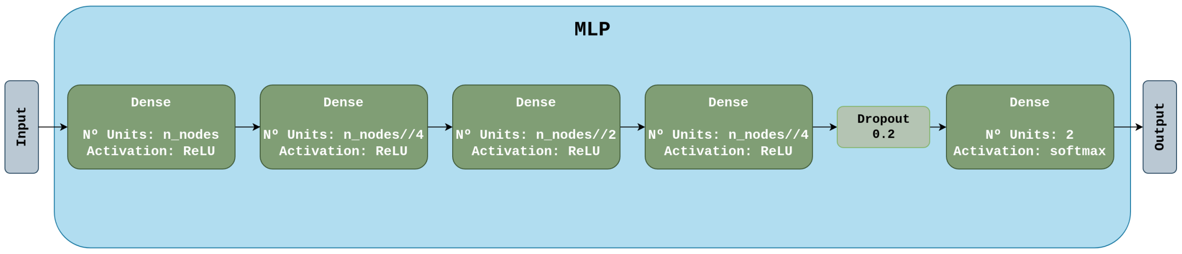


Figure 3.17: Architecture of the first proposed MLP model for classification.

The results were promising, but we had a margin for improvement, so instead of increasing and decreasing the number of nodes in the layers, we tried to progressively reduce them, diminishing the chance of overfitting the training data. The proposed architecture is shown in Figure 3.18, and by using it, we obtained similar results to the ones obtained by the previous network. The results were also within the expected values for the transition towards disaggregation, so we did not explore the model further on this task.

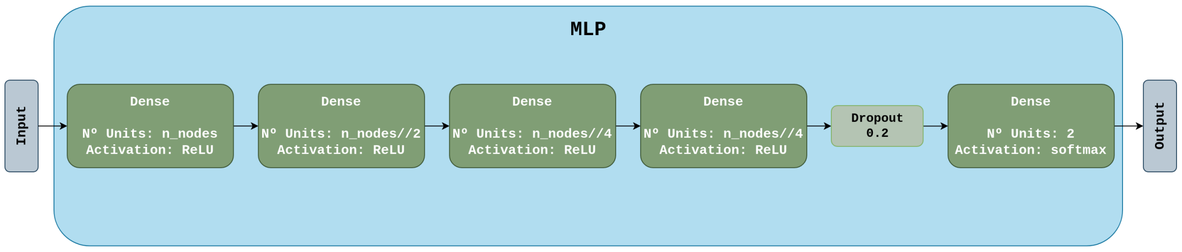


Figure 3.18: Architecture of the second proposed MLP model for classification.

3.3.2 Regression models

Disaggregating energy is a much more challenging task than ON /OFF classification, as we need to focus on the appliances' multiple states and their power consumption. So, when we transitioned our models to this task, we were not expecting that they would solve it perfectly, quite the opposite since we were sure we needed to continue optimizing the networks and experimenting with different architectures. Even with the added effort, our models must solve this task as the simple ON/OFF classification does not provide enough information to educate the consumer and create a HEMS system that gives actionable feedback.

ResNet

Starting with the ResNet, we decided that since disaggregation is a task that is not as simple as classification, we should test the network without the added dropout layers to verify if the network still overfitted the training data. This meant that we would use the first proposed architecture for classification, only changing the last layer to solve the regression task. The resulting architecture can be seen in Figure 3.19, where N is 32.

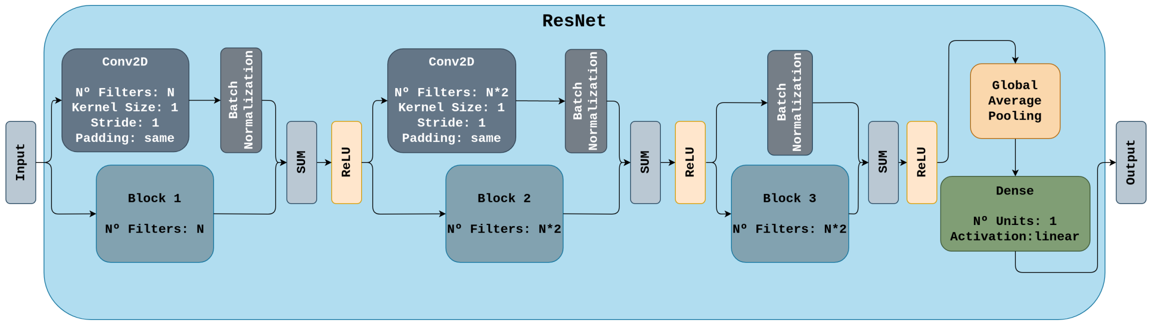


Figure 3.19: Architecture of the first proposed ResNet model for regression.

The results showed that the model overfitted the training data, so we decided to put back the dropout layers, which resulted in the architecture presented in Figure 3.20, maintaining N as 32.

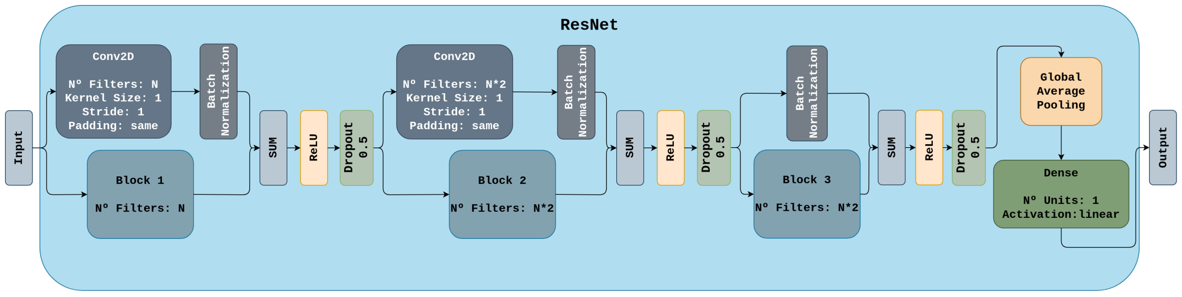


Figure 3.20: Architecture of the second proposed ResNet model for regression.

Depending on the appliance, the model was either not overfitting the data or had worse performance. To find a middle ground where every appliance obtained better performance, we tried reducing the dropout we were using to 0.2, which slightly improved the test set. At this point, due to time constraints, we could no longer experiment with new revisions to the architecture, making the final architecture used, the one presented in Figure 3.21, with N equal to 32.

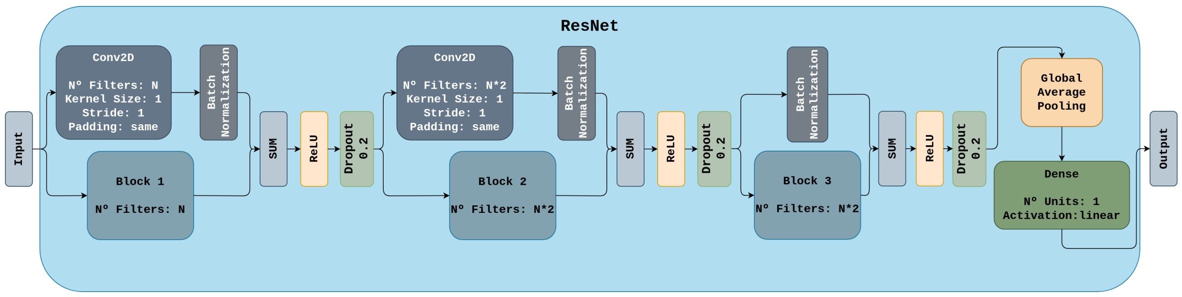


Figure 3.21: Architecture of the third proposed ResNet model for regression.

DeepGRU

In DeepGRU, we opted to use the second proposed model for classification and evaluate it before making any changes to the architecture. This meant we would only change the network's last layer, originating the architecture in Figure 3.22, where *n_nodes* is 90.

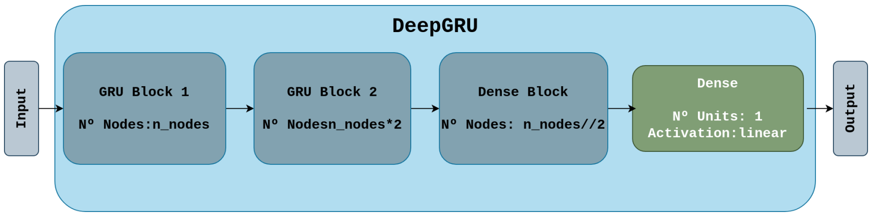


Figure 3.22: Architecture of the first proposed DeepGRU model for regression.

The results were satisfactory, but there was room for improvement, so we changed the architecture. We proposed an architecture still based on blocks, where each block would have two GRU layers followed by a dropout. The network followed the principle of funneling that we also used in the MLP in Section 3.3.1. The block architecture is shown in Figure 3.23, and the complete model is presented in Figure 3.24, where *n_nodes* is 128.

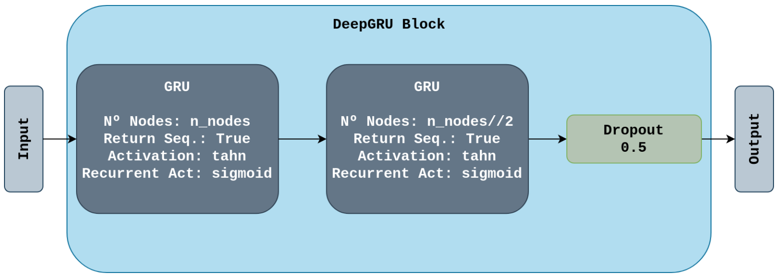


Figure 3.23: Architecture of the first DeepGRU block used for regression.

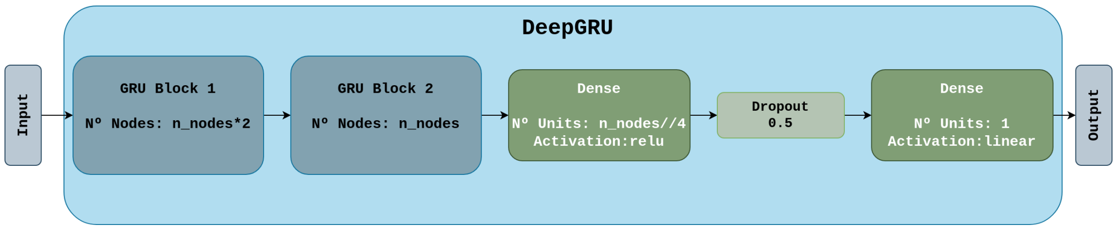


Figure 3.24: Architecture of the second proposed DeepGRU model for regression.

The network improved the results obtained but also suffered from the vanishing gradient problem, and the training process was highly irregular. This meant that this block created an unstable network, and we should change it to obtain a network with a stable learning process that does not suffer from the vanishing gradient problem. We found that we might be misusing the GRUs, so we backtracked to the first regression model, changing the block architecture, substituting the LeakyRelu with a dense layer with a *ReLU* activation. The model was initially tested following the funneling technique. However, because it obtained the best results compared to the other models, we also decided to experiment with it under a random hyperparameters search. The final block is presented in Figure 3.25, and the final model architecture is displayed in Figure 3.26.

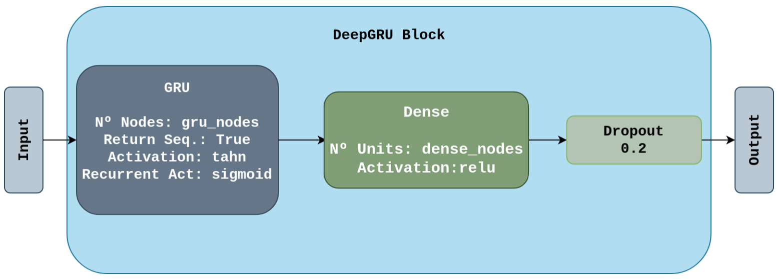


Figure 3.25: Architecture of the second DeepGRU block used for regression.

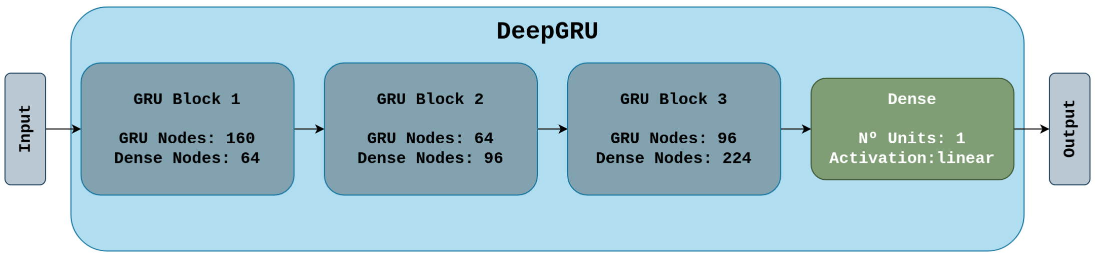


Figure 3.26: Architecture of the third proposed DeepGRU model for regression.

MLP

The transition from classification to regression with the MLP followed the same line of thought we used in the DeepGRU, where the first model proposed only changed the last layer of the architecture. This created the architecture presented in Figure 3.27, where *n_nodes* is 256.

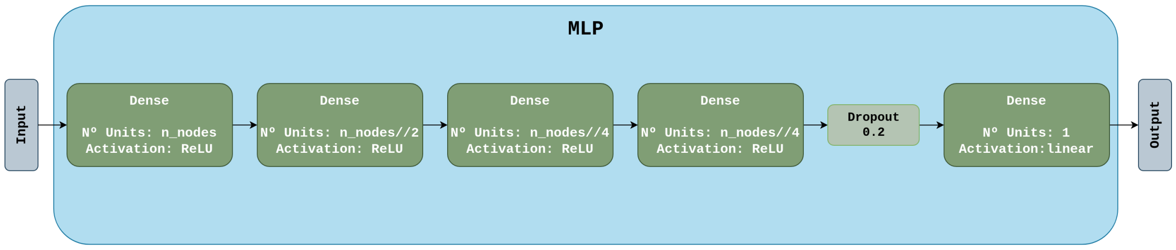


Figure 3.27: Architecture of the first proposed MLP model for regression.

The results showed that the network struggled to find patterns in the data, so its performance when using DWTs was the worst compared to all other models. To help the model find more features in the data, we changed *n_nodes* to 2048 and increased the number of nodes in the last dense layer before the regression layer, resulting in the architecture shown in Figure 3.28.

The increase in the overall number of nodes did not help the network, as the results did not improve, so we decided to try a similar architecture to the first one we proposed in classification. We start the network with a high number of nodes, which are drastically reduced

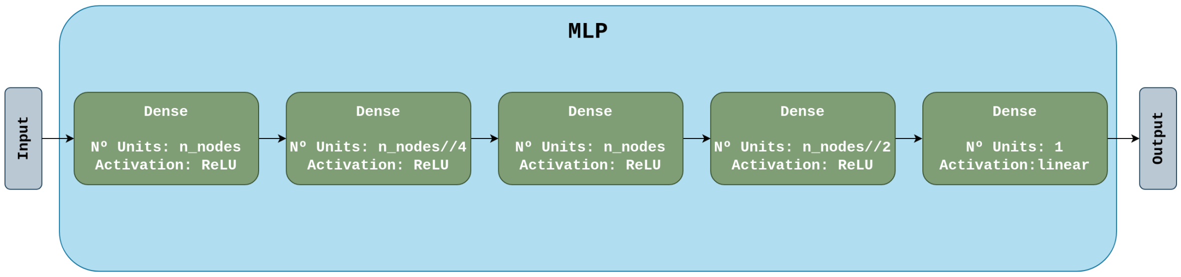


Figure 3.28: Architecture of the second proposed MLP model for regression.

in the next layer. In the third layer, we increase the number of nodes again and reduce them in the fourth layer. The most significant differences between the two architectures are the *n_nodes* we considered, as we are using 1024, and the amplitude of the increasing/decreasing changes in the number of nodes, as shown in Figure 3.29. With this architecture, we wanted to force the network to find a new latent feature space where the combined features allowed for better recognition of the appliance energy consumption.

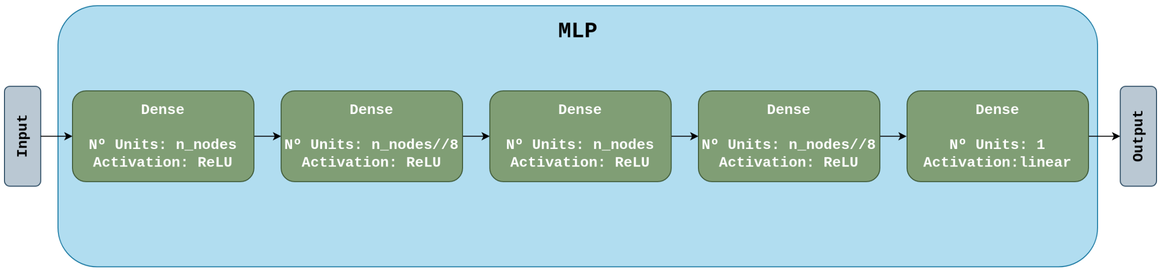


Figure 3.29: Architecture of the third proposed MLP model for regression.

Even with this architecture, the results did not improve, and unfortunately, due to time constraints, we could not research this model further using the last architecture for the remaining tests.

CHAPTER 4

Evaluation

During the development of the NILM dataset, we had multiple options for the preprocessing step. To choose which one to use, we needed to evaluate them. The same goes for the models we propose, as we needed a way to know how good they are and if the changes made improved them. In this chapter, we explain the experiments performed and present their results.

When faced with multiple options, the best way to choose is to evaluate them and opt for the one with the best results. To ensure that the option with the best results is, in fact, the best one, the tests made must be representative of the real-world scenario where we are applying it.

In this dissertation, we developed various preprocessing algorithms and NILM models, which needed to be evaluated to understand their advantages/disadvantages. In the remainder of this chapter, we will explain the experiments developed, the measurements used to assess our options, why those experiments are representative of a real-world scenario, and the results obtained.

