UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA E INFORMÁTICA INDUSTRIAL

LUCAS DA SILVA NOLASCO

DEEPDFML-NILM: UM MODELO BASEADO EM APRENDIZADO PROFUNDO PARA DETECÇÃO, EXTRAÇÃO DE CARACTERÍSTICAS E CLASSIFICAÇÃO DE SINAIS DE NILM

DISSERTAÇÃO

CURITIBA
2023

LUCAS DA SILVA NOLASCO

DEEPDFML-NILM: UM MODELO BASEADO EM APRENDIZADO PROFUNDO PARA DETECÇÃO, EXTRAÇÃO DE CARACTERÍSTICAS E CLASSIFICAÇÃO DE SINAIS DE NILM

DeepDFML-NILM: A Deep Learning-Based Model for Detection, Feature Extraction and Classification in NILM Signals

Dissertação apresentado(a) como requisito para obtenção do título(grau) de Mestre em Engenharia Elétrica e Informática Industrial, do Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial, da Universidade Tecnológica Federal do Paraná (UTFPR).

Orientador: Prof. Dr. André Eugênio Lazzaretti
Coorientador: Prof. Dr. Heitor Silvério Lopes

CURITIBA

2023

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Campus Curitiba

LUCAS DA SILVA NOLASCO

DEEPDFML-NILM: UM MODELO BASEADO EM APRENDIZADO PROFUNDO PARA DETECÇÃO, EXTRAÇÃO DE CARACTERÍSTICAS E CLASSIFICAÇÃO DE SINAIS DE NILM

Trabalho de pesquisa de mestrado apresentado como requisito para obtenção do título de Mestre Em Ciências da Universidade Tecnológica Federal do Paraná (UTFPR). Área de concentração: Engenharia De Computação.

Data de aprovação: 23 de Junho de 2023

ACKNOWLEDGEMENTS

First of all, I would like to express my deepest gratitude to my advisor, André Eugenio Lazzaretti, for his support and guidance over the years, and to my co-advisor, Heitor Silvério Lopes, for all the knowledge and ideas shared.

I also would like to thank my friends Everton Luiz de Aguiar and Bruna Machado Mulinari for all the discussions, ideas, and collaboration that helped to achieve the results presented in this work.

To my colleagues at Dell, my sincere appreciation for all the support that allowed me to conciliate this research with all my obligations and work duties.

Finally, I would like to thank my siblings Lohan and Livia, my parents, and all of my family, for which, without their support, I would not have a chance to be here today.

RESUMO

NOLASCO, Lucas da Silva. **DeepDFML-NILM: Um modelo baseado em aprendizado profundo para detecção, extração de características e classificação de sinais de NILM**. 2023. 100 f. Dissertação (Mestrado em Engenharia Elétrica e Informática Industrial) – Universidade Tecnológica Federal do Paraná. Curitiba, 2023.

Nas décadas futuras, o aumento contínuo do consumo de energia elétrica demandará o uso de recursos renováveis e soluções inteligentes para o gerenciamento do consumo. Neste sentido, técnicas de Monitoramento Não-Intrusivo de Cargas (NILM) detalham informações de consumo para usuários, permitindo um gerenciamento melhor da energia elétrica e evitando desperdícios. Para realizar esse monitoramento, esta técnica é geralmente composta por quatro etapas: detecção de eventos, desagregação de sinal, extração de características, e identificação da carga. No entanto, as abordagens do estado-da-arte para o monitoramento de cargas em sinais de alta frequência, geralmente baseadas em redes neurais, não apresentam uma arquitetura completa de solução. Para resolver essa questão, este trabalho apresenta um método integrado para a detecção, extração de características e classificação em sinais de alta-frequência, testado no conjunto de dados público LIT-*Dataset*. Para a detecção, os resultados encontrados estiveram acima de 90% para a maioria dos casos, enquanto os métodos do estado-da-arte apresentam desempenho inferior a 70% para cenários com oito cargas conectadas à rede elétrica. Para a classificação, o desempenho final nas métricas avaliadas foi equiparável a outros trabalhos recentes, apresentando F-*score* e acurácia superiores a 97%. A arquitetura proposta ainda conta com uma estratégia *multi-label* para evitar a necessidade de um estágio de desagregação, indicando todas as cargas conectadas em um dado momento, aumentando o reconhecimento de múltiplas cargas. Esse trabalho ainda avalia o desempenho do método proposto em cenários com inserção de ruído no sinal de entrada. Finalmente, também é avaliado o desempenho da arquitetura proposta em um sistema embarcado, um assunto ainda sub-explorado na literatura recente, demonstrando a exequibilidade para a análise de sinais em tempo real e aplicações práticas envolvendo monitoramento de cargas.

Palavras-chave: Monitoramento de Cargas. NILM. Redes Neurais. Sistemas Embarcados.