4.1 Preprocessing experiments

When testing the preprocessing tools, we wanted to evaluate which one could clean the noisy signal without removing the characteristics of the original one and fill the missing values the most accurately. The original signal we chose is a day of the UKDALE's house one fridge, presented in Figure 4.1. We opted for the fridge, as its signature contains sudden spikes in energy consumption that can easily be mistaken by noise and removed.

The metric used to evaluate how closely the processed signal was from the original is the Root Mean Square Error (RMSE). The RMSE tells us the average distance between the processed values and the original ones, meaning that a good reconstruction has an RMSE close to zero. The formula used to calculate it is presented in Equation 4.1, where P is the processed signal, O is the original signal, and N is the length of the signal. We opted for

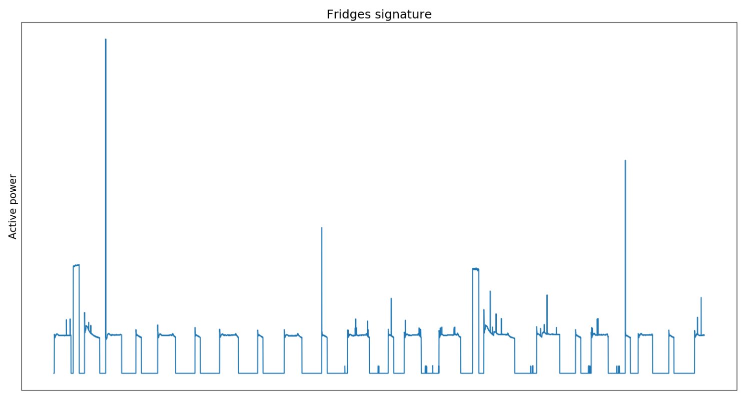


Figure 4.1: Signature used in the experiments of the preprocessing steps.

RMSE because it is a recommended error measure for signal processing, and it is commonly used in NILM to evaluate the disaggregation models [22].

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N}(P_i - O_i)^2}{N}} \tag{4.1}$$

Since our preprocessing was divided into two phases, filling missing values and smoothing data, we created separate experiments to evaluate them.

4.1.1 Filling missing values

The fridge's original signal we are using does not have missing values. This is actually a requirement for this experiment, as we need to know the value present in the gap of information to compare it with the one proposed by the algorithm.

To evaluate our algorithms, we artificially added 50 random holes in the time series experimenting with various sizes 1, 3, 5, 10, and 20. We used such a wide range of values because we wanted to understand the behavior of the solutions even in edge cases such as two minutes without readings (20 missing samples), allowing the team building the dataset to opt for the solution that better fitted the dataset's needs.

After adding the holes to the time series, each algorithm filled the missing values, and once they finished, we calculated the RMSE between the original signal and the processed signal. Each algorithm was tested ten times for each gap size to obtain its average performance. In Figure 4.2, a scatter plot shows the average of the errors obtained by the different algorithms according to the different sizes of gaps.

The results show that the *previous* technique is the worst one regardless of the size of the gap and that both our algorithm and the *linear* approach obtained close results until the five missing samples. That said, the best option to fill in missing values is the *linear* approach of Traces if we consider that gaps superior to 30 s can happen.

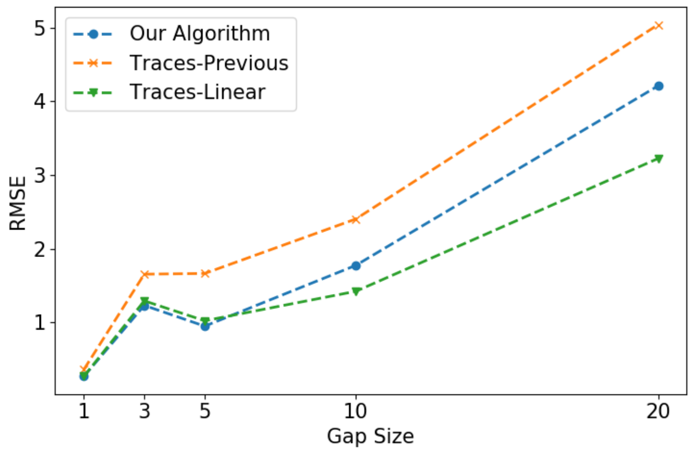


Figure 4.2: Average RMSE of the different algorithms while filling missing values depending on the gap size.

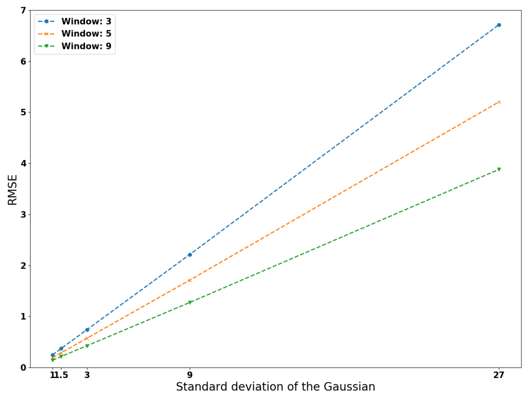
4.1.2 Smoothing data

When we are smoothing data to remove noise, there are two things we need to take into account. How well does the algorithm detect and remove noise, and how much information is lost using the smoothing algorithm.

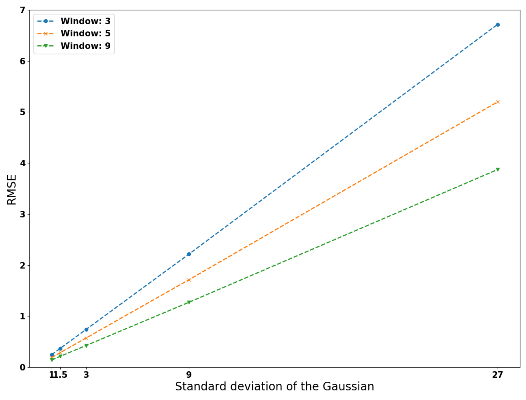
To evaluate these two conditions, we created two testing environments. The first one is used to assess the model's ability to detect and remove noise. We achieve this by applying the smoothing technique to the original data with and without noise. After smoothing the data, we calculate the RMSE between the two resulting signals. An RMSE of zero means that the smoothing approach removed any noise we added to the original signal since the resulting signals are the same. Any difference in the outputs means some noise passed through the smoothing technique, resulting in an RMSE bigger than zero. The second one analyzes the amount of information lost from the original signal. To do so, we calculate the RMSE between the original signal and the output of the smoothing technique using the same signal. The closer RMSE is to zero, the less information is lost.

Similar to the experiment developed for the missing values, we assess the smoothing capacity of the different techniques using a wide range of Gaussian noise. The Gaussian noise has zero mean, but the standard deviation changes between 1, 1.5, 3, 9, and 27. Since we do not know the specifics of the metering noise produced by Withus meters, as it is out of the scope of this dissertation, the range of values assures us different levels of noise, from values between -3 and 3 to values between -100 and 100. Besides the Gaussian noise, we also experimented with the window size used by the algorithms, either 3, 5, or 9. We did this because using different window sizes changes the smoothing degree, affecting the algorithm's overall performance. Using these two ranges of values allows the team responsible for recording the dataset to choose the combination that best suits the metering reality.

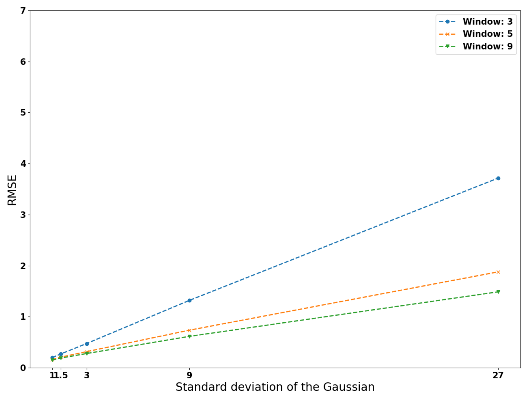
Figures 4.3a, 4.3b, and 4.3c present the results obtained by the three approaches in the first testing environment considering the various standard deviations and window sizes. Table 4.1 contains the results of the second testing environment.



(a) EWMA



(b) SMA



(c) Median Filter

Figure 4.3: Results of the data smoothing.

Model	EWMA			SMA			Median Filter		
Window Size	3	5	9	3	5	9	3	5	9
RMSE	5.82	8.15	10.92	8.01	10.11	12.83	9.95	12.58	15.61

Table 4.1: RMSE between the original signal and the output of the smothing techniques.

Since we do not know the metering device's exact noise, choosing between the solutions presented is hard. Nevertheless, to make a fair comparison between them, we compared the three solutions when they obtained similar RMSE in the second testing environment, which meant using the EWMA with window size 9, SMA with window size 5, and median filter with window size 3. Figure 4.4 exhibits the results comparing the three solutions on their noise-canceling capabilities. The median filter shows a clear advantage over SMA and better performance for bigger errors than the EWMA, justifying its use in the other NILM articles.

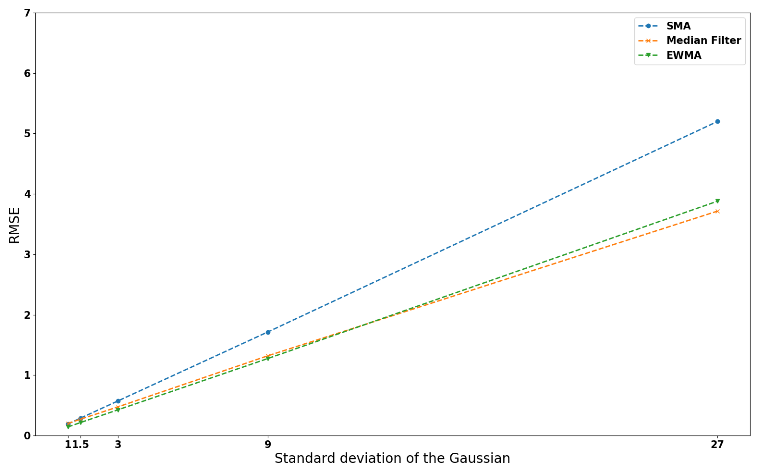


Figure 4.4: Comparison of the results of the smoothing techniques.

4.1.3 Final solutions

Recalling what we said in Section 3.2.1, Traces has a third option to fill missing values and create an evenly spaced time series using a moving average. If we think about it, this method does the same thing our complete preprocessing step does, given that it fills the missing values and applies a smoothing technique to the data. So we decided to compare the two solutions and verify which one is the best at reconstructing the original signal.

To do so, we combined the experiments of the two preprocessing steps. Using the cleaned fridge time series, we start by adding ten gaps of each size we used in the missing values experiment. After that, we distorted 4% of the signal with each type of noise we used in the smoothing experience. This creates a very irregular time series which is perfect for evaluating the solutions. The results are presented in Table 4.2 and show that our combination of filling missing values using Traces *linear* option and smoothing data using median filter reduces the RMSE by 2.02 compared to the Traces moving average, making it the option chosen.

Model	Average	Standard Deviation
Traces - Moving Average	8.03	0.85
Traces - Median Filter Combination	6.01	0.35

Table 4.2: Results of the comparison between the two final options for dataset preprocessing.

4.2 NILM models

Similar to developing a NILM model, evaluating it is not straightforward as we must consider many factors. Even after deciding which dataset we are going to use. We need to choose the appliances we will disaggregate, the amount of data to use, and how to divide said data into training, cross-validation, and testing.

As we explained in Section 3.1.4, we implemented experiments based on the UKDALE and REFIT datasets using the washing machine, dishwasher, kettle, microwave, and fridge. We opted for these appliances as they are popular for evaluating NILM models [17], [18] and represent various appliance types. The kettle and microwave are type I appliances. The fridge is a type II appliance. Furthermore, the dishwasher and washing machine are type II appliances with irregular signatures, similar to type III appliances. We excluded the type IV appliances, as we first focused on disaggregating high-consumption devices or deferrable loads.

Since we cannot afford to train a new model for each house, a model's generalization capability is crucial in its viability. This meant that the experiments created needed to evaluate this characteristic. To do so, we started by dividing the houses in the datasets into training, cross-validation, and testing sets, which ensures us that the households used to evaluate the model are never seen during training. In the UKDALE, from the five houses, we used three, leaving one for training, one for cross-validation, and one for testing. In the REFIT dataset from the twenty households, we could use at least fourteen for each appliance. This led to a division of eight houses for training, two for cross-validation, and four for testing. Table 4.3 shows the houses used by each appliance in the UKDALE dataset, and Table 4.4 shows the ones used in the REFIT.

Set	Washing Machine	Dishwasher	Kettle	Microwave	Fridge
Training	1	1	2	1	1
Cross-Validation	2	2	2	2	2
Testing	5	5	5	5	5

Table 4.3: Houses used for each appliance in the different sets for the UKDALE.

Set	Washing Machine	Dishwasher	Kettle	Microwave	Fridge
Training	1, 2, 3, 5, 6, 7, 9, 10	1, 2, 3, 5, 6, 7, 9, 10	2, 3, 4, 5, 6, 7, 8, 9,	2, 3, 4, 5, 6, 9, 10, 11	2, 3, 4, 5, 9, 19, 11, 12
Cross-Validation	11, 13, 14	11, 13, 14	11, 12	12, 13, 14	14, 15
Testing	15, 17, 19, 20	15, 17, 19, 20	13, 16, 18, 19	16, 17, 18, 19	16, 17, 18, 20

Table 4.4: Houses used for each appliance in the different sets for the REFIT.

While we were preparing the experiments, we came across another problem. Most appliances have at most a couple of activations per day. This means that if we use a week of recordings for training and another for testing, we end up with a very unbalanced dataset. Even so, this is how the NILMTK API defines the experiments. To solve this problem, we changed the NILMTK API to allow the extraction of the appliance's activations from the period of recording we chose. Using the parameters defined in Table 4.5, based on the ones proposed in [17], to detect appliance activations, we obtained a much more balanced dataset than the original, as shown by the comparison in Table 4.6. The changes made to the NILMTK were also placed in a pull request so that the other researchers could easily create a balanced dataset in their experiments.

Regarding the amount of data used to train and test our models, ideally, we would use as much data as possible. However, we had to consider the time the models took to train and evaluate, so we reduced the data used in most cases to obtain approximately 100 000 to 200 000 training examples, 30 000 to 70 000 cross-validation examples, and 200 000 testing examples. Since the models we initially proposed were not definitive and intended to change, the trade-off is worth it as we could experiment with more architectures in a shorter period.

Parameter	Washing Machine	Dishwasher	Kettle	Microwave	Fridge
Min. off time	12	0	30	1800	160
Min. on time	60	12	12	1800	1800
Padding	120	10	5	250	200
Min. on power	50	2000	200	10	20

Table 4.5: Parameters used to obtain appliance activations.

	Washing Machine	Dishwasher	Kettle	Microwave	Fridge
		Training Set			
Before	3/97	2/98	0.7/99.3	0.6/99.4	39/61
After	54/46	62/38	57/43	51/49	54/46
		Cross-Validation Set			
Before	0.9/99.1	3/97	1/99	0.4/99.6	45/55
After	44/56	40/60	60/40	58/42	48/52
		Test Set			
Before	4/96	2/98	0.4/99.6	9/91	29/71
After	57/43	58/42	49/51	68/32	53/47

Table 4.6: ON/OFF percentage before and after using appliance activation detection in UKDALE.

Once the models are ready for production and their performance is the only factor in play, they should be trained with all the available data. The number of training, cross-validation, and testing examples for each appliance is displayed in Table 4.7 and 4.8 for the UKDALE and REFIT, respectively. The UKDALE's kettle is the only appliance we could not obtain the 100 000 examples for training. However, we had difficulties reaching the aimed numbers for testing and cross-validation in the UKDALE's dishwasher, kettle, and microwave since the houses had shorter recording times. For the REFIT, given the number of houses and recording period, all of the appliances easily had the number of samples we wanted, except for the microwave, where the testing houses did not have enough activations.

Set	Washing Machine	Dishwasher	Kettle	Microwave	Fridge
Training	171 303	191 060	30 647	147 960	238 003
Cross-Validation	40 224	75 516	7 413	12 680	92 390
Testing	132 406	73 040	7 520	2 115	239 462

Table 4.7: Number of examples used on each appliance in UKDALE.

Set	Washing Machine	Dishwasher	Kettle	Microwave	Fridge
Training	127 552	118 048	110 169	117 745	130 906
Cross-Validation	39 736	63 354	87 804	45 602	70 886
Testing	223 618	238 183	286 514	107 916	297 163

Table 4.8: Number of examples used on each appliance in REFIT.

One last aspect we need to consider is the task solved by the model since, as explained in Sections 3.1 and 3.3, we developed models for ON/OFF classification and energy disaggregation. Even though these models used the same training/testing sets, the value predicted and metrics used changed. Because of these differences, we divided the experiments according to the corresponding task.

4.2.1 Classification models

The classification models were the first models we developed during the dissertation. We had no idea if any of them would work, so we started by classifying only three appliances, the washing machine, kettle, and fridge, from the UKDALE dataset. Once we considered that the models were ready to transition towards regression, we also evaluated them on the dishwasher and microwave.

Since we are solving a classification task, we decided to use the F1-Score and Mathews Correlation Coefficient (MCC) as evaluation metrics. The F1-Score was chosen given its popularity, [10], [17], [22], [92], [94], which allowed us to have an indirect comparison between our models and SOA ones. The MCC was used because it penalizes the misclassification harder than F1-Score, making it easy to understand if the model is wrongly classifying some of the appliances. The equations used to obtain the F1-Score and MCC are presented in Equations 4.2 and 4.3, respectively. The TP, TN, FP, and FN represent the true positives, true negatives, false positives, and false negatives.

$$F1 = \frac{TP}{TP + \frac{1}{2}(FP + FN)} \tag{4.2}$$

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{4.3}$$

As we already mentioned when talking about the architecture of the models in Section 3.3.1, the DeepGRU, MLP, and ResNet models went through some iterations. To understand the changes made in the models, we will now analyze the results obtained in each iteration following the models' evolution.