ABSTRACT

NOLASCO, Lucas da Silva. **DeepDFML-NILM: A Deep Learning-Based Model for Detection, Feature Extraction and Classification in NILM Signals**. 2023. 100 p. Dissertation (Master's Degree in Engenharia Elétrica e Informática Industrial) – Universidade Tecnológica Federal do Paraná. Curitiba, 2023.

In the coming decades, the growth in energy consumption will require renewable resources and intelligent solutions to manage consumption. In this sense, Non-Intrusive Load Monitoring (NILM) techniques emerge as a way to provide detailed consumption information to users, enabling better power management and avoiding energy losses. These load monitoring techniques typically require four steps: event detection, signal disaggregation, feature extraction, and load identification. However, for high-frequency NILM methods, state-of-the-art approaches, mainly based on deep learning solutions, do not provide a complete NILM architecture that includes all the required steps. To fill this gap, this work presents an integrated method for the detection, feature extraction, and classification of high-frequency NILM signals for the publicly available LIT-Dataset. For detection, the results were above 90% in most cases, while the state-of-the-art methods were below 70% for eight simultaneous loads. For classification, the performance of the proposed model on the evaluated metrics was comparable to other recent works, reaching over 97% for F-score and accuracy. The proposed architecture also includes a multi-label classification strategy to avoid the disaggregation stage, indicating the loads connected at a given time and allowing the identification of multiple loads simultaneously. This work also evaluates the robustness of the proposed method to noise insertion. Finally, this work presents results in an embedded system, a topic also underexplored in the recent literature, demonstrating the feasibility of the proposal for real-time signal analysis and practical applications involving NILM.

Keywords: Load Monitoring. NILM. Neural Networks. Embedded Systems.

LIST OF ALGORITHMS

Algorithm 1 – Algorithm for label creation 54
Algorithm 2 – Algorithm to create the events from the predictions 59

LIST OF FIGURES

Figure 1 – Overall structure of the proposed solution. 15
Figure 2 – General sequence of steps for applying the NILM technique. 19
Figure 3 – Example of the proposed variance curve in High Accuracy NILM Detector (HAND) calculated for a fan signal. 32
Figure 4 – Example of an octuple waveform containing the loads: (i) Cell phone charger (Motorola), (ii) Fume Extractor, (iii) Soldering Station, (iv) LED Panel, (v) Vaio Notebook, (vi) Incandescent Lamp, (vii) Eleganza Hair Dryer, and (viii) Parlux Hair Dryer. In blue is the aggregate current of all loads. In orange are the load-switching events, where up indicates a turn-on and down indicates a turn-off. 41
Figure 5 – Overview of the proposed method. 43
Figure 6 – Highlighted are the convolutional layers used to extract features shared by the three subproblems. The features extracted by this set of layers pass through each subnetwork, which will compute features for its specific problem. . . . 44
Figure 7 – Representation of the grid division used by the model to interpret the input signal. 45
Figure 8 – Comparison between predictions generated by the proposed model and by YOLO. On the left is shown a detection performed by the architecture presented in this work, and on the right is shown a detection performed by YOLOv3. 48
Figure 9 – Overview of the training and evaluation processes of the proposed model. . 49
Figure 10 – Example of applying the HAND algorithm in choosing event-free excerpts in the signal. 51
Figure 11 – Example of a signal excerpt created using the proposed method. 52
Figure 12 – Distribution of the number of instances for each load present in the dataset. . 55
Figure 13 – Losses for training and validation sets during model training. 57
Figure 14 – Comparison of noise levels applied during the tests. 60
Figure 15 – Platform used for performance testing in an embedded system. 61
Figure 16 – Overview of the proposed model using Scattering Transforms (STs). 62
Figure 17 – Grid division applied for the windowed average. 62
Figure 18 – An example scenario where the model could not correctly detect an event. This clipping represents an acquisition of the LIT-SYN-8 at the time of the Soldering Station (40 W) shutdown, while other loads, such as the Hair Dryer Eleganza (365 W) and Hair Dryer Parlux (660 W), were connected. The indication of the connected loads uses the appliance ID defined in Table 2. . 70
Figure 19 – Comparison of the effect of the load triggering angle on the waveform for Microwave Standby switching events. 72
Figure 20 – ROC curves for all appliances in LIT-SYN. 73
Figure 21 – Confusion matrices for some of the devices in LIT-SYN. 74
Figure 22 – False negatives observed for the Lenovo Laptop. 75
Figure 23 – Examples of waveforms containing the LED Panel. 75
Figure 24 – Comparison of the effects of using weights for training the load identification output. 76
Figure 25 – Visual examples with acquisitions of the LIT-SYN-8 subset. 77