MLP

The MLP was used under two different feature sets, the raw data and the statistical features of the DWT, to evaluate if there is benefit in using the DWTs. The first model we proposed obtained good results using both feature sets, as shown in Tables 4.9 and 4.10. Looking at Figures 4.5, 4.6, the learning curves are smooth, indicating that the model was learning correctly. Even so, there were two exceptions, the fridge in the raw data, which overfitted the training data, and the washing machine in the DWTs that presented a poor cross-validation curve. The DWTs obtained better training and testing scores in the fridge but worst in the kettle and washing machine, making their usefulness in finding the appliance states unclear. The MCC score shows that the model struggles to correctly generalize the fridge and washing machine, given the difference between the training and test scores.

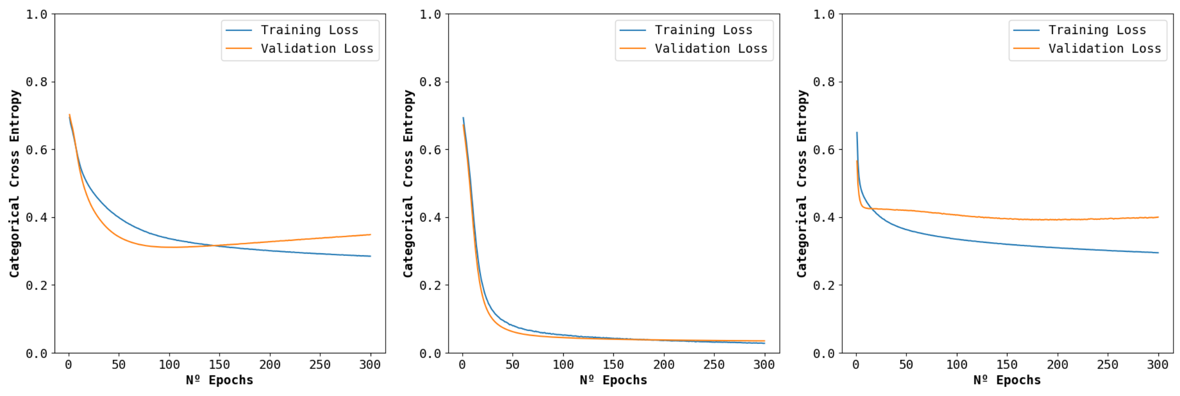
Since the results had a margin for improvement, we changed the proposed architecture in an attempt to increase the performance. After testing it, the results did not improve using both feature sets, as they were within the margin of error. Tables 4.11 and 4.12 contain the results obtained by the second architecture and Figures 4.7 and 4.8 contain the comparison between the loss evolution of the two architectures.

Set	Fridge	Washing Machine	Kettle
		MCC	
Train	0.71	0.71	0.98
Test	0.35	0.17	0.88
		F1-Score	
Train	0.87	0.87	0.99
Test	0.67	0.49	0.94

Table 4.9: Results for the first architecture of the MLP in classification using DWTs.

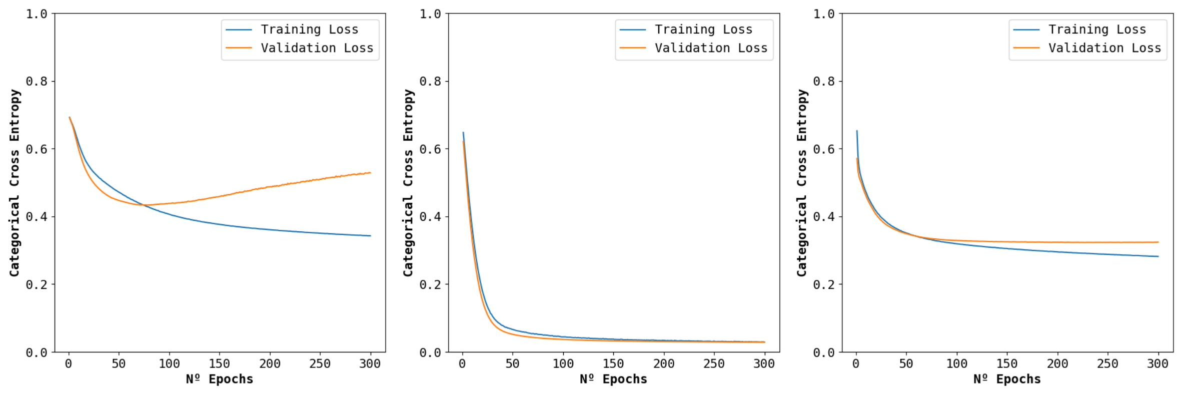
Set	Fridge	Washing Machine	Kettle
		MCC	
Train	0.57	0.75	0.98
Test	0.06	0.35	0.92
		F1-Score	
Train	0.80	0.88	0.99
Test	0.69	0.76	0.96

Table 4.10: Results for the first architecture of the MLP in classification using raw data.



(a) Fridge **(b)** Kettle **(c)** Washing Machine

Figure 4.5: MLP's first architecture's loss for the different appliances using DWTs.



(a) Fridge **(b)** Kettle **(c)** Washing Machine

Figure 4.6: MLP's first architecture's loss for the different appliances using raw data.

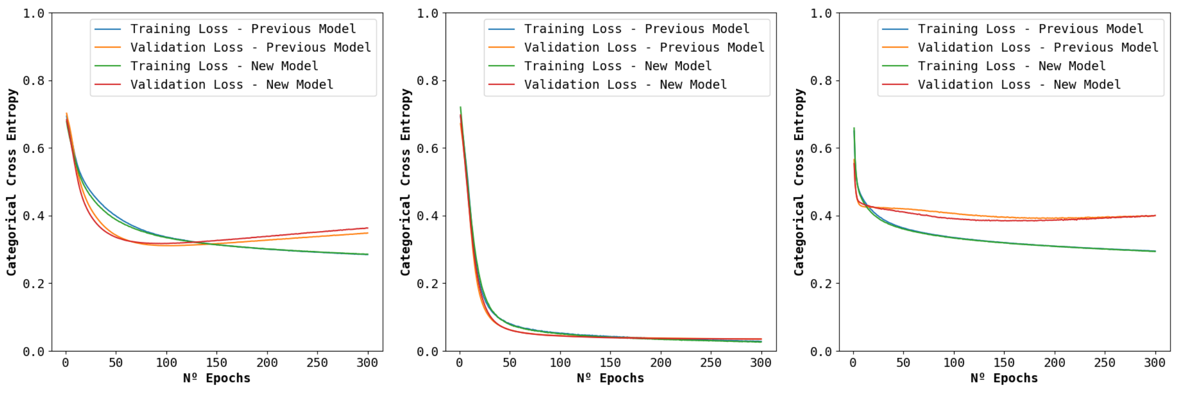
Given that the results were reasonable, even with the poor generalization in some appliances, we decided to maintain the second architecture as the final one in the classification and focus our attention on the other two networks.

Set	Fridge	Washing Machine	Kettle
		MCC	
Train	0.70	0.71	0.98
Test	0.34	0.17	0.88
		F1-Score	
Train	0.86	0.87	0.99
Test	0.66	0.47	0.94

Table 4.11: Results for the second architecture of the MLP in classification using DWTs.

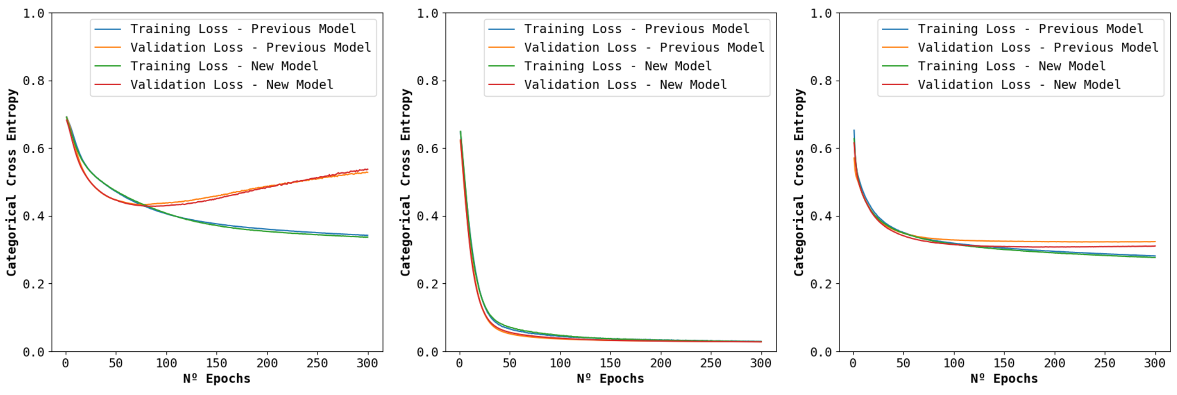
Set	Fridge	Washing Machine	Kettle
		MCC	
Train	0.58	0.75	0.98
Test	0.08	0.29	0.92
		F1-Score	
Train	0.80	0.88	0.99
Test	0.69	0.75	0.96

Table 4.12: Results for the second architecture of the MLP in classification using raw data.



(a) Fridge (b) Kettle (c) Washing Machine

Figure 4.7: MLP's first and second classification architectures' loss for the different appliances using DWTs.



(a) Fridge (b) Kettle (c) Washing Machine

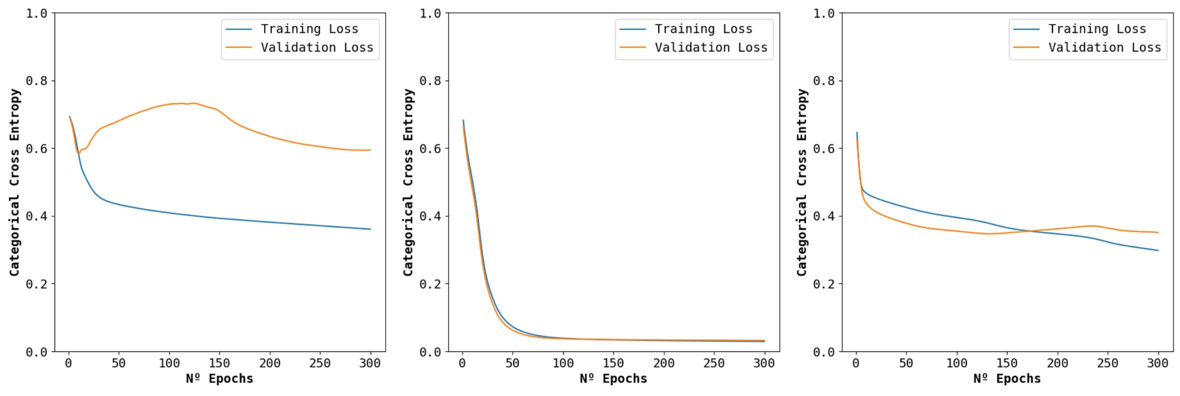
Figure 4.8: MLP's first and second classification architectures' loss for the different appliances using raw data.

DeepGRU

This model was the one that elicited the most significant doubts, as we were unsure if the reason no one had proposed deeper RNN was that it was tried and it did not work. Even so, as we can see in Table 4.13, the first results were promising, achieving similar performance to both MLP alternatives in the kettle. The learning curves displayed in Figure 4.9 show that the model learns smoothly for the kettle with some irregularities in the other two appliances. Given the MCC values, the model struggles to correctly classify the fridge and washing machine states in the training and testing sets.

Set	Fridge	Washing Machine	Kettle
		MCC	
Train	0.23	0.69	0.98
Test	0.01	0.07	0.91
		F1-Score	
Train	0.64	0.86	0.99
Test	0.69	0.72	0.96

Table 4.13: Results for the first architecture of the DeepGRU in classification.



(a) Fridge. **(b)** Kettle. **(c)** Washing Machine.

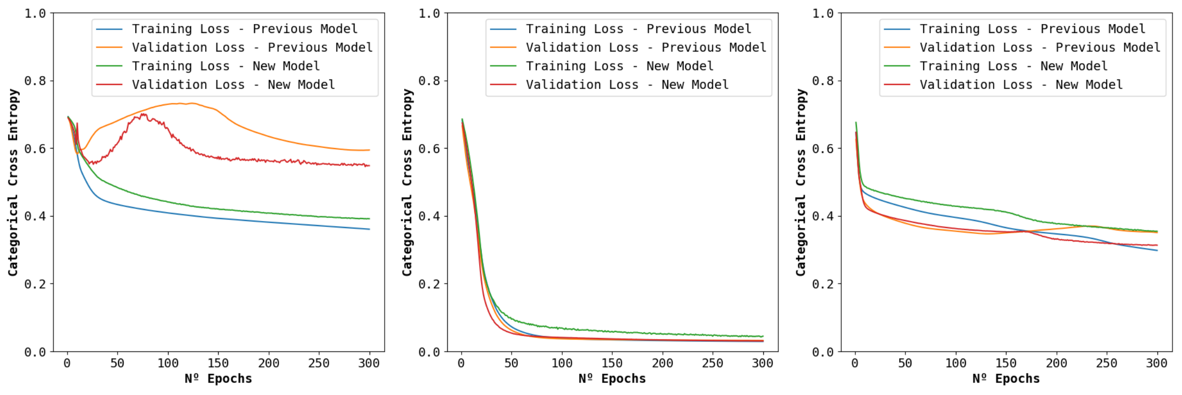
Figure 4.9: DeepGRU's first classification architectures' loss for the different appliances.

To smooth the fridges learning curve and improve the overall results, we changed the architecture to one based on blocks. The results presented in Table 4.14 show that the new architecture improved the model's performance on the fridge and washing machine training while maintaining it in the test set. When comparing the learning curves of the two models presented in Figure 4.10, it is clear that the second model's fridge and washing machine learning curves were smoother. Even so, there was still room for improvement as it did not decrease steadily.

Set	Fridge	Washing Machine	Kettle
		MCC	
Train	0.59	0.73	0.97
Test	0.01	0.07	0.92
		F1-Score	
Train	0.81	0.88	0.99
Test	0.69	0.72	0.96

Table 4.14: Results for the second architecture of the DeepGRU in classification.

Still focused on improving the results in the fridge, we changed the model's architecture again. This time we decided to add a new block to see if the model could extract longer-term



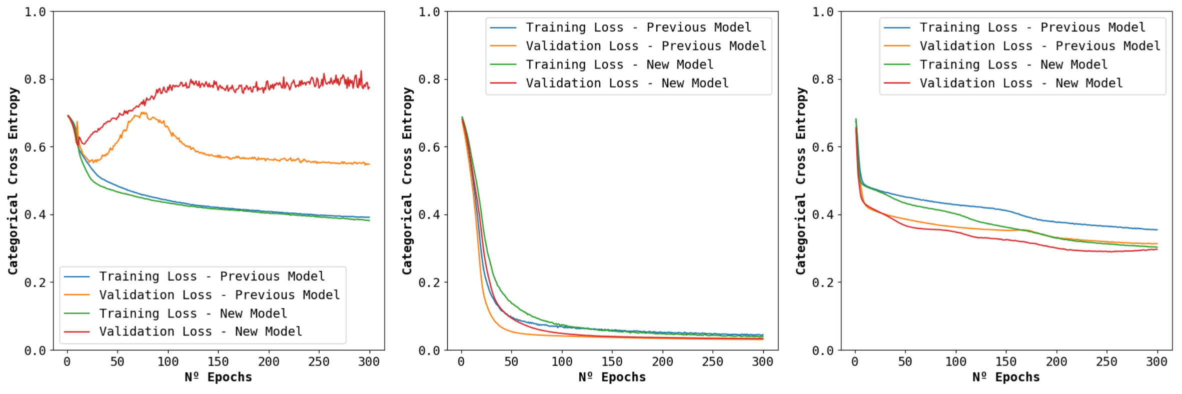
(a) Fridge. **(b)** Kettle. **(c)** Washing Machine.

Figure 4.10: DeepGRU's first and second classification architectures' loss for the different appliances.

features given the added GRU block. Unfortunately, the results did not match our reasoning as the model performance worsened the fridge's training score, as seen in Table 4.15. The worst performance is easily understood once we analyze the learning curves seen in Figure 4.11, where the model overfitted the training data. Regarding the washing machine, the new model took advantage of the extra layer, slightly improving the MCC score and presenting a better learning curve.

Set	Fridge	Washing Machine	Kettle
		MCC	
Train	0.23	0.75	0.98
Test	0.00	0.14	0.91
		F1-Score	
Train	0.64	0.88	0.99
Test	0.69	0.71	0.96

Table 4.15: Results for the third architecture of the DeepGRU in classification.



(a) Fridge. **(b)** Kettle. **(c)** Washing Machine.

Figure 4.11: DeepGRU's second and third classification architectures' loss for the different appliances.

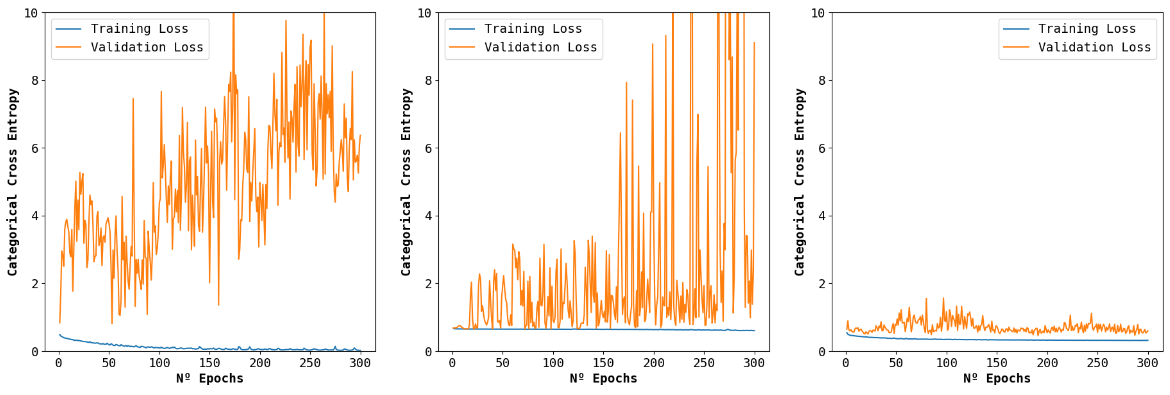
Due to time constraints, we decided that we should stop trying to improve the network in the classification task and use the best model so far, the second one, for the regression task where we would further optimize it.

ResNet

Considering the results obtained by this model in other time series classification tasks, we were expecting it to perform well with the basic architecture. To our surprise, the model was the worst classifier of all, obtaining considerably worst results in every appliance, as shown in Table 4.16. Once we analyzed the learning curves presented in Figure 4.12, the reason behind the poor performance was evident, as the model was overfitting the training data to the point of creating unstable cross-validation results.

Set	Fridge	Washing Machine	Kettle
		MCC	
Train	0.03	0.40	0.12
Test	-0.05	0.12	0.16
		F1-Score	
Train	0.45	0.68	0.67
Test	0.13	0.68	0.61

Table 4.16: Results for the first architecture of the ResNet in classification.



(a) Fridge. **(b)** Kettle. **(c)** Washing Machine.

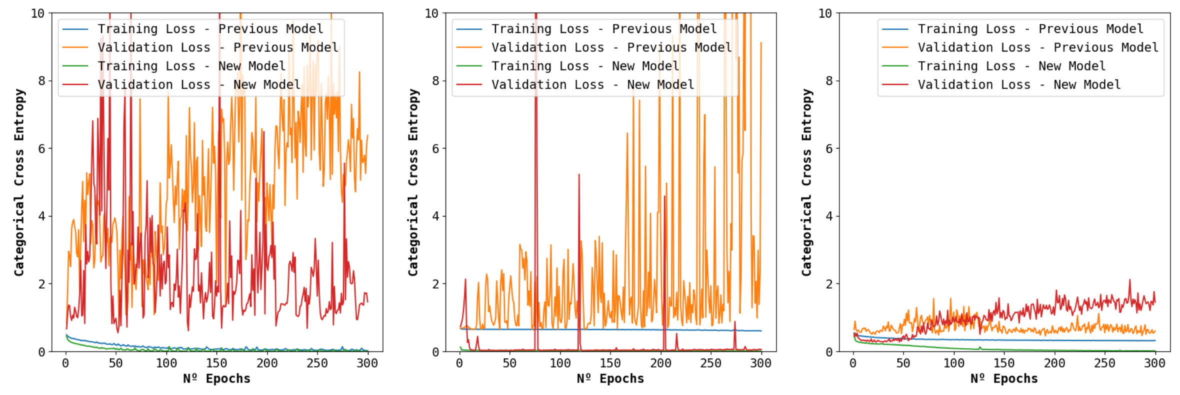
Figure 4.12: ResNet's first classification architecture's loss for the different appliances.

To solve the overfitting, we concluded that we should add a dropout layer at the end of the network. With the dropout layer, the model obtained a better F1-Score for both the training and testing sets, as presented in Table 4.17, and even though the overfitting was reduced, it was still present, as shown in Figure 4.13 and the MCC scores.

Set	Fridge	Washing Machine	Kettle
		MCC	
Train	0.34	0.75	0.98
Test	-0.31	-0.03	0.91
		F1-Score	
Train	0.66	0.89	0.99
Test	0.28	0.70	0.95

Table 4.17: Results for the second architecture of the ResNet in classification.

Since the model is based on a block structure and the dropout reduces the overfitting, we changed the architecture again, adding a dropout layer at the end of each block. Looking



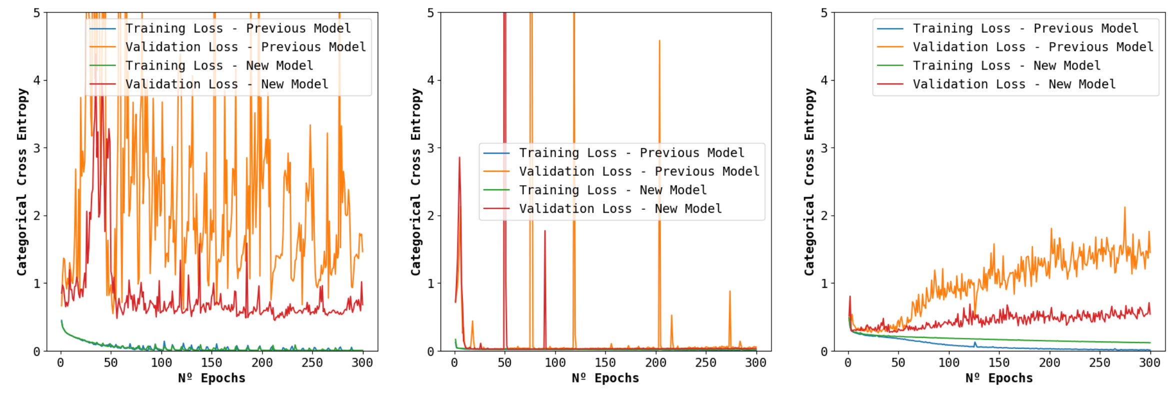
(a) Fridge. **(b)** Kettle. **(c)** Washing Machine.

Figure 4.13: ResNet's first and second classification architectures' loss for the different appliances.

at the results in Table 4.18, we can state that this architecture finally obtained results close to those from the other models. Unfortunately, by analyzing the learning curves present in Figure 4.14, it is clear that even though the learning curves are much smoother, the model is still overfitting the training data.

Set	Fridge	Washing Machine	Kettle
		MCC	
Train	0.57	0.72	0.98
Test	-0.07	-0.17	0.91
		F1-Score	
Train	0.80	0.87	0.99
Test	0.57	0.68	0.96

Table 4.18: Results for the third architecture of the ResNet in classification.



(a) Fridge. **(b)** Kettle. **(c)** Washing Machine.

Figure 4.14: ResNet's second and third classification architectures' loss for the different appliances.

At this point, we decided to no longer optimize the model for classification and use it in regression for two reasons. We were removing features through the dropout that might be useful in energy disaggregation since it is a more complex task, and we had time constraints associated with the project.

Final Comparison

After all the different iterations of the models and before adapting them to the regression task, we decided to run the complete UKDALE experiment with all the appliances and compare our models with some SOA ones. Keeping in mind that the comparison we are doing is not a direct one as the testing environment is not entirely equal, there is some truth in comparing the models as the dataset used was the same.

The models used for the comparison are the ML models proposed in NeuralNILM [17], where the author also used house 5 to test the models. After analyzing Figure 4.15, we can state that our algorithms obtained good classification results, achieving better results in all the appliances except for the fridge. Once again, this comparison needs to be done, keeping in mind that it is not a direct comparison, as the results obtained in the microwave and washing machine by the NeuralNILM models are awful compared to ours, which is not expected. Even so, the results show us that we are on the right track and that our models are a viable solution.

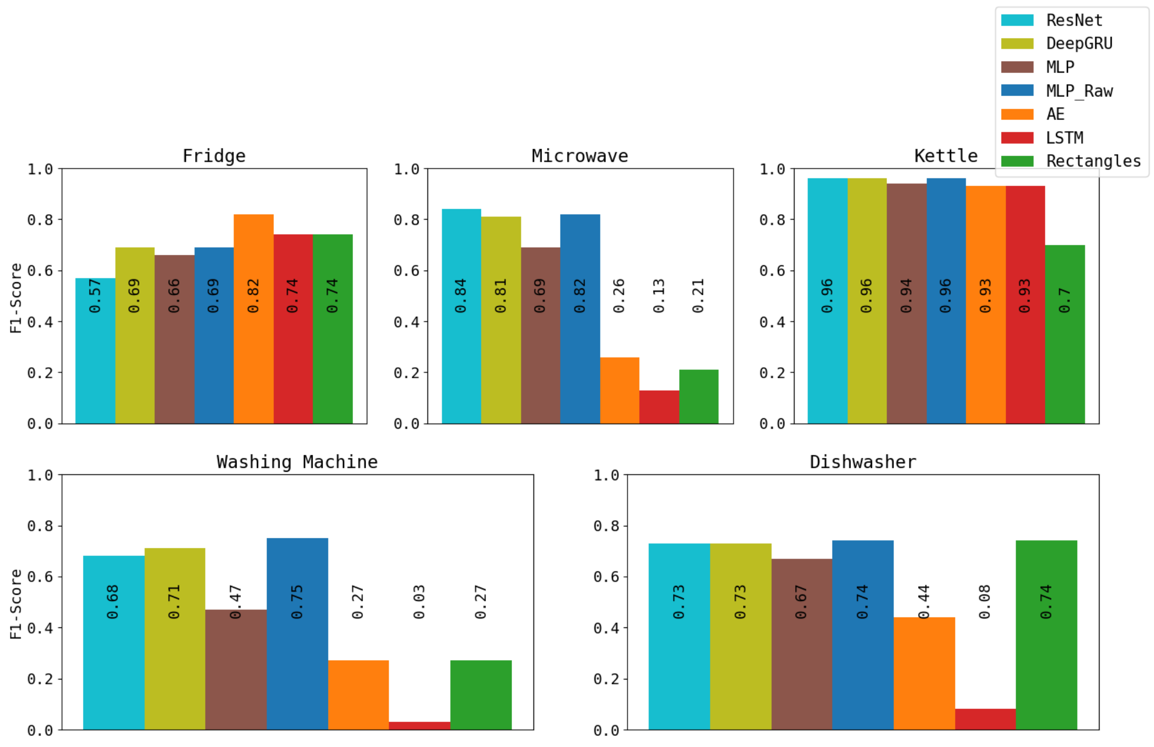


Figure 4.15: F1-Score comparison between our models and the NeuralNILM ones.

4.2.2 Regression models

With the classification models as a baseline for the regression models, we believed that the results obtained would be reasonable but would need improvements given the greater difficulty of this task. Since we are starting from a strong baseline, we began testing the models with the complete UKDALE experience, using the five appliances. This will allow us to have a better understanding of the models' performance for different appliance types.

To evaluate the models, we decided to use two metrics, the MAE and RMSE. These are two popular metrics in NILM to assess the energy disaggregation model's accuracy [17]. When combined, they allow us to understand how the models are predicting and the errors' distribution. This is possible since MAE's error penalization is linear, given it only averages the absolute error between measures, as shown in Equation 4.4, and RMSE's is quadratic, as presented in Equation 4.1. So when comparing two models, if the MAE is similar, but one has a higher RMSE, it means that the model with higher RMSE when the prediction is wrong, the error is usually higher, making it the worst classifier.

$$MAE = \frac{\sum_{i=1}^{N} |P_i - O_i|}{N} \tag{4.4}$$

Similar to what we did in the previous section, we will now follow the models' evolution analyzing the results obtained in each iteration.

MLP

Given the results obtained by this model using both DWTs and raw data in classification, we were expecting it to present good results in the disaggregation task. Unfortunately, that was not the case for the DWTs, where the results presented in Table 4.19, in some cases, are much worse than the ones using raw data displayed in Table 4.20. The learning curves of the models shown in Figures 4.16 and 4.17 justify the results obtained since they show a much more irregular training curve for DWTs where the model is not learning how to generalize to unseen houses. In the dishwasher, both models even overfit the training data.

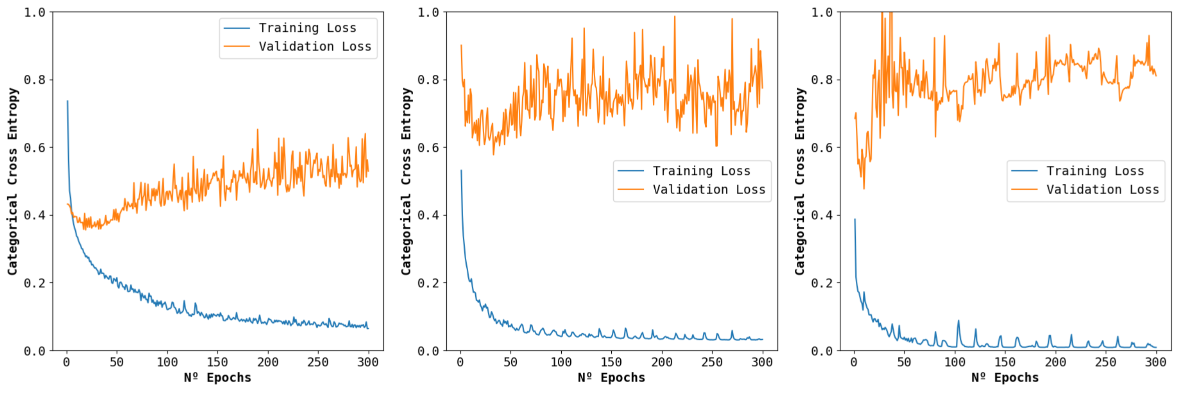
Set	Fridge	Microwave	Kettle	Washing Machine	Dishwasher
			MAE		
Train	14.03	171.18	59.81	125.64	213.70
Test	34.32	14228.10	4218.74	346.62	406.59
			RMSE		
Train	31.55	326.94	176.88	289.30	435.19
Test	45.23	14240.67	4446.26	581.98	652.93

Table 4.19: Results for the first architecture of the MLP in regression using DWTs.

Set	Fridge	Microwave	Kettle	Washing Machine	Dishwasher
			MAE		
Train	14.03	128.79	74.19	93.83	140.97
Test	30.67	312.79	200.89	318.96	350.85
			RMSE		
Train	31.75	270.36	210.80	222.31	353.70
Test	39.00	513.09	557.54	525.11	610.15

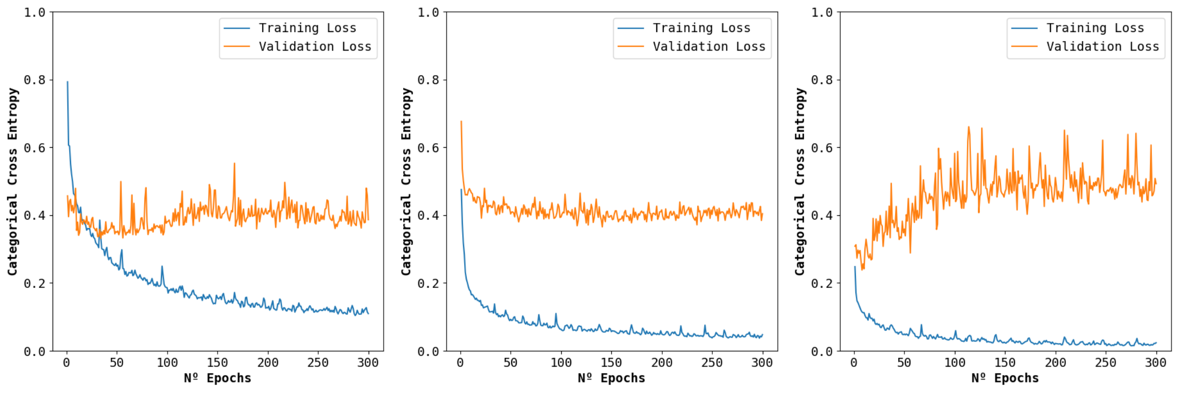
Table 4.20: Results for the first architecture of the MLP in regression using raw data.

In an attempt to improve the models' generalization capability, we changed the architecture, forcing it to create a smaller latent feature space which is then expanded to combine those features. Sadly, the results, presented in Tables 4.21 and 4.22, were still bad and in some appliances were even worse. The learning curves presented in Figures 4.18 and 4.19 did not improve compared to the previous architecture explaining the similar results obtained.



(a) Fridge. **(b)** Washing Machine. **(c)** Dishwasher.

Figure 4.16: MLP's first regression architecture's loss for the different appliances using DWTs.



(a) Fridge. **(b)** Washing Machine. **(c)** Dishwasher.

Figure 4.17: MLP's first regression architecture's loss for the different appliances using raw data.

Set	Fridge	Microwave	Kettle	Washing Machine	Dishwasher
			MAE		
Train	16.61	158.02	85.46	113.47	202.49
Test	35.18	7,230.63	3,946.96	380.23	431.15
			RMSE		
Train	33.70	312.45	206.97	269.93	436.84
Test	46.89	7254.07	4105.85	629.12	747.89

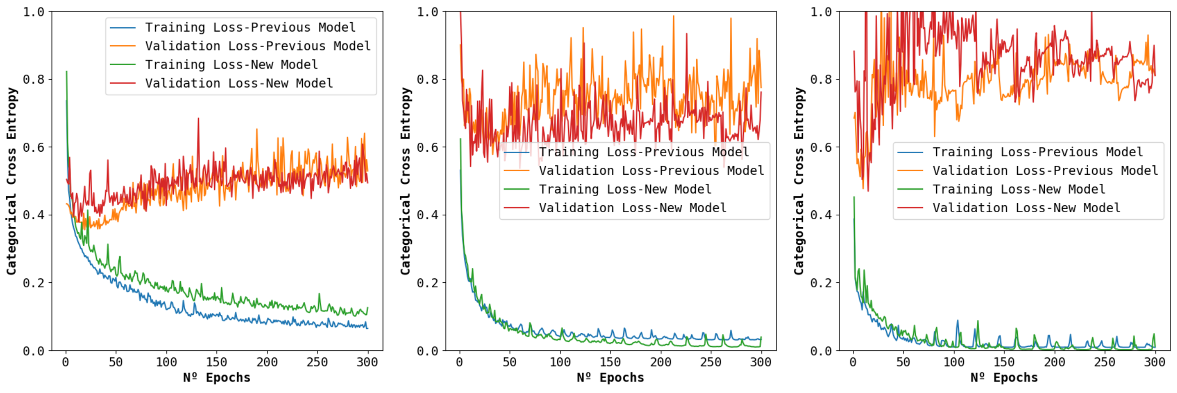
Table 4.21: Results for the second architecture of the MLP in regression using DWTs.

Given that the smaller latent feature space of the network was not enough for the model to improve the results and present smoother learning curves, a new model was proposed, which increases the size difference between the smaller feature space and the regular space. Even with this architecture, the results did not improve. As shown in Tables 4.23 and 4.24, the DWTs model still performed worse, especially in the microwave and kettle. Regarding the learning curves, the models maintained their learning patterns, as shown in Figures 4.20 and 4.21.

Even though the model still had a margin for improvement, we could not propose more architectures due to time constraints.