Figure 26 – Graphical comparison of the detection performance of the methods in Table 7. 79
Figure 27 – Evolution of detection performance as the signal quality degrades. The farther to the right on the x-axis, the higher the noise level relative to the signal. Regarding notation, “-” indicates the signal without adding noise. 82
Figure 28 – Comparison of switching a higher power load in scenarios with and without noise addition (10 dB SNR). These signal excerpts include the following devices: Drill (15), Eleganza Hair Dryer (20), and Parlux Hair Dryer (25). . 84
Figure 29 – Comparison of switching a lower power load in scenarios without noise addition (SNR of 10 dB) and with noise addition. These signal sections include the devices: Fume Extractor (4), Soldering Station (7), Motorola Cell Phone Charger (8), Vaio Laptop (12), Incandescent Lamp (13), Drill (15), and Oil Heater (16). 85
Figure 30 – Evolution of classification performance measures as signal degradation increases. The farther to the right on the x-axis, the higher the noise level relative to the signal. 86

LIST OF TABLES

Table 1 – Related work and contributions of this research. Concerning notation, “Y” indicates that the paper contains the feature in question, while “–” indicates the absence of such a feature. 40
Table 2 – Classes present in the synthetic subset of the LIT Dataset (LIT-SYN). 42
Table 3 – Model training parameters. 56
Table 4 – Detection results. 68
Table 5 – Classification results. 71
Table 6 – Classification results without using weights. 75
Table 7 – Comparison of the detection results. 78
Table 8 – Comparison of classification results between works that use the LIT-Dataset. 80
Table 9 – Comparison of classification results considering only multi-label models. . . 81
Table 10 – Comparison of proposed architecture’s event detection performance in noisy scenarios. Regarding the notation, “-” stands for the original signal, i.e., without added noise. 83
Table 11 – Comparison of the performance of the proposed architecture for load identification in noisy scenarios. 87
Table 12 – Comparison of results on an embedded system. Regarding the notation, “Y” means that the paper implements this aspect, while “-” means that the paper does not consider this aspect. 87
Table 13 – Resource consumption on the embedded system. 88
Table 14 – Detection performance comparison between Scattering Transform (ST)-based and Convolutional Neural Network (CNN)-based architectures. 89
Table 15 – Classification performance comparison between ST-based and CNN-based architectures. 90

LIST OF ACRONYMS

INITIALISM

ADL	Active Deep Learning
AUC	Area Under the Curve
AWGN	Additive White Gaussian Noise
BCE	Binary Cross-Entropy
CCE	Categorical Cross-Entropy
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CWT	Continuous Wavelet Transform
DT	Decision Tree
DWT	Discrete Wavelet Transform
FPGA	Field Programmable Gate Array
GPU	Graphical Processing Unit
IEEE	Institute of Electrical and Electronics Engineers
LSTM	Long Short-Term Memory
MSE	Mean Squared Error
PNN	Probabilistic Neural Network
RNN	Recurrent Neural Networks
SGD	Stochastic Gradient Descent
SNR	Signal-to-Noise Ratio
SSE	Sum Squared Error
ST	Scattering Transform
SVM	Support Vector Machine
TSCNN	Two-Stream Convolutional Neural Network
WRG	Weighted Recurrence Graph

ACRONYMS

ARM	Advanced RISC Machine
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
GAN	Generative Adversarial Network
HAND	High Accuracy NILM Detector
LIT	Laboratory of Innovation and Technology in Embedded Systems and Energy
LIT-SYN	Synthetic subset of the LIT-Dataset
MAE	Mean Absolute Error
NILM	Non-Intrusive Load Monitoring
PLAID	Plug Load Appliance Identification Dataset
RAM	Random Access Memory

REDD	Reference Energy Disaggregation Dataset
REFIT	Personalised Retrofit Decision Support Tools for UK Homes using Smart Home Technology
ReLU	Rectified Linear Unit
ROC	Receiver Operating Characteristic
UK-DALE	United Kingdom Domestic Appliance-Level Electricity
V-I	Voltage and Current
WHITED	Worldwide Household and Industry Transcient Energy Dataset
YOLO	You Only Look Once

CONTENTS

1 INTRODUCTION . . . 14
1.1 MOTIVATION . . . 16
1.2 OBJECTIVES . . . 16
1.2.1 General Objectives . . . 16
1.2.2 Specific Objectives . . . 16
1.3 WORK STRUCTURE . . . 17
1.4 PUBLICATIONS . . . 17