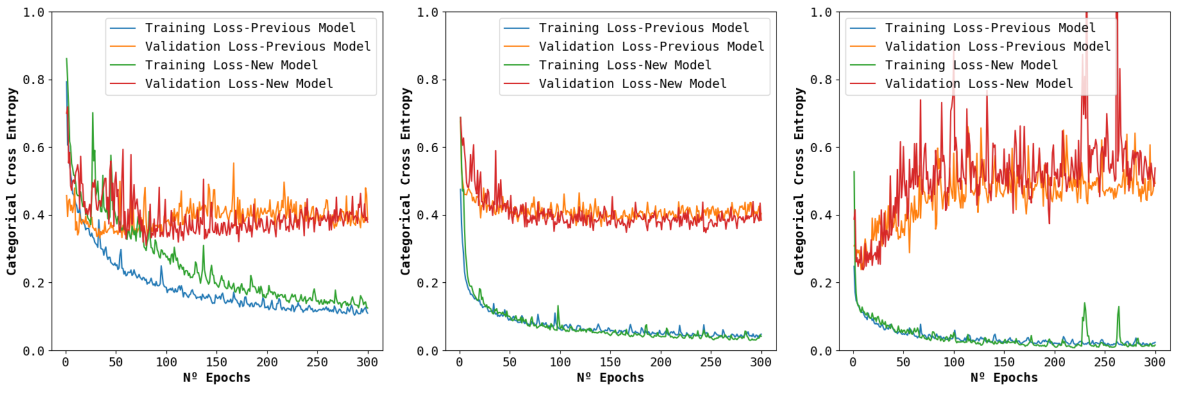
Set	Fridge	Microwave	Kettle	Washing Machine	Dishwasher
			MAE		
Train	13.33	209.31	61.34	86.08	137.16
Test	747.89	308.23	160.41	302.28	309.97
			RMSE		
Train	31.29	340.76	198.78	208.67	346.05
Test	39.49	478.80	551.01	504.79	558.22

Table 4.22: Results for the second architecture of the MLP in regression using raw data.



(a) Fridge. **(b)** Washing Machine. **(c)** Dishwasher.

Figure 4.18: MLP's first and second regression architectures' loss for the different appliances using DWTs.



(a) Fridge. **(b)** Washing Machine. **(c)** Dishwasher.

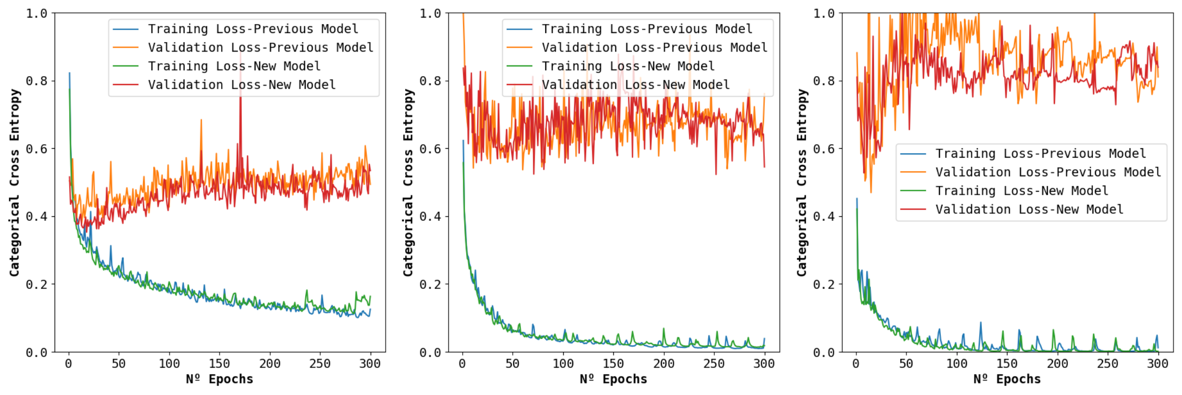
Figure 4.19: MLP's first and second regression architectures' loss for the different appliances using raw data.

Set	Fridge	Microwave	Kettle	Washing Machine	Dishwasher
			MAE		
Train	14.49	166.28	65.71	102.33	206.20
Test	33.89	776.31	5,034.14	363.60	481.18
			RMSE		
Train	32.16	313.59	196.85	252.76	471.50
Test	46.56	1,026.01	5,155.08	597.91	968.19

Table 4.23: Results for the third architecture of the MLP in regression using DWTs.

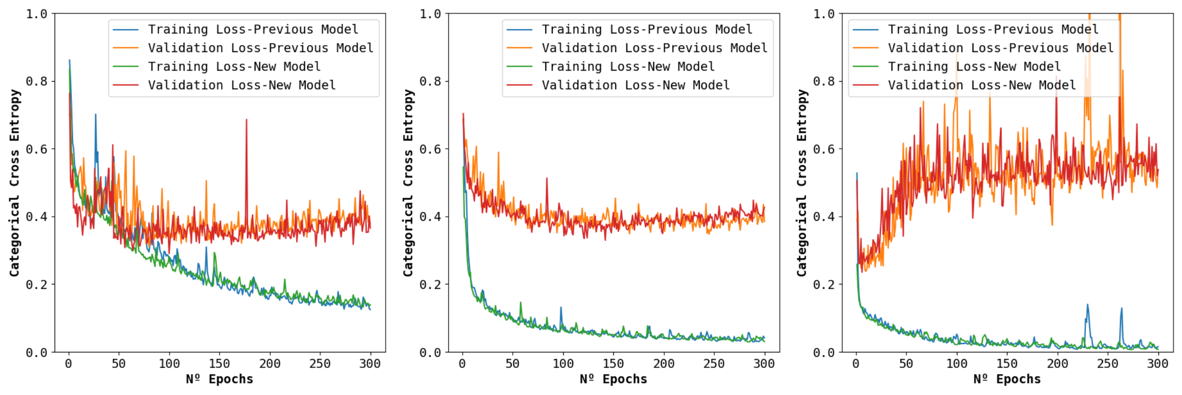
Set	Fridge	Microwave	Kettle	Washing Machine	Dishwasher
			MAE		
Train	12.47	158.29	62.79	99.31	155.92
Test	30.81	287.69	167.85	307.39	336.47
			RMSE		
Train	29.26	302.57	199.09	228.92	353.46
Test	38.40	501.95	532.98	498.81	585.78

Table 4.24: Results for the third architecture of the MLP in regression using raw data.



(a) Fridge. **(b)** Washing Machine. **(c)** Dishwasher.

Figure 4.20: MLP's second and third regression architectures' loss for the different appliances using DWTs.



(a) Fridge. **(b)** Washing Machine. **(c)** Dishwasher.

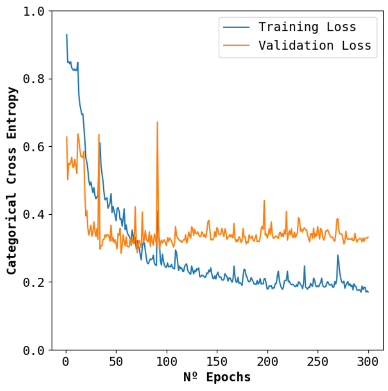
Figure 4.21: MLP's second and third regression architectures' loss for the different appliances using raw data.

DeepGRU

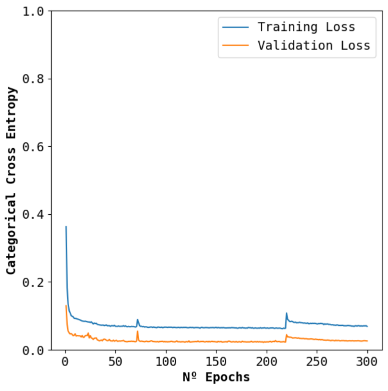
Since the results obtained by the last architecture of the classification model were not the best, we opted by using the second architecture of the classification model for regression. The results obtained were good, as shown in Table 4.25, but we could see that the fridge, washing machine, and dishwasher were not obtaining smooth cross-validation learning curves when looking at Figure 4.22.

Set	Fridge	Microwave	Kettle	Washing Machine	Dishwasher
			MAE		
Train	14.79	116.04	72.15	71.91	101.42
Test	35.28	309.12	185.71	335.32	198.55
			RMSE		
Train	35.12	247.14	212.74	185.68	261.55
Test	41.18	535.13	571.42	572.50	363.47

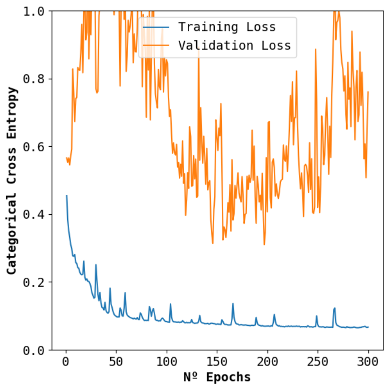
Table 4.25: Results for the first architecture of the DeepGRU in regression.



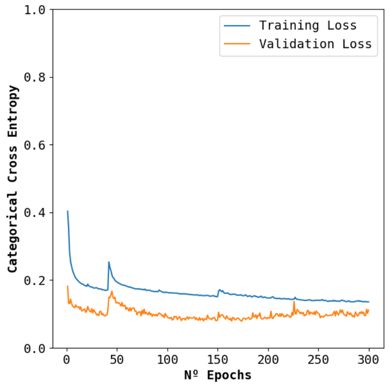
(a) Fridge.



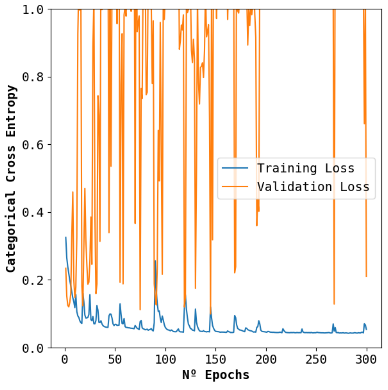
(b) Kettle.



(c) Washing Machine.



(d) Microwave.



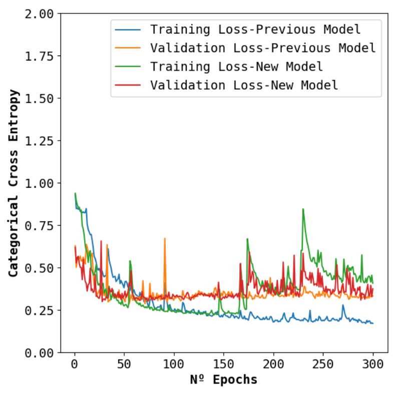
(e) Dishwasher.

Figure 4.22: DeepGRU's first regression architecture's loss for the different appliances.

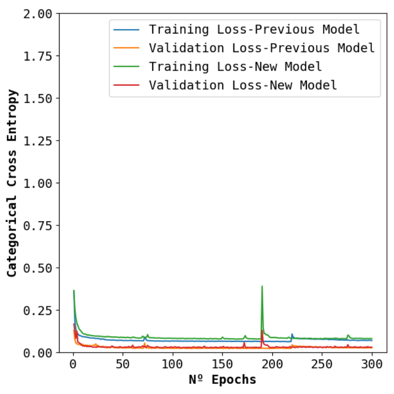
To improve the learning curves on the cross-validation, we decided to change the architecture, focusing on increasing the number of GRU layers. When tested, the results presented in Table 4.26 showed that the architecture obtained similar performance to the previous. Even so, when looking at the learning curves displayed in Figure 4.23, the results were worst as the new architecture was unstable and unable to finish the training for the washing machine.

Set	Fridge	Microwave	Kettle	Washing Machine	Dishwasher
			MAE		
Train	15.34	115.40	105.99	220.66	139.14
Test	34.95	282.05	205.18	281.95	225.13
			RMSE		
Train	31.86	226.99	225.18	363.05	281.44
Test	41.09	511.36	577.68	373.52	418.21

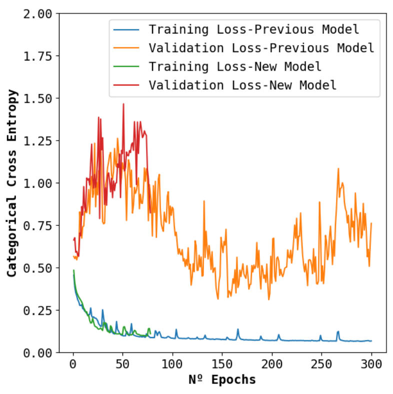
Table 4.26: Results for the second architecture of the DeepGRU in regression.



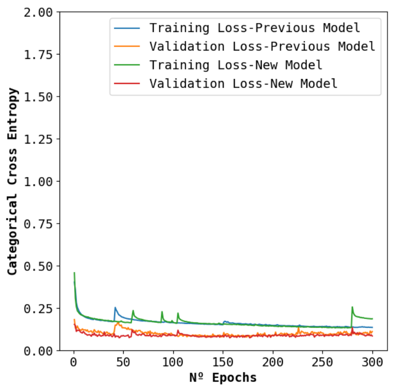
(a) Fridge.



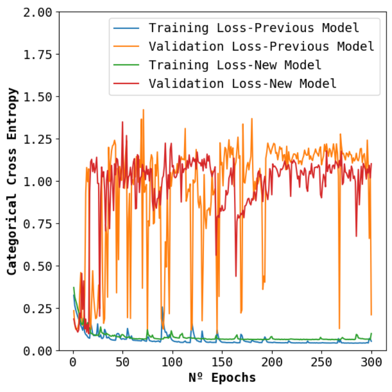
(b) Kettle.



(c) Washing Machine.



(d) Microwave.



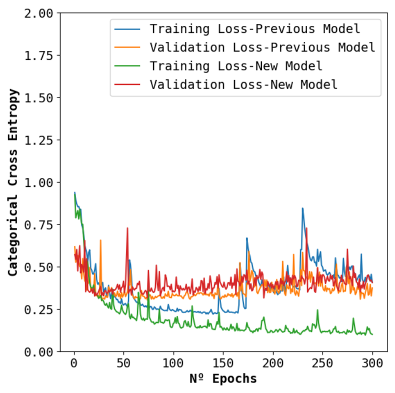
(e) Dishwasher.

Figure 4.23: DeepGRU's first and second regression architectures' loss for the different appliances.

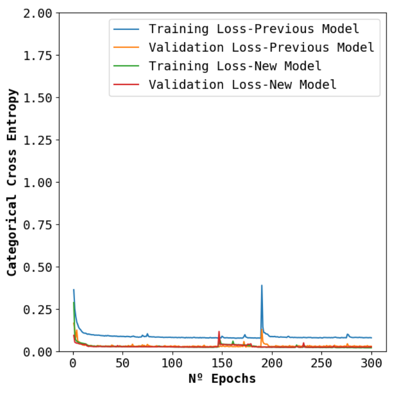
Given the failure of the previous architecture, we decided to track back to the original block structure and change its blocks. The new architecture improved the results in some appliances, as shown in Table 4.27, obtaining smoother learning curves in all appliances as presented in Figure 4.24.

Set	Fridge	Microwave	Kettle	Washing Machine	Dishwasher
			MAE		
Train	12.26	106.88	63.02	69.06	275.11
Test	26.05	284.14	161.01	352.63	251.62
			RMSE		
Train	29.74	246.36	208.63	187.97	110.92
Test	36.02	499.60	562.80	584.12	453.46

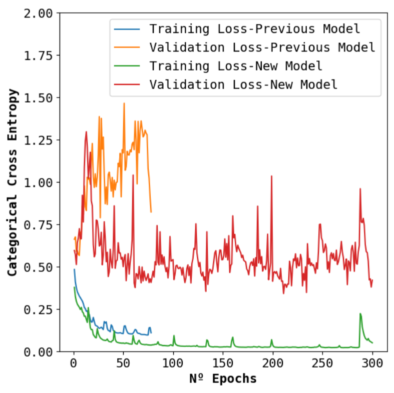
Table 4.27: Results for the third architecture of the DeepGRU in regression.



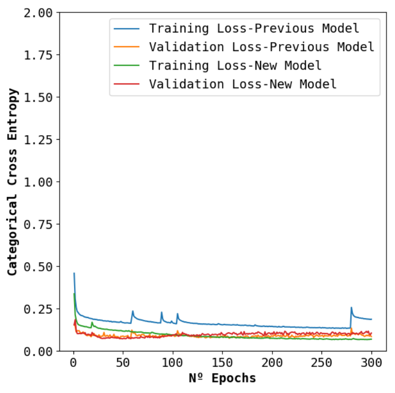
(a) Fridge.



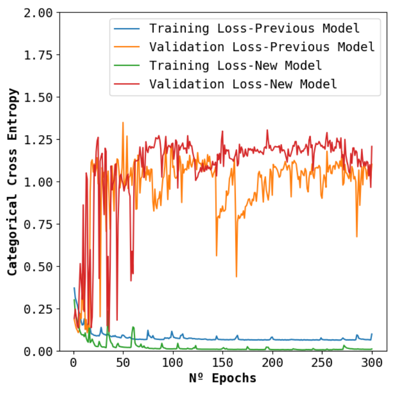
(b) Kettle.



(c) Washing Machine.



(d) Microwave.



(e) Dishwasher.

Figure 4.24: DeepGRU's second and third regression architectures' loss for the different appliances.

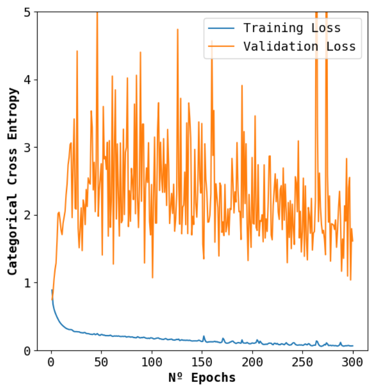
Similar to the MLP, this network still had a margin for improvement, but we could not propose more architectures due to time constraints.

ResNet

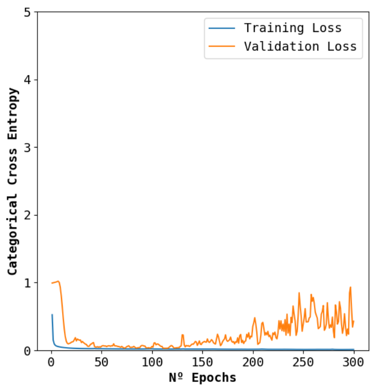
After seeing both the MLP and DeepGRU struggling to obtain good results, we decided to start the ResNet model for energy disaggregation with the original architecture, expecting it to use the features removed by the dropout to obtain better results than the other two architectures. Unfortunately, the results presented in Table 4.28 do not correspond to our reasoning as they are very similar to the other models' results. Analyzing the learning curves displayed in Figure 4.25 helps us understand the results. The model presents an inconsistent cross-validation error, failing to learn how to generalize and overfitting the training data in the fridge and microwave.

Set	Fridge	Microwave	Kettle	Washing Machine	Dishwasher
			MAE		
Train	45.00	175.21	82.76	333.70	313.39
Test	55.03	315.88	246.38	361.12	331.32
			RMSE		
Train	55.55	294.79	225.05	475.94	422.56
Test	58.23	502.18	589.28	556.50	589.51

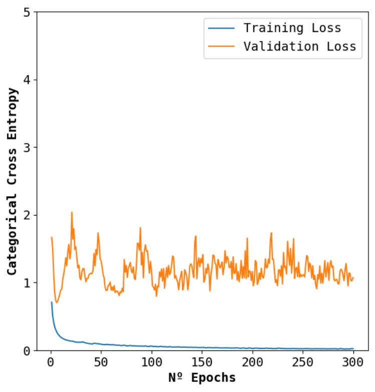
Table 4.28: Results for the first architecture of the ResNet in regression.



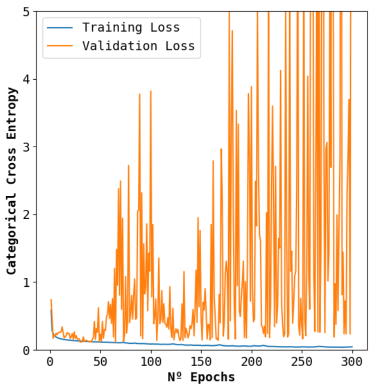
(a) Fridge.



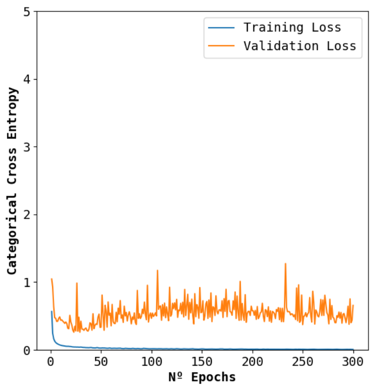
(b) Kettle.



(c) Washing Machine.



(d) Microwave.



(e) Dishwasher.

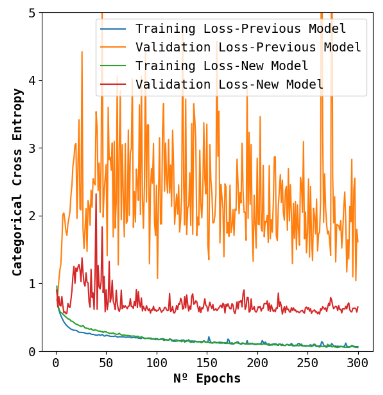
Figure 4.25: ResNet's first regression architecture's loss for the different appliances.

Given that the model was overfitting the training data, we experimented with the last classification architecture containing multiple dropout layers. The results presented in Table 4.29 show improvements, reducing the errors for every appliance. Even so, when looking at the learning curves presented in Figure 4.26, we could see that the network had smoother curves for some appliances like the fridge and washing machine while having irregular curves for the kettle and microwave.

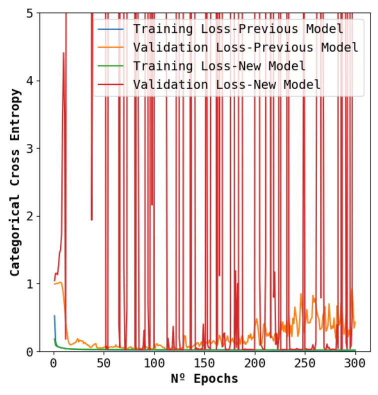
Set	Fridge	Microwave	Kettle	Washing Machine	Dishwasher
			MAE		
Train	34.92	121.30	64.93	296.48	153.84
Test	53.05	298.12	181.03	332.06	326.29
			RMSE		
Train	50.99	260.50	217.59	473.97	327.67
Test	60.64	534.49	586.61	491.29	703.73

Table 4.29: Results for the second architecture of the ResNet in regression.

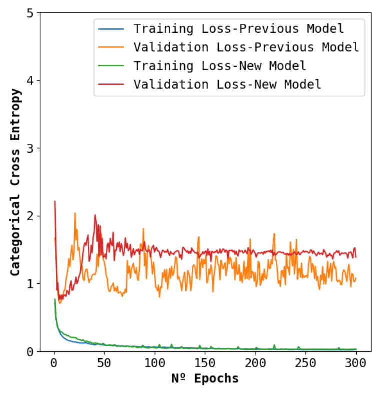
Since the model benefited from the lack of dropout between layers for the kettle and microwave, we decided to reduce the dropout value to see if the learning curves stabilized. The results presented in Table 4.30 show that the dishwasher was the only appliance that benefited from the new architecture as its performance improved substantially. As for the learning curves presented in Figure 4.27, the network's objective was not fulfilled, given that



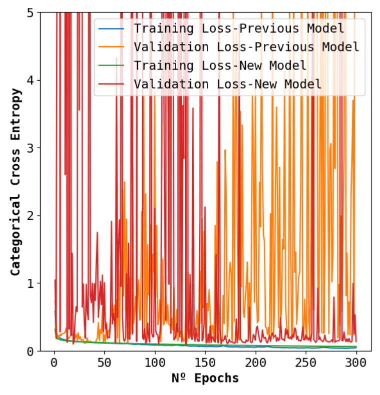
(a) Fridge.



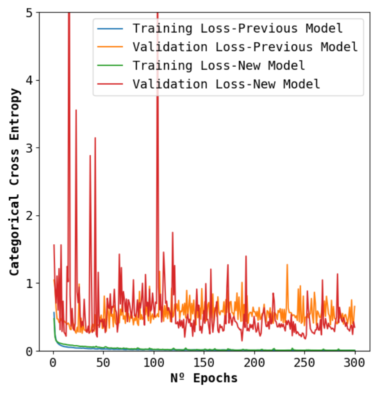
(b) Kettle.



(c) Washing Machine.



(d) Microwave.



(e) Dishwasher.

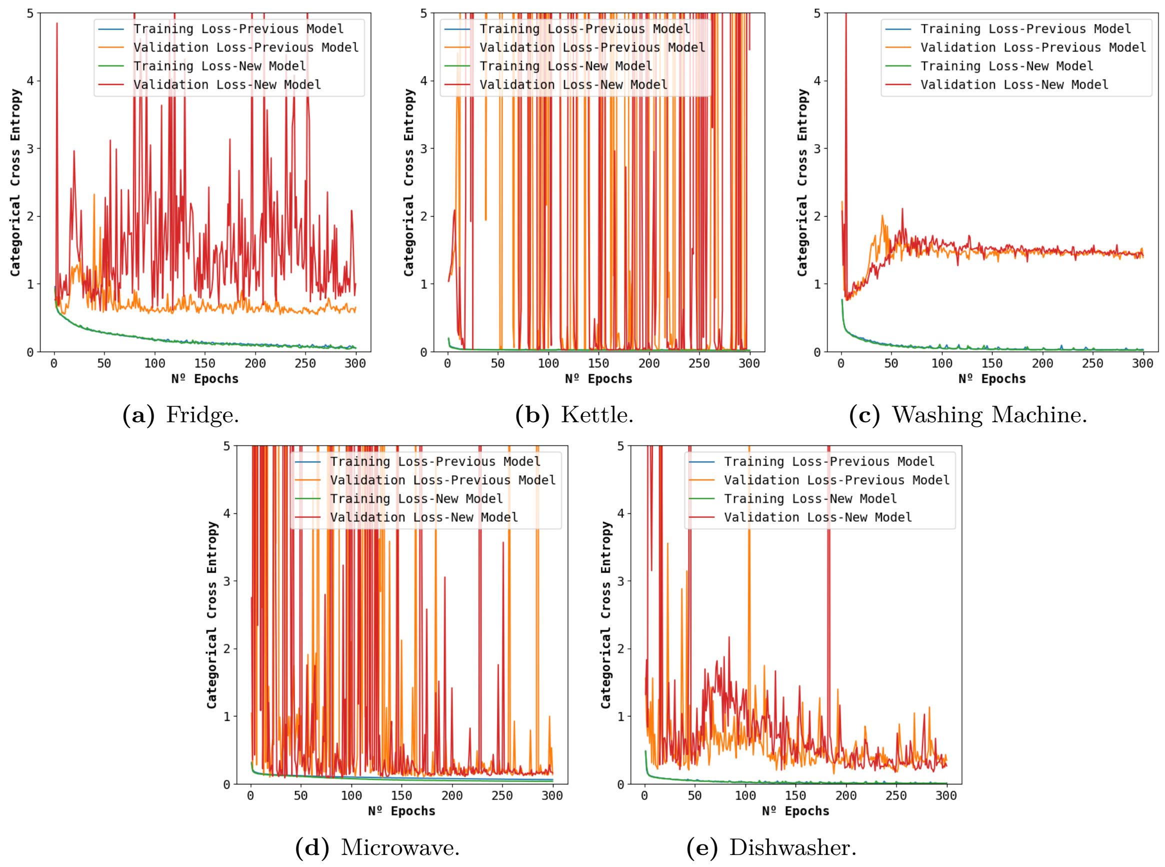
Figure 4.26: ResNet's first and second regression architectures' loss for the different appliances.

all the appliances present irregular learning curves, except for the washing machine.

Set	Fridge	Microwave	Kettle	Washing Machine	Dishwasher
			MAE		
Train	39.10	125.20	63.42	282.36	158.37
Test	54.03	309.04	205.45	328.65	223.04
			RMSE		
Train	51.36	264.52	213.67	434.82	303.68
Test	58.74	543.33	607.18	486.84	414.62

Table 4.30: Results for the third architecture of the ResNet in regression.

For the same reason presented in the two previous networks, we could not experiment with other architectures using the last one as the final architecture.



(a) Fridge. **(b)** Kettle. **(c)** Washing Machine.

(d) Microwave. **(e)** Dishwasher.

Figure 4.27: ResNet's second and third regression architectures' loss for the different appliances.

4.2.3 SOA comparison

After several iterations in the models' architecture, we had reached our time limit for searching for new architectures. However, we had only evaluated our models against each other. To compare the performance of our models with some of the SOA ones, we used three models available in NILMTK-Contrib, which we slightly modified to use the same data as ours. The models chosen are the Seq2Point, Seq2Seq, and dAE, which we already mentioned in Section 2.7. We opted for these models because, at the time of implementation, they were the best models available in NILMTK-Contrib, obtaining SOA results as shown in their papers [17], [18].

Following the experiments we described at the beginning of this section, we used the UKDALE and REFIT datasets to evaluate the models. Since we wanted to understand which model was the overall best, we needed to have an error measure that allowed us to use the errors of the different appliances together without considering the energy consumed by each one. The error measure used was the Normalized Root Mean Square Error (NRMSE), which calculates the regular RMSE and divides it by the maximum registered value of the original time series, as shown in Equation 4.5. With this measurement, the errors from the fridge and kettle are comparable, even though when they are on, one consumes more than five times what the other consumes.

$$NRMSE(P, O) = \frac{\sqrt{\frac{\sum_{i=1}^{N}(P_i - O_i)^2}{N}}}{\max O} \tag{4.5}$$

After running both experiments, each appliance had five NRMSE measurements, each related to a specific house used on the test sets. We used the 25 measurements of each model to create a box plot and understand the average models' performance and its conciseness. The resulting box plot is presented in Figure 4.28, where our models, except for the MLP using DWTs, obtained better results than the Seq2Seq and dAE. As for the Seq2Point, even though they cannot beat it, our models obtain just a slightly worse performance. This is positive given that our models are doing real-time regression while Seq2Point has a 30 min delay. Analyzing further the results, we can state that DeepGRU reduced the NRMSE of the dAE, the real-time model, on average by 49% and reduced the Seq2Seq error by 50%. When comparing the DeepGRU with Seq2Point, our model increases the error on average by 17%.

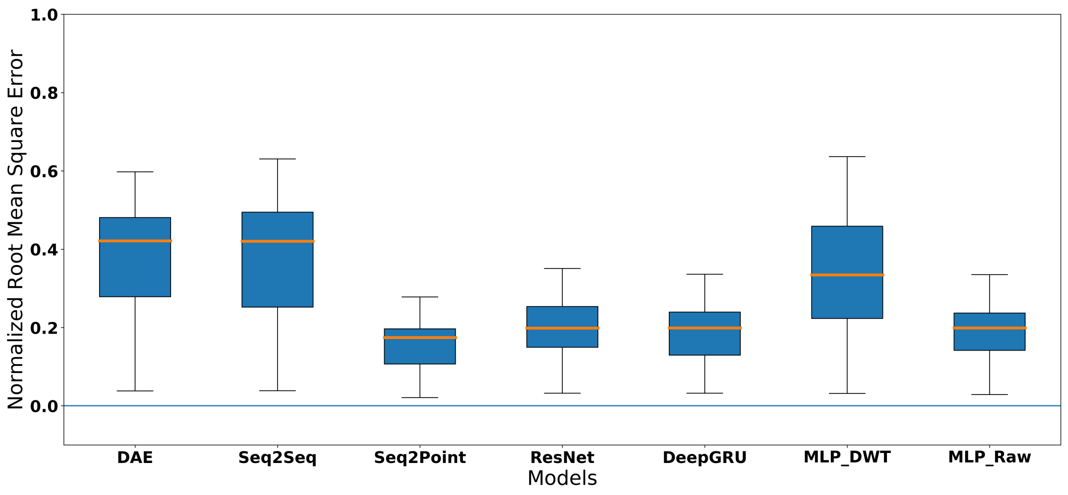


Figure 4.28: NRMSE box plots' comparison between our models and the SOA NILMTK models.

4.2.4 Transfer learning

All of our tests evaluate the ability of a model to generalize to unseen data. However, we are training and testing in the same dataset. Suppose we could use an already existing large dataset to train our models and, after that, do a quick training in a smaller Portuguese dataset to adapt to the Portuguese usage patterns. In that case, we could save Withus a substantial amount of money in the dataset recording. To evaluate the viability of this setting, we implemented a transfer learning experiment that assessed the added benefit of training in one dataset and then, using only a small sample of the second dataset, train the models again. After the second training, we evaluate the models on the remaining houses of the second dataset.

The data used in the experiments was the same we presented at the beginning of this section, and we tested with both the UKDALE and REFIT as dataset one and two. To better understand the training and testing procedure, Figure 4.29 presents the example where the UKDALE is the first dataset and REFIT is the second dataset. The training set used is the training and cross-validation data presented in Tables 4.3 and 4.4, and the test set is the test data of the same tables.

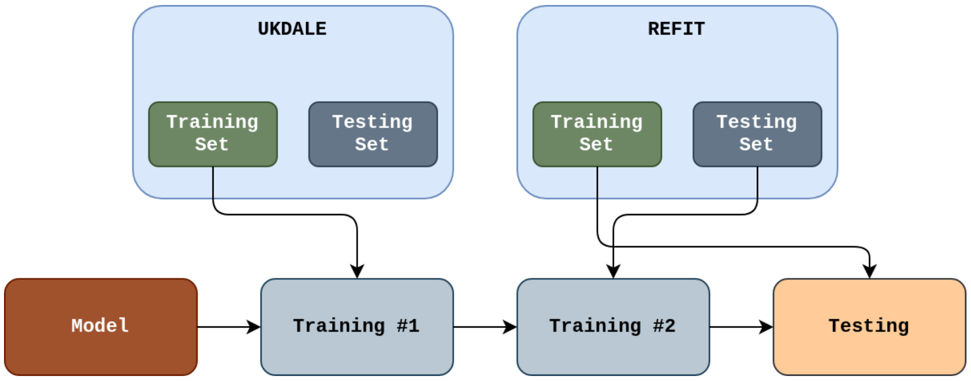


Figure 4.29: Transfer learning scheme using UKDALE as dataset one and REFIT as dataset two.

In this experiment, we only considered the Seq2Point, ResNet, and DeepGRU models. We chose these models because they obtained the best results, and Seq2Point and ResNet are composed of convolutions, which already benefit from transfer learning in various areas such as image recognition.

To understand how much the second training impacted the models' performance, we tested the models with and without it. If there is a reasonable difference in the results obtained, we can state that the models benefited from the second training step.

Similar to how we presented the results of the SOA models, we created box plots with the models' performance with and without the second training step. The results are shown in Figure 4.30, where the models with the second training, named *Transfer*, appear to benefit a little from it, presenting a smaller distance between quartiles and smaller median error. This means that the transfer learning generated more consistent models better adapted to the target houses when applied. Even so, the results are not good enough to justify the viability of using such an environment to train our models.

The model that benefited the most on average from the transfer learning was Seq2Point reducing the error by 27%, followed by the DeepGRU with 17% and ResNet with 0.9%. The ResNet results indicate that the smaller second training step was insufficient for the network to adapt to the new dataset. This is explained by the network's size and the overfitting we experienced during training, which requires more epochs to adapt the model to the different patterns on the second dataset. Even so, we expected a more significant improvement given that the network mainly uses convolutions. The DeepGRU results were positive, presenting a middle term between architectures and proving that a network using GRUs benefits from the weights trained on a different dataset.