2 THEORETICAL BACKGROUND . . . 18
2.1 NON-INTRUSIVE LOAD MONITORING – NILM . . . 19
2.2 FEEDFORWARD NEURAL NETWORKS . . . 20
2.3 CONVOLUTIONAL NEURAL NETWORKS – CNN . . . 21
2.4 ACTIVATION FUNCTIONS . . . 22
2.4.1 ReLU . . . 22
2.4.2 *Leaky* ReLU . . . 22
2.4.3 *Sigmoid* . . . 23
2.4.4 *Softmax* . . . 23
2.5 LOSS FUNCTIONS . . . 23
2.5.1 Sum Squared Error . . . 24
2.5.2 Cross-Entropy . . . 24
2.6 TRAINING . . . 25
2.6.1 Optimizers . . . 26
2.6.1.1 Stochastic Gradient Descent – SGD . . . 26
2.6.1.2 Adam . . . 27
2.7 REGULARIZATION . . . 28
2.7.1 Dropout . . . 28
2.7.2 Early-Stopping . . . 29
2.8 VALIDATION . . . 29
2.8.1 K-Fold . . . 30
2.9 MULTI-TASK LEARNING . . . 30
2.10 HIGH ACCURACY NILM DETECTOR – HAND . . . 31
2.11 ADDITIVE WHITE GAUSSIAN NOISE – AWGN . . . 33
2.12 SCATTERING TRANSFORM . . . 33
2.13 ROC CURVES AND CONFUSION MATRICES . . . 34

3 LITERATURE REVIEW . . . 36
3.1 RELATED WORKS . . . 36
3.2 WORK CONTRIBUTIONS . . . 39

4 METHODOLOGY . . . 41
4.1 LIT-DATASET . . . 41
4.2 PROPOSED METHOD (DEEPDFML-NILM) . . . 43
4.2.1 Architecture . . . 43
4.2.1.1 Model input . . . 44
4.2.1.2 Feature extraction . . . 45

4.2.1.3 Model output 46
4.2.2 Similarities and differences to YOLO 47
4.3 TRAINING 48
4.3.1 Preprocessing 49
4.3.1.1 Signal excerpts with events 49
4.3.1.2 Signal excerpts without events 49
4.3.1.3 Creating the labels 50
4.3.1.4 Creating the training and test sets 53
4.3.2 Training and model selection 54
4.3.3 Evaluating and processing the model output 58
4.4 NOISE 59
4.5 EMBEDDED SYSTEM 61
4.6 SCATTERING TRANSFORM – ST 61

5 RESULTS 65
5.1 PERFORMANCE MEASURES 65
5.1.1 Detection Performance Measures 65
5.1.1.1 Percentage of Correct detections – PC 65
5.1.1.2 Average Distance – D 66
5.1.2 Classification Performance Measures 67
5.1.2.1 Macro Averaged Accuracy – ACC 67
5.1.2.2 F_1-Score 68
5.2 DETECTION 68
5.3 CLASSIFICATION 69
5.4 VISUAL EXAMPLE 76
5.5 COMPARISON WITH RELATED WORK 78
5.5.1 Detection 78
5.5.2 Classification 79
5.6 NOISE 81
5.6.1 Detection 82
5.6.2 Classification 85
5.7 EMBEDDED SYSTEM 87
5.8 SCATTERING TRANSFORM – ST 89
5.8.1 Detection Performance 89
5.8.2 Classification Performance 90

6 CONCLUSIONS AND FUTURE WORKS 91
6.1 FUTURE WORK 93

REFERENCES 94

1 INTRODUCTION

Non-Intrusive Load Monitoring (NILM) techniques consist of strategies to identify the devices connected to an electrical network by taking measurements at a single point on the network. These techniques estimate the consumption of each appliance in commercial or residential environments, allowing more effective identification of unnecessary electricity use by consumers, reducing energy waste and improving energy efficiency (EHRHARDT-MARTINEZ *et al.*, 2010). Thus, in addition to providing savings for the consumer by helping to improve energy efficiency, these techniques can also have a positive impact on the environment by helping to reduce the carbon emissions that result from some forms of electricity generation.

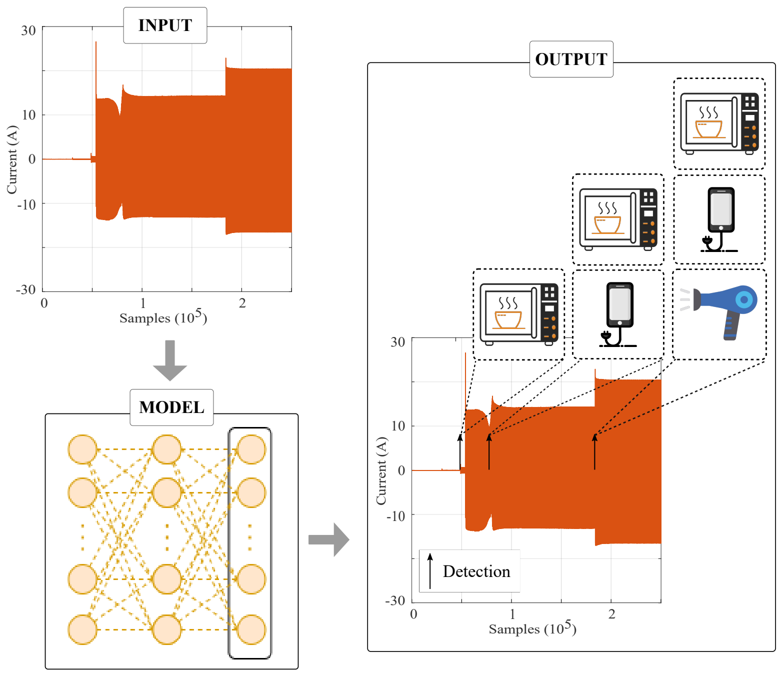
As a non-intrusive technique, NILM methods need to identify the devices in an aggregated waveform, making it a challenging problem. For this reason, NILM solutions typically have several steps to achieve this goal, namely: (i) event detection, (ii) signal disaggregation, (iii) feature extraction, and (iv) load classification. In the first step, NILM methods are supposed to detect a switching event in the voltage and current measurements (NAIT MEZIANE *et al.*, 2017). Then, in the disaggregation step, the load signal responsible for the event is separated from the background signal composed of the previously connected loads (KELLY; KNOTTENBELT, 2015). Then, with the signal of the load responsible for the event separated from the rest of the signal, it proceeds to feature extraction. In this third phase, the features describing the loads can be extracted manually (WANG *et al.*, 2018) or not (AGUIAR *et al.*, 2021; FAUSTINE; PEREIRA, 2020). Finally, with the extracted features, the device is identified using classification methods based on machine learning (LAZZARETTI *et al.*, 2020) or deep learning (FAUSTINE; PEREIRA, 2020) techniques.

However, the existing solutions for NILM have some complications, the main one being the significant dependence on the event detection stage. In some studies, such as the one proposed in Faustine and Pereira (2020), the system performance is analyzed assuming that the moments of occurrence of the load switching events are known in advance, i.e. without considering the impact of the detection step on the performance or analyzing the effects that detections with a significant error in the classification could imply. In Lazzaretti *et al.* (2020), the detection step is implemented. However, it is shown that there is a significant loss of performance with the increase in the number of loads connected to the electrical grid, dropping to 60% accuracy for detection in the scenario with up to 8 loads.

Another problem some techniques present is the need for a load disaggregation operation to identify the device responsible for the switching event in scenarios where the electrical grid has multiple connected loads. This step has a direct impact on the classification process. In some cases, it is simply implemented as a subtraction between the signal before and after the event, as is the case in the work proposed by Mukaroh *et al.* (2020). The problem with this approach is that the resulting signal can be deformed in the case of nonlinear loads, which can interfere with the classification step.

Based on all the above, this project proposes the DeepDFML-NILM[1], a model capable of simultaneously detecting and identifying the activation of loads in current measurements of the power grid using 1D convolutional neural networks, taking inspiration from ideas that perform similar functions for images, such as You Only Look Once (YOLO) (REDMON *et al.*, 2016). In addition, this model has a multi-label architecture (SOROWER, 2010), which allows the simultaneous identification of multiple devices without the need to perform a disaggregation step, simplifying and speeding up signal analysis. A general structure of this proposed solution is shown in Figure 1. This proposal uses deep learning techniques to create a model capable of detecting switching events and identifying all connected devices from the aggregated current waveform.

Figure 1 – Overall structure of the proposed solution.



Source: Own work.

[1] **Deep** neural network model for **D**etection, **F**eature extraction, and **M**ulti-**L**abel **NILM** classification.

1.1 MOTIVATION

With the annual increase in demand for electricity, as indicated by the International Energy Agency (IEA, 2019), comes the need for improved energy efficiency in both commercial and residential environments. A more direct way to improve energy efficiency is to develop devices with this feature, a more costly solution. As a result, research in areas such as NILM has intensified.

With devices implementing NILM algorithms, the user would have more information about their consumption, thus reducing wasted electricity and consequently improving the energy efficiency of their home or business (ARMEL *et al.*, 2013). In addition, since NILM has the characteristic of being a non-intrusive technique, its installation would be easier, requiring only a single measurement point, which would facilitate the expansion of its use.

In this context, this work aims to contribute to the scenario of seeking more robust alternatives for NILM solutions. To this end, in addition to the new proposed architecture, noise robustness and performance tests are performed on an embedded system to analyze the feasibility of the practical implementation of this solution.

1.2 OBJECTIVES

1.2.1 General Objectives

To develop a model capable of performing feature extraction, detection, and classification steps in NILM signals in a unified way.

1.2.2 Specific Objectives

- To study the state-of-the-art of solutions for NILM techniques;
- To study digital signal processing and pattern recognition techniques for building the final solution;
- To study correlated techniques for object detection in images and videos;
- Study deep learning and multi-task learning techniques;
- To develop a model capable of detecting events in NILM signals;

- To develop a model capable of identifying connected loads in NILM signals;
- Evaluate the model created in public databases;
- Evaluate the robustness to noise of the system created;
- Evaluate the performance in an embedded system;
- Compare the performance of the proposed method with the state-of-the-art.