Even though the models benefited from the transfer learning as they reduced the errors obtained when they were trained on the smaller subset of the second dataset. The critical question we need to ask is how much worse the model with transfer learning is compared

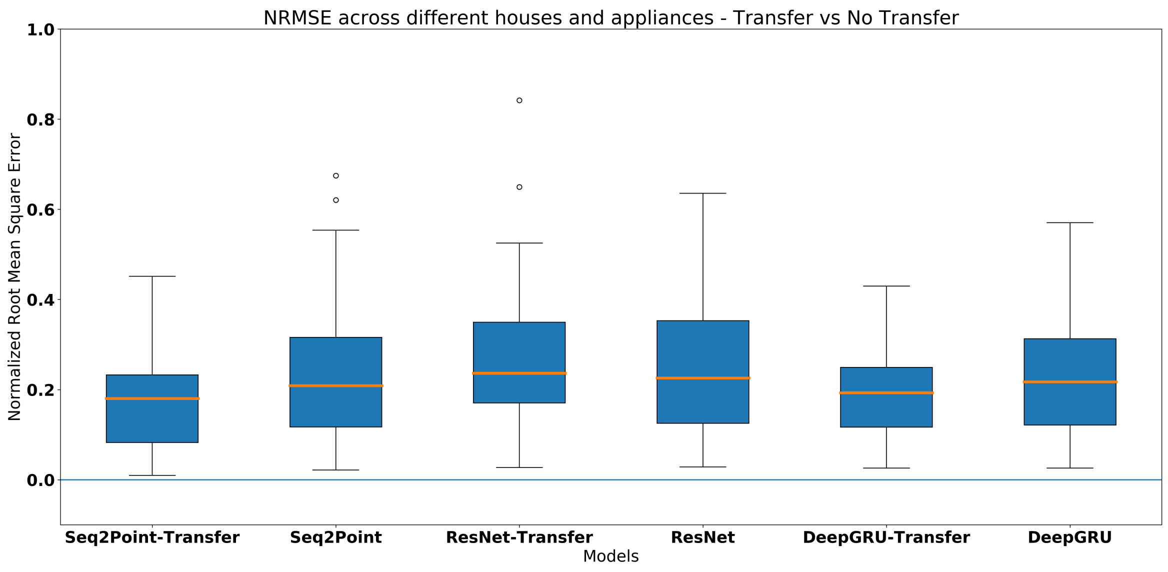


Figure 4.30: NRMSE box plots' comparison between models with and without the second training step.

to a model trained and evaluated on the same dataset, as we did in the SOA comparison. To answer this question, we created box plots with the NRMSE obtained by the models trained on the SOA comparison and those that went through transfer learning, as presented in Figure 4.31. After careful analysis, we can state that the models trained and evaluated on the same dataset have smaller distances between quartiles, which indicates that they are more consistent than the transfer learning models. Even so, when we look at the model's median, there is not a significant difference between transfer and regular models, indicating a similar average performance.

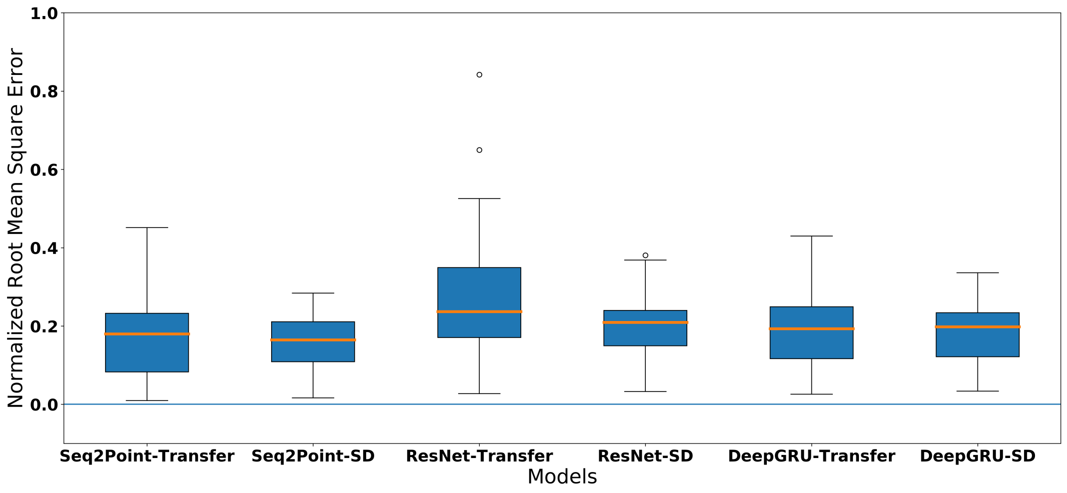


Figure 4.31: NRMSE box plots' comparison between models with transfer learning and trained and evaluated on the same dataset.

Analyzing the average error between the two types of model, the regular models (trained in SOA comparison), and models with transfer learning, we can state that the average error increases by 6% for the Seq2Point, 3% for the DeepGRU, and 21% for the ResNet, when using transfer learning.

CHAPTER 5

Conclusion

Looking back at the path we followed to get here. It is clear that it was challenging and that there were many shortcuts we missed. Even so, by writing this dissertation, we hope to give anyone trying to follow the same path a map pinpointing all the shortcuts, dead ends, and possible future routes we found.

This project started from the idea that we could develop a NILM system using ML. How we would achieve it, what were the requirements, and what was the path to follow were all questions without an answer. After reading many papers, the dots started connecting, and the outline for the system was created. We concluded three things. That we needed to make a new dataset, as no other dataset had the characteristics we needed. We should use the NILMTK library to facilitate the development of our models. And that using ML models was the most viable solution.

Developing a new dataset expanded the work proposal since we needed to implement a preprocessing step to fill missing values and remove noise and a conversion step, to transform our dataset in the NILMTK data format. The best results in the preprocessing were obtained using the library Traces to fill missing values with the linear option and using a median filter with a window size of three to remove noise.

As for the models, since we had no previous work developed in this area, we decided that the best option was to explore various research directions. We proposed three models using different architectures, layer types, and features, DeepGRU, ResNet, and MLP. The three models were firstly implemented for ON/OFF classification and later adapted to energy disaggregation going through several iterations in the process. Once they were evaluated against some of the SOA models, Seq2Point, Seq2Seq, and dAE, we concluded that the ResNet, DeepGRU, and MLP using raw data beat the real-time SOA model, the dAE. Regarding the SOA model with a 30 min delay, Seq2Point, the models obtained close results even though they could not beat it. Comparing our models between them, the DeepGRU presented itself as the most promising architecture. The MLP using DWTs obtained good results in classification but had a terrible performance under energy disaggregation. This leads us to believe that the statistical features of a DWT are not enough to obtain the energy consumed by the appliance but are a good indicator of its activation. Regarding the ideal NILM model, we believe that there is still work to be developed before we reach it, although we have already made relevant progress towards it. Our models can correctly classify type one appliances from houses never seen during training and classify 20 appliances simultaneously as they take at most 0.9 ms per sample.

Regarding the viability of using transfer learning to train our models, the results show close performance to models trained and tested on the same dataset. Even so, the results are not conclusive as this might be because the datasets used are from the same country and their differences are not as notorious as between countries. To understand the benefit of transfer learning, we should have used the REFIT dataset as the first dataset and the Withus dataset as the second one. Unfortunately, with time being a limiting factor, we could not perform this experiment.

Besides, given that we did not have the Withus dataset, we could not study the benefit of using the multiple electrical features we initially wanted.

Concerning the implementation, since NILMTK did not allow training of models for ON/OFF classification, we had to spend some time developing a similar API that would allow the models to be trained and tested for this task, and at the same time, be easily transferable to energy disaggregation. Once we transferred the models to energy disaggregation, we found that the NILMTK API was not ready to use appliance activation detection to generate balanced datasets. So we also developed the code that would allow us to use it. The resulting code was made part of a pull request to the NILMTK GitHub so that anyone could use it.

One of the most significant difficulties we found throughout the dissertation was the lack of documentation regarding NILMTK, as most of our time when creating the codebase was discovering the available methods. This sometimes resulted in inefficient solutions that were later corrected because we found a better way to do it. One other setback during development was the time the models took to train. Even after reducing the dataset size, for 300 training epochs, the ResNet took 14 hours to train a single appliance. With five appliances to evaluate, the complete ResNet testing took three days. The use of a reduced dataset also causes problems in the models' performance. Training the models in the entire dataset would most likely change the results as the ResNet especially would not overfit the training data so quickly.

These problems slowed the development. Even so, the final results were a solid codebase and three new viable NILM models. Therefore, the dataset and models developed are expected to represent the challenges faced when creating a NILM dataset or model from scratch. So, hopefully, the ideas discussed here can be an example of the problems to keep in mind when starting a NILM model, the possible solutions, and what mistakes to avoid, helping the newcomers by reducing the learning curve of the different libraries.

Since the beginning of the dissertation, we knew that at the end, there would be work left to be done and new research directions to follow.

The models we proposed used only electrical features to do both the ON/OFF classification and energy disaggregation. This means that we left out non-traditional features, which have proven to increase the model's performance [59]. Some of the non-traditional features that we would have liked to experiment with are:

- External temperature;
- Time dependencies (hour of the day, day of the week, season);
- Household inhabitants profile.

Besides non-traditional features, we would like to experiment with feature extraction libraries for time series such as TSFresh [95] and ROCKET [96], which presented promising results in other classification tasks.

Regarding the models, there are still many research directions to follow. For example, we could use multiple convolutions with different strides and sizes on the input to obtain different resolutions, study the benefit of attention mechanisms, and experiment with a Committee Decision Mechanism that combines the output of various models. We would also like to implement subsequent removal of appliance signatures from the aggregated data or even evaluate the benefit of pretraining on unlabeled data for the house we want to classify.

As for DWTs, we only scratched the surface. Besides using their statistical features, we can feed them directly to the DeepGRU or Seq2Point. Given its multiresolution capabilities, it might help these models increase their performance.

Lastly, we cannot forget one of the most important things left to develop. Even though we created the NILM models and obtained good results, we do not have a way to show the household owners their energy consumption. So the last thing we need to develop is a tool that consumes the energy disaggregation data and organizes it into actionable feedback that actively stimulates energy-saving practices.

One other idea is the study of the models on a per appliance basis. Instead of developing a generally good model, we create a model for the general appliance type, for example, an architecture for washing machines and dishwashers and another for microwaves and kettles. Since, as we saw when training our models, some changes we did improve some appliances' results while worsening others.

References

[1] A. Faustine, N. H. Mvungi, S. Kaijage, and K. Michael, *A survey on non-intrusive load monitoring methodies and techniques for energy disaggregation problem*, 2017. arXiv: 1703.00785 [cs.OH].

[2] C. Fischer, "Feedback on household electricity consumption: A tool for saving energy?" *Energy Efficiency*, vol. 1, pp. 79–104, Feb. 2008. DOI: 10.1007/s12053-008-9009-7.

[3] N. Batra, J. Kelly, O. Parson, H. Dutta, W. Knottenbelt, A. Rogers, A. Singh, and M. Srivastava, "Nilmtk: An open source toolkit for non-intrusive load monitoring," in *Proceedings of the 5th International Conference on Future Energy Systems*, ser. e-Energy '14, Cambridge, United Kingdom: Association for Computing Machinery, 2014, pp. 265–276, ISBN: 9781450328197. DOI: 10.1145/2602044.2602051. [Online]. Available: https://doi.org/10.1145/2602044.2602051.

[4] R. Teixeira, M. Antunes, and D. Gomes, "Using deep learning and knowledge transfer to disaggregate energy consumption," Unpublished paper, 2021.

[5] M. Zhong, N. Goddard, and C. Sutton, "Signal aggregate constraints in additive factorial hmms, with application to energy disaggregation," in *Advances in Neural Information Processing Systems*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, Eds., vol. 27, Curran Associates, Inc., 2014. [Online]. Available: https://proceedings.neurips.cc/paper/2014/file/2d1b2a5ff364606ff041650887723470-Paper.pdf.

[6] S. Darby, "The effectiveness of feedback on energy consumption: A review of the literature on metering, billing and direct displays," pp. 1–21, 2006.

[7] H. Matthews, L. Soibelman, M. Berges, and E. Goldman, "Automatically disaggregating the total electrical load in residential buildings: A profile of the required solution," Aug. 2014.

[8] J. Kelly and W. Knottenbelt, "The UK-DALE dataset, domestic appliance-level electricity demand and whole-house demand from five UK homes," *Scientific Data*, vol. 2, no. 150007, DOI: 10.1038/sdata.2015.7.

[9] J. Liang, S. K. K. Ng, G. Kendall, and J. W. M. Cheng, "Load signature study—part i: Basic concept, structure, and methodology," *IEEE Transactions on Power Delivery*, vol. 25, no. 2, pp. 551–560, 2010. DOI: 10.1109/TPWRD.2009.2033799.

[10] M. Figueiredo, A. De Almeida, and B. Ribeiro, "An experimental study on electrical signature identification of non-intrusive load monitoring (nilm) systems," Jan. 2011, pp. 31–40, ISBN: 978-3-642-20266-7. DOI: 10.1007/978-3-642-20267-4_4.

[11] A. Zoha, A. Gluhak, M. A. Imran, and S. Rajasegarar, "Non-intrusive load monitoring approaches for disaggregated energy sensing: A survey," *Sensors*, vol. 12, no. 12, pp. 16 838–16 866, 2012, ISSN: 1424-8220. DOI: 10.3390/s121216838. [Online]. Available: https://www.mdpi.com/1424-8220/12/12/16838.

[12] C. C. Yang, C. S. Soh, and V. V. Yap, "A systematic approach to on-off event detection and clustering analysis of non-intrusive appliance load monitoring," *Frontiers in Energy*, vol. 9, no. 2, pp. 231–237, Jun. 2015, ISSN: 2095-1698. DOI: 10.1007/s11708-015-0358-6. [Online]. Available: https://doi.org/10.1007/s11708-015-0358-6.

[13] Z. Zhou, Y. Xiang, H. Xu, Y. Wang, and D. Shi, "Unsupervised learning for non-intrusive load monitoring in smart grid based on spiking deep neural network," *Journal of Modern Power Systems and Clean Energy*, pp. 1–11, 2021. DOI: 10.35833/MPCE.2020.000569.

[14] H.-H. Chang, “Non-intrusive demand monitoring and load identification for energy management systems based on transient feature analyses,” *Energies*, vol. 5, no. 11, pp. 4569–4589, 2012, ISSN: 1996-1073. DOI: 10.3390/en5114569. [Online]. Available: https://www.mdpi.com/1996-1073/5/11/4569.

[15] W. Wichakool, A.-T. Avestruz, R. W. Cox, and S. B. Leeb, “Modeling and estimating current harmonics of variable electronic loads,” *IEEE Transactions on Power Electronics*, vol. 24, no. 12, pp. 2803–2811, 2009. DOI: 10.1109/TPEL.2009.2029231.

[16] S. N. Patel, T. Robertson, J. A. Kientz, M. S. Reynolds, and G. D. Abowd, “At the flick of a switch: Detecting and classifying unique electrical events on the residential power line (nominated for the best paper award),” in *UbiComp 2007: Ubiquitous Computing*, J. Krumm, G. D. Abowd, A. Seneviratne, and T. Strang, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 271–288, ISBN: 978-3-540-74853-3.

[17] J. Kelly and W. Knottenbelt, “Neural nilm,” *Proceedings of the 2nd ACM International Conference on Embedded Systems for Energy-Efficient Built Environments*, Nov. 2015. DOI: 10.1145/2821650.2821672. [Online]. Available: http://dx.doi.org/10.1145/2821650.2821672.

[18] C. Zhang, M. Zhong, Z. Wang, N. Goddard, and C. Sutton, *Sequence-to-point learning with neural networks for nonintrusive load monitoring*, 2017. arXiv: 1612.09106 [stat.AP].

[19] G. Hart, “Nonintrusive appliance load monitoring,” *Proceedings of the IEEE*, vol. 80, no. 12, pp. 1870–1891, 1992. DOI: 10.1109/5.192069.

[20] M. Sun, F. M. Nakoty, Q. Liu, X. Liu, Y. Yang, and T. Shen, “Non-intrusive load monitoring system framework and load disaggregation algorithms: A survey,” in *2019 International Conference on Advanced Mechatronic Systems (ICAMechS)*, 2019, pp. 284–288. DOI: 10.1109/ICAMechS.2019.8861646.

[21] L. Mauch, K. S. Barsim, and B. Yang, “How well can hmm model load signals,” May 2016.

[22] M. Zhuang, M. Shahidehpour, and Z. Li, “An overview of non-intrusive load monitoring: Approaches, business applications, and challenges,” in *2018 International Conference on Power System Technology (POWERCON)*, 2018, pp. 4291–4299. DOI: 10.1109/POWERCON.2018.8601534.

[23] H. Ismail Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, “Deep learning for time series classification: A review,” *Data Mining and Knowledge Discovery*, vol. 33, no. 4, pp. 917–963, Jul. 2019, ISSN: 1573-756X. DOI: 10.1007/s10618-019-00619-1. [Online]. Available: https://doi.org/10.1007/s10618-019-00619-1.

[24] J. Revuelta Herrero, Á. Lozano Murciego, A. Barriuso, D. Hernández de la Iglesia, G. Villarrubia, J. Corchado Rodríguez, and R. Carreira, “Non intrusive load monitoring (nilm): A state of the art,” Jun. 2018, pp. 125–138, ISBN: 978-3-319-61577-6. DOI: 10.1007/978-3-319-61578-3_12.

[25] L. Mauch and B. Yang, “A new approach for supervised power disaggregation by using a deep recurrent lstm network,” in *2015 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, 2015, pp. 63–67. DOI: 10.1109/GlobalSIP.2015.7418157.

[26] T.-T.-H. Le, J. Kim, and H. Kim, “Classification performance using gated recurrent unit recurrent neural network on energy disaggregation,” Jul. 2016, pp. 105–110. DOI: 10.1109/ICMLC.2016.7860885.

[27] S. Albawi, T. A. Mohammed, and S. Al-Zawi, “Understanding of a convolutional neural network,” in *2017 International Conference on Engineering and Technology (ICET)*, 2017, pp. 1–6. DOI: 10.1109/ICEngTechnol.2017.8308186.

[28] A. Iwayemi and C. Zhou, “Saraa: Semi-supervised learning for automated residential appliance annotation,” *IEEE Transactions on Smart Grid*, vol. 8, no. 2, pp. 779–786, 2017. DOI: 10.1109/TSG.2015.2498642.

[29] A. M. Fatouh, O. A. Nasr, and M. M. Eissa, “New semi-supervised and active learning combination technique for non-intrusive load monitoring,” in *2018 IEEE International Conference on Smart Energy Grid Engineering (SEGE)*, 2018, pp. 181–185. DOI: 10.1109/SEGE.2018.8499498.