1.3 WORK STRUCTURE

This dissertation consists of six chapters. The second chapter presents the working steps of NILM techniques and the theoretical concepts that will be used in the solution. The third chapter presents the state of the art and the contributions of this work. The fourth and fifth chapters present the proposed methodology and the results obtained. Finally, the sixth and last chapter presents the conclusions of this research.

1.4 PUBLICATIONS

During the development of this dissertation, several topics were addressed. The contributions of these research efforts have been published as scientific papers in international journals. Mulinari *et al.* (2022) present a strategy for load classification using Voltage and Current (V-I) trajectories and Fourier 2-D as an alternative to hand-crafted features, which may be suboptimal, and neural network models, which require a large amount of data to achieve good performance.

Nolasco *et al.* (2022) present DeepDFML-NILM, a novel neural network architecture to simultaneously perform event detection and load identification in NILM signals. This paper, published in the IEEE Sensors Journal, was listed as one of the journal's most viewed papers in its year of publication, resulting in an invitation for the authors to present their work at the IEEE Sensors Conference 2022, in Dallas, Texas. In addition, the Institute of Electrical and Electronics Engineers (IEEE) awarded the authors a financial grant to guarantee their attendance at the event.

Finally, Aguiar *et al.* (2023) discuss the use of scattering transforms in combination with fully connected neural network layers to perform event detection and multi-label load classification in NILM signals. This paper is currently under review for publication in the IEEE Sensors Journal (Impact Factor: 4.325).

2 THEORETICAL BACKGROUND

This chapter presents the main theoretical concepts involved in this work. For this purpose, the following sections are organized as follows:

- Section 2.1: Presents more details about NILM techniques, which is the central problem of this work. This section provides a clearer view of the problems that the method proposed in this research aims to solve;
- Sections 2.2 and 2.3: Introduces the concept behind the layers composing the neural network used to create the proposed model. Section 2.2 deals with the feedforward layers, and Section 2.3 presents the convolutional layers;
- Section 2.4: Presents the activation functions responsible for adding nonlinearity to the proposed model;
- Section 2.5: Presents the loss functions used to measure the cost of the model's output predictions, indicating how close they are to what is expected;
- Section 2.6: This section outlines methods for training a neural net, where the goal is to decrease the cost of outputs;
- Section 2.7: Introduces the regularization methods used to increase the generalization ability of the trained model by preventing it from overfitting on the training data;
- Section 2.8: Explains the need for validation sets and presents a strategy for cross-validation;
- Section 2.9: Introduces the Multi-task Learning technique used to create the architecture in the approach chosen to solve the problem;
- Section 2.10: Presents the method of event detection in NILM signals used to choose event-less excerpts that will be used later in the model training;
- Section 2.11: Presents the Additive White Gaussian Noise (AWGN) employed in the noise robustness tests;
- Section 2.12: Introduces the Scattering Transform (ST), an alternative to CNNs based on the wavelet transform;

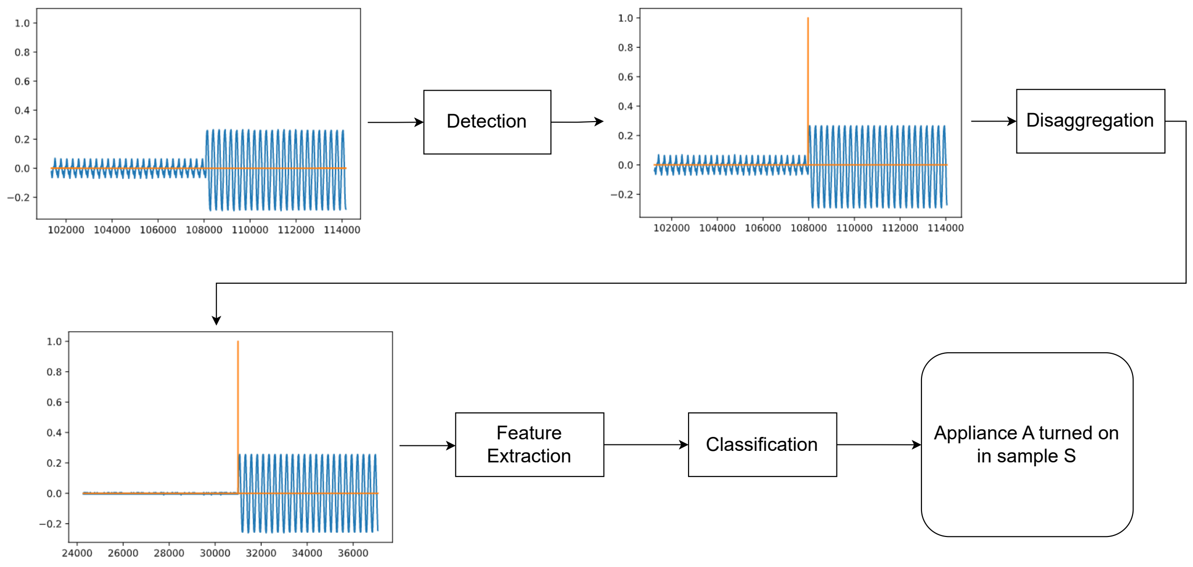
- Section 2.13: Presents the Receiver Operating Characteristic (ROC) Curves and Confusion Matrices, which are helpful tools to analyze a model performance in classification tasks.

The content of the following sections of this chapter is heavily inspired by the work of Goodfellow *et al.* (2016).

2.1 NON-INTRUSIVE LOAD MONITORING – NILM

The idea of monitoring loads is not new. It was first described by Hart (1992). This paper proposed to identify the loads connected to the electrical system using only the aggregated voltage and current signals obtained from a single metering point. This allows voltage and current readings to be taken directly from the electrical panel, eliminating the need to monitor each device individually, which characterizes the non-intrusive nature of the technique. In practice, the NILM technique generally follows a structure composed of four steps, namely: detection, disaggregation, feature extraction and classification. This structure is shown in Figure 2.

Figure 2 – General sequence of steps for applying the NILM technique.



Source: Own work.

The first of these steps is to identify when a new load-switching event has occurred on the grid. Next, the disaggregation step is performed (KELLY; KNOTTENBELT, 2015), where the system attempts to separate the signal of the load that caused the new event from the aggregate measured signal, typically by subtracting the (post and pre-event) signals. From this disaggregated signal, the system extracts features that are used to classify the connected or disconnected device using machine learning (LAZZARETTI *et al.*, 2020) or deep learning (FAUSTINE; PEREIRA, 2020) methods.

2.2 FEEDFORWARD NEURAL NETWORKS

The basis for many pattern recognition applications, neural networks have their origins in the 20th century. At that time, the name “neural networks” was coined based on the inspiration for their creation, the representation of information processing in biological systems, i.e. the functioning of neurons in animal brains. An example of this attempt is the work proposed by McCulloch and Pitts (1943), where the authors propose a way to represent neural activity using propositional logic.

However, modern neural networks are closer to function approximation techniques designed to achieve statistical generalization rather than accurate models of how the brain works. In this scenario, a feedforward neural network model’s goal is to approximate somehow a function f^*, where $\mathbf{y} = f^*(\mathbf{x})$ maps an input $\mathbf{x}$ to an output $\mathbf{y}$. To obtain this approximation, the network defines a mapping from the function $f(\mathbf{x}; \boldsymbol{\theta})$, where the parameter $\boldsymbol{\theta}$ is the one that results in the best approximation of the function f^*. To obtain $\boldsymbol{\theta}$, the model goes through a training process, also popularly known as “learning”, which will be covered in more detail in the following sections (see Section 2.6).

Regarding the idea of networks behind the name neural networks, this is explained by the fact that the function $f(\mathbf{x})$ mentioned above is decomposed into a chain of functions $f^{(1)}$, $f^{(2)}$, $f^{(3)}$, ..., $f^{(n)}$, where each of these functions represents a layer in the model. In this scenario, $f^{(1)}$ represents the first layer of the network, $f^{(2)}$ the second, and so on. Still, regarding terminology, the term feedforward comes from how the information is propagated through the network. The input $\mathbf{x}$ passes through the first layer, the output of that layer passes through the next layer, and so on until it passes through all the layers that define $f(\mathbf{x})$ until the output is obtained.

Conversely, intermediate functions $f^{(i)}$ can be defined as a nonlinear function of the linear combination between their inputs and weights calculated during training. The following expression represents the output layer’s result:

$$\mathbf{h}^{(i)} = g^{(i)}\left(\mathbf{W}^{(i)T}\mathbf{h}^{(i-1)} + \mathbf{b}^{(i)}\right), \tag{1}$$

where $\mathbf{W}^{(i)}$ denotes the weight matrix for layer $f^{(i)}$, $\mathbf{b}^{(i)}$ is a vector of constants calculated during the training phase, also known as bias, $g^{(i)}$ represents the activation function for this layer, and $\mathbf{h}^{(i)}$ is the output for layer i. This activation function is responsible for adding the nonlinearity of the layer, which is essential for the model to represent nonlinear functions f^*. Some examples of possible activations will be discussed in more detail in the following sections.