[30] K. S. Barsim and B. Yang, “Toward a semi-supervised non-intrusive load monitoring system for event-based energy disaggregation,” in *2015 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, 2015, pp. 58–62. DOI: 10.1109/GlobalSIP.2015.7418156.

[31] H. Kim, M. Marwah, M. Arlitt, G. Lyon, and J. Han, “Unsupervised disaggregation of low frequency power measurements,” vol. 11, Apr. 2011, pp. 747–758. DOI: 10.1137/1.9781611972818.64.

[32] O. Parson, S. Ghosh, M. Weal, and A. Rogers, “Non-intrusive load monitoring using prior models of general appliance types,” in *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, ser. AAAI’12, Toronto, Ontario, Canada: AAAI Press, 2012, pp. 356–362.

[33] K. He, L. Stankovic, J. Liao, and V. Stankovic, “Non-intrusive load disaggregation using graph signal processing,” *IEEE Transactions on Smart Grid*, vol. 9, no. 3, pp. 1739–1747, 2018. DOI: 10.1109/TSG.2016.2598872.

[34] B. Zhang, S. Zhao, Q. Shi, and R. Zhang, “Low-rate non-intrusive appliance load monitoring based on graph signal processing,” in *2019 International Conference on Security, Pattern Analysis, and Cybernetics (SPAC)*, 2019, pp. 11–16. DOI: 10.1109/SPAC49953.2019.237866.

[35] B. Zhao, K. He, L. Stankovic, and V. Stankovic, “Improving event-based non-intrusive load monitoring using graph signal processing,” *IEEE Access*, vol. 6, pp. 53 944–53 959, 2018. DOI: 10.1109/ACCESS.2018.2871343.

[36] F. M. Wittmann, J. C. López, and M. J. Rider, “Nonintrusive load monitoring algorithm using mixed-integer linear programming,” *IEEE Transactions on Consumer Electronics*, vol. 64, no. 2, pp. 180–187, 2018. DOI: 10.1109/TCE.2018.2843292.

[37] R. Machlev, Y. Levron, and Y. Beck, “Modified cross-entropy method for classification of events in nilm systems,” *IEEE Transactions on Smart Grid*, vol. 10, no. 5, pp. 4962–4973, 2019. DOI: 10.1109/TSG.2018.2871620.

[38] M. Kaselimi, N. Doulamis, A. Voulodimos, E. Protopapadakis, and A. Doulamis, “Context aware energy disaggregation using adaptive bidirectional lstm models,” *IEEE Transactions on Smart Grid*, vol. 11, no. 4, pp. 3054–3067, 2020. DOI: 10.1109/TSG.2020.2974347.

[39] O. Krystalakos, C. Nalmpantis, and D. Vrakas, “Sliding window approach for online energy disaggregation using artificial neural networks,” in *Proceedings of the 10th Hellenic Conference on Artificial Intelligence*, ser. SETN ’18, Patras, Greece: Association for Computing Machinery, 2018, ISBN: 9781450364331. DOI: 10.1145/3200947.3201011. [Online]. Available: https://doi.org/10.1145/3200947.3201011.

[40] E. G. Santos, C. G. S. Freitas, and A. L. L. Aquino, “A deep learning approach for energy disaggregation considering embedded devices,” in *2019 IX Brazilian Symposium on Computing Systems Engineering (SBESC)*, 2019, pp. 1–8. DOI: 10.1109/SBESC49506.2019.9046095.

[41] G. Zhou, Z. Li, M. Fu, Y. Feng, X. Wang, and C. Huang, *Sequence-to-sequence load disaggregation using multi-scale residual neural network*, 2020. arXiv: 2009.12355 [cs.LG].

[42] W. He and Y. Chai, “An empirical study on energy disaggregation via deep learning,” in *Proceedings of the 2016 2nd International Conference on Artificial Intelligence and Industrial Engineering (AIIE 2016)*, Atlantis Press, Nov. 2016, pp. 338–342, ISBN: 978-94-6252-271-8. DOI: https://doi.org/10.2991/aiie-16.2016.77. [Online]. Available: https://doi.org/10.2991/aiie-16.2016.77.

[43] H. Chen, Y.-H. Wang, and C.-H. Fan, “A convolutional autoencoder-based approach with batch normalization for energy disaggregation,” *The Journal of Supercomputing*, vol. 77, no. 3, pp. 2961–2978, Mar. 2021, ISSN: 1573-0484. DOI: 10.1007/s11227-020-03375-y. [Online]. Available: https://doi.org/10.1007/s11227-020-03375-y.

[44] T. Sirojan, B. T. Phung, and E. Ambikairajah, “Deep neural network based energy disaggregation,” in *2018 IEEE International Conference on Smart Energy Grid Engineering (SEGE)*, 2018, pp. 73–77. DOI: 10.1109/SEGE.2018.8499441.

[45] Y. Quek, W. Woo, and T. Logenthiran, “Dc equipment identification using k-means clustering and knn classification techniques,” in *2016 IEEE Region 10 Conference (TENCON)*, 2016, pp. 777–780. DOI: 10.1109/TENCON.2016.7848109.

[46] G. Jacobs and P. Henneaux, “Unsupervised learning procedure for nilm applications,” in *2020 IEEE 20th Mediterranean Electrotechnical Conference (MELECON)*, 2020, pp. 559–564. DOI: 10.1109/MELECON48756.2020.9140477.

[47] H. Y. Lam, G. S. K. Fung, and W. K. Lee, “A novel method to construct taxonomy electrical appliances based on load signaturesof,” *IEEE Transactions on Consumer Electronics*, vol. 53, no. 2, pp. 653–660, 2007. DOI: 10.1109/TCE.2007.381742.

[48] M. Aiad and P. Lee, “Energy disaggregation of overlapping home appliances consumptions using a cluster splitting approach,” *Sustainable Cities and Society*, vol. 43, Sep. 2018. DOI: 10.1016/j.scs.2018.08.020.

[49] L. Mauch and B. Yang, “A novel dnn-hmm-based approach for extracting single loads from aggregate power signals,” in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2016, pp. 2384–2388. DOI: 10.1109/ICASSP.2016.7472104.

[50] H. Ismail Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, “Deep learning for time series classification: A review,” *Data Mining and Knowledge Discovery*, vol. 33, no. 4, pp. 917–963, Jul. 2019, ISSN: 1573-756X. DOI: 10.1007/s10618-019-00619-1. [Online]. Available: https://doi.org/10.1007/s10618-019-00619-1.

[51] L. Pereira, F. Quintal, R. Gonçalves, and N. Nunes, “Sustdata: A public dataset for ict4s electric energy research,” Jul. 2014. DOI: 10.2991/ict4s-14.2014.44.

[52] J. Z. Kolter and M. J. Johnson, “Redd: A public data set for energy disaggregation research,” in *IN SUSTKDD*, 2011.

[53] K. Anderson, A. Ocneanu, D. Benitez, D. Carlson, A. Rowe, and M. Berges, “Blued: A fully labeled public dataset for event-based non-intrusive load monitoring research,” *Proceedings of the 2nd KDD Workshop on Data Mining Applications in Sustainability (SustKDD)*, pp. 1–5, Jan. 2012.

[54] S. Barker, A. Mishra, D. Irwin, E. Cecchet, P. Shenoy, and J. Albrecht, *Smart*: An open data set and tools for enabling research in sustainable homes.*

[55] N. Batra, H. Dutta, and A. Singh, “Indic: Improved non-intrusive load monitoring using load division and calibration,” vol. 1, Dec. 2013. DOI: 10.1109/ICMLA.2013.21.

[56] A. Nambi, A. Lua, and V. Prasad, “Loced: Location-aware energy disaggregation framework,” Nov. 2015, pp. 45–54. DOI: 10.1145/2821650.2821659.

[57] C. Beckel, W. Kleiminger, R. Cicchetti, T. Staake, and S. Santini, “The eco data set and the performance of non-intrusive load monitoring algorithms,” in *Proceedings of the 1st ACM Conference on Embedded Systems for Energy-Efficient Buildings*, ser. BuildSys ’14, Memphis, Tennessee: Association for Computing Machinery, 2014, pp. 80–89, ISBN: 9781450331449. DOI: 10.1145/2674061.2674064. [Online]. Available: https://doi.org/10.1145/2674061.2674064.

[58] S. Makonin, B. Ellert, I. Bajic, and F. Popowich, “Electricity, water, and natural gas consumption of a residential house in canada from 2012 to 2014,” *Scientific Data*, vol. 3, p. 160 037, Jun. 2016. DOI: 10.1038/sdata.2016.37.

[59] N. Batra, O. Parson, M. Berges, A. Singh, and A. Rogers, *A comparison of non-intrusive load monitoring methods for commercial and residential buildings*, 2014. arXiv: 1408.6595 [cs.SY].

[60] O. Parson, G. Fisher, A. Hersey, N. Batra, J. Kelly, A. Singh, W. Knottenbelt, and A. Rogers, “Dataport and nilmtk: A building data set designed for non-intrusive load monitoring,” in *2015 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, 2015, pp. 210–214. DOI: 10.1109/GlobalSIP.2015.7418187.

[61] D. Murray, L. Stankovic, and V. Stankovic, “An electrical load measurements dataset of united kingdom households from a two-year longitudinal study,” *Scientific Data*, vol. 4, no. 1, p. 160 122, Jan. 2017, ISSN: 2052-4463. DOI: 10.1038/sdata.2016.122. [Online]. Available: https://doi.org/10.1038/sdata.2016.122.

[62] A. Monacchi, D. Egarter, W. Elmenreich, S. D’Alessandro, and A. M. Tonello, *Greend: An energy consumption dataset of households in italy and austria*, 2014. arXiv: 1405.3100 [cs.OH].

[63] M. Wenninger, A. Maier, and J. Schmidt, “Deddiag, a domestic electricity demand dataset of individual appliances in germany,” *Scientific Data*, vol. 8, no. 1, p. 176, Jul. 2021, ISSN: 2052-4463. DOI: 10.1038/s41597-021-00963-2. [Online]. Available: https://doi.org/10.1038/s41597-021-00963-2.

[64] G. Van Rossum and F. L. Drake, *Python 3 Reference Manual.* Scotts Valley, CA: CreateSpace, 2009, ISBN: 1441412697.

[65] Martin Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Y. Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mane, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viegas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng, *TensorFlow: Large-scale machine learning on heterogeneous systems*, Software available from tensorflow.org, 2015. [Online]. Available: `https://www.tensorflow.org/`.

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

[67] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. DOI: `10.1038/s41586-020-2649-2`. [Online]. Available: `https://doi.org/10.1038/s41586-020-2649-2`.

[68] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007. DOI: `10.1109/MCSE.2007.55`.

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

[70] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

[71] C. Diou and G. Andreou, "Eeris-nilm: An open source, unsupervised baseline for real-time feedback through nilm," Sep. 2020, pp. 1–6. DOI: `10.1109/UPEC49904.2020.9209866`.

[72] N. Batra, R. Kukunuri, A. Pandey, R. Malakar, R. Kumar, O. Krystalakos, M. Zhong, P. Meira, and O. Parson, "Towards reproducible state-of-the-art energy disaggregation," Nov. 2019, pp. 193–202, ISBN: 9781450370059. DOI: `10.1145/3360322.3360844`.

[73] F. D. Garcia, W. A. Souza, I. S. Diniz, and F. P. Marafão, "Nilm-based approach for energy efficiency assessment of household appliances," *Energy Informatics*, vol. 3, no. 1, p. 10, Oct. 2020, ISSN: 2520-8942. DOI: `10.1186/s42162-020-00131-7`. [Online]. Available: `https://doi.org/10.1186/s42162-020-00131-7`.

[74] N. Batra, R. Kukunuri, A. Pandey, R. Malakar, R. Kumar, O. Krystalakos, M. Zhong, P. Meira, and O. Parson, "Towards reproducible state-of-the-art energy disaggregation," in *Proceedings of the 6th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation*, ser. BuildSys '19, New York, NY, USA: Association for Computing Machinery, 2019, pp. 193–202, ISBN: 9781450370059. DOI: `10.1145/3360322.3360844`. [Online]. Available: `https://doi.org/10.1145/3360322.3360844`.

[75] A. Tongta and K. Chooruang, "Long short-term memory (lstm) neural networks applied to energy disaggregation," Mar. 2020, pp. 1–4. DOI: `10.1109/iEECON48109.2020.229559`.

[76] M. Stringer, *Traces*, `https://github.com/datascopeanalytics/traces`, 2021.

[77] G. R. Lee, R. Gommers, F. Waselewski, K. Wohlfahrt, and A. O'Leary, "Pywavelets: A python package for wavelet analysis," *Journal of Open Source Software*, vol. 4, no. 36, p. 1237, 2019. DOI: `10.21105/joss.01237`. [Online]. Available: `https://doi.org/10.21105/joss.01237`.

[78] P. Heracleous, P. Angkititrakul, N. Kitaoka, and K. Takeda, "Unsupervised energy disaggregation using conditional random fields," *IEEE PES Innovative Smart Grid Technologies Conference Europe*, vol. 2015, Jan. 2015. DOI: 10.1109/ISGTEurope.2014.7028933.

[79] E. Nashrullah and A. Halim, "Performance evaluation of superstate hmm with median filter for appliance energy disaggregation," in *2019 6th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI)*, 2019, pp. 374–379. DOI: 10.23919/EECSI48112.2019.8977080.

[80] M. D'Incecco, S. Squartini, and M. Zhong, "Transfer learning for non-intrusive load monitoring," *IEEE Transactions on Smart Grid*, vol. 11, no. 2, pp. 1419–1429, 2020. DOI: 10.1109/TSG.2019.2938068.

[81] M. E. Haque, M. N. Sakib Khan, and M. R. Islam Sheikh, "Smoothing control of wind farm output fluctuations by proposed low pass filter, and moving averages," in *2015 International Conference on Electrical Electronic Engineering (ICEEE)*, 2015, pp. 121–124. DOI: 10.1109/CEEE.2015.7428234.

[82] I. E. Varshavskiy, I. A. Dmitriev, A. I. Krasnova, and V. V. Polivanov, "Selection of sampling rate for digital noise filtering algorithms," in *2020 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus)*, 2020, pp. 932–935. DOI: 10.1109/EIConRus49466.2020.9039135.

[83] S. Lauren and S. D. Harlili, "Stock trend prediction using simple moving average supported by news classification," in *2014 International Conference of Advanced Informatics: Concept, Theory and Application (ICAICTA)*, 2014, pp. 135–139. DOI: 10.1109/ICAICTA.2014.7005929.

[84] P. Schirmer, I. Mporas, and M. Paraskevas, "Evaluation of regression algorithms and features on the energy disaggregation task," Nov. 2019. DOI: 10.1109/IISA.2019.8900695.

[85] P. Schirmer and I. Mporas, "Statistical and electrical features evaluation for electrical appliances energy disaggregation," *Sustainability*, vol. 11, p. 3222, Jun. 2019. DOI: 10.3390/su11113222.

[86] F. Gong, N. Han, Y. Zhou, S. Chen, D. Li, and S. Tian, "A svm optimized by particle swarm optimization approach to load disaggregation in non-intrusive load monitoring in smart homes," in *2019 IEEE 3rd Conference on Energy Internet and Energy System Integration (EI2)*, 2019, pp. 1793–1797. DOI: 10.1109/EI247390.2019.9062124.

[87] Y.-H. Lin and M.-S. Tsai, "Applications of hierarchical support vector machines for identifying load operation in nonintrusive load monitoring systems," in *2011 9th World Congress on Intelligent Control and Automation*, 2011, pp. 688–693. DOI: 10.1109/WCICA.2011.5970603.

[88] Z. Wang, W. Yan, and T. Oates, *Time series classification from scratch with deep neural networks: A strong baseline*, 2016. arXiv: 1611.06455 [cs.LG].

[89] P. Chaovalit, A. Gangopadhyay, G. Karabatis, and Z. Chen, "Discrete wavelet transform-based time series analysis and mining," *ACM Comput. Surv.*, vol. 43, p. 6, Jan. 2011. DOI: 10.1145/1883612.1883613.

[90] *A guide for using the Wavelet Transform in Machine Learning*, Accessed: 24-02-2021. [Online]. Available: https://ataspinar.com/2018/12/21/a-guide-for-using-the-wavelet-transform-in-machine-learning/.

[91] Y.-C. Su, K.-L. Lian, and H.-H. Chang, "Feature selection of non-intrusive load monitoring system using stft and wavelet transform," in *2011 IEEE 8th International Conference on e-Business Engineering*, 2011, pp. 293–298. DOI: 10.1109/ICEBE.2011.49.

[92] L. Guo, S. Wang, H. Chen, and Q. Shi, "A load identification method based on active deep learning and discrete wavelet transform," *IEEE Access*, vol. 8, pp. 113 932–113 942, 2020. DOI: 10.1109/ACCESS.2020.3003778.

[93] S. Alshareef and W. G. Morsi, "Application of wavelet-based ensemble tree classifier for non-intrusive load monitoring," in *2015 IEEE Electrical Power and Energy Conference (EPEC)*, 2015, pp. 397–401. DOI: 10.1109/EPEC.2015.7379983.

[94] I. Habou Laouali, H. Qassemi, M. Marzouq, A. Ruano, S. Bennani, and H. el fadili, "A survey on computational intelligence techniques for non intrusive load monitoring," Dec. 2020, pp. 1–6. DOI: 10.1109/ICECOCS50124.2020.9314383.

[95] M. Christ, N. Braun, J. Neuffer, and A. W. Kempa-Liehr, “Time series feature extraction on basis of scalable hypothesis tests (tsfresh – a python package),” *Neurocomputing*, vol. 307, pp. 72–77, 2018, ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2018.03.067. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231218304843.

[96] A. Dempster, F. Petitjean, and G. I. Webb, “Rocket: Exceptionally fast and accurate time series classification using random convolutional kernels,” *Data Mining and Knowledge Discovery*, vol. 34, no. 5, pp. 1454–1495, Jul. 2020, ISSN: 1573-756X. DOI: 10.1007/s10618-020-00701-z. [Online]. Available: http://dx.doi.org/10.1007/s10618-020-00701-z.