2.3 CONVOLUTIONAL NEURAL NETWORKS – CNN

CNNs are a particular type of neural networks. Their use is strongly linked to data such as images and time series, gaining notoriety because of their high performance in practical applications. The first mention of CNNs can be found in Le Cun *et al.* (1989), where this neural network was used to classify images containing handwritten numbers.

Nevertheless, this approach only became a success with the evolution of hardware, with the emergence of Graphical Processing Units (GPUs), which allowed the training of models containing convolutional layers with a much larger volume of data more efficiently. In 2012, the architecture AlexNet, proposed by Krizhevsky *et al.* (2012), brought renewed attention to this type of neural network by achieving a significant reduction of approximately 10% in the error rate in the image classification task compared to the state-of-the-art at the time of its publication.

One of the main differences between this type of neural network and the feedforward type is the operation performed in each layer. In this type of network, as the name implies, a convolution operation is performed between the input and the kernel – the name for the matrix that stores the weights of this type of layer. Thus, the process performed in a convolutional layer could be represented as follows:

$$\mathbf{O}(i,j) = (\mathbf{K} * \mathbf{I})(i,j) = \sum_m \sum_n \mathbf{I}(i+m, j+n) \cdot \mathbf{K}(m,n), \tag{2}$$

where $\mathbf{I}$ denotes the matrix containing the input data, $\mathbf{K}$ denotes the kernel, $\mathbf{O}$ represents the output, and i and j represent the coordinates of the input matrix at which the convolution is being done. It is worth noting that the convolution operation requires the kernel to be rotated, which is not usually done. So, although the idea is similar, the correct name for the action presented in Equation 2 would be Cross-Correlation.

After the convolution, the output of the layer is further composed of an activation function, following the same principles as feedforward networks. The other difference for CNNs is the addition of pooling layers, which act as a sub-sampling layer, usually placed after the convolutional layers. The pooling layers replace the output of a layer at each of its positions with statistical functions of the surrounding outputs. That allows for a decrease in the output size of a layer and consequently reduces the number of parameters needed by the model. One of the most popular pooling layers is Max-Polling, which reduces the number of coefficients in a layer by selecting the maximum value within a rectangular region.

2.4 ACTIVATION FUNCTIONS

The activation functions are responsible for adding the nonlinear transformation to the output of each network layer. For this purpose, there are many options in the literature for choosing this type of function, some of the most famous being Rectified Linear Unit (ReLU), Sigmoid, and Softmax. The following subsections will detail some of these options.

2.4.1 ReLU

The ReLU is one of the most famous activation functions. Its main characteristic is that it behaves linearly, as its name suggests. However, to present some non-linearity, the rectified part of the ReLU comes in. Thus, the output for every negative value is zero, creating non-linearity at the function's zero point. This behavior can be summarized in the following equation:

$$g(z) = \text{max}\{0, z\}. \tag{3}$$

The linearity and constant derivative make it easy to optimize the model by gradient descent strategies (see Section 2.6), maintaining a high gradient at any point that the function is active. The downside of this function, however, is that the gradient is 0 for the negative part, making the model unable to learn when the activation is zero.

2.4.2 *Leaky* ReLU

The Leaky ReLU emerged as a way to solve the problem of the ReLU gradient when the function is inactive. Created by Maas *et al.* (2013), this activation function proposes a simple solution in which ReLU could "leak" part of the input to the output when the input values are negative, hence its name. With this change, the new activation function would take the following form:

$$g(z, \alpha) = \text{max}\{0, z\} + \alpha \cdot \text{min}\{0, z\}, \tag{4}$$

where α is a user-defined hyperparameter, usually assuming low values like 0.01. This change will give the function a derivative equal to α for negative values, preventing it from stopping learning for these cases and mitigating the problem cited earlier for ReLU while retaining all the advantages for the case where z is positive.

2.4.3 *Sigmoid*

Another famous option among the activation functions is Sigmoid. Its main feature is to saturate to 1 for high input and 0 for low values. To achieve this, its activation function is defined as follows:

$$\sigma(z) = \frac{1}{1 + \exp(-z)}. \tag{5}$$

Its characteristic of keeping the output within the interval $[0, 1]$ makes Sigmoid efficient in modeling binary problems that behave like a Bernoulli distribution. However, this saturation feature causes this activation function to have a 0 gradient over much of its domain, which can be detrimental to training, as will be detailed later. Because of this, ReLU and Leaky ReLU have gained ground, with Sigmoid being more common for output layers.

2.4.4 *Softmax*

Conversely, softmax, like Sigmoid, is generally used as an activation function for output layers. However, unlike Sigmoid, which is usually applied to binary problems, Softmax excels in scenarios of a probability distribution among n classes. For this, the activation function is defined as follows:

$$\mathit{softmax}(z)_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}. \tag{6}$$

Among other properties, exponentiation keeps the values positive. Normalization keeps the sum of all output probabilities within the interval $[0,1]$ and with a unit total, resembling a probability distribution among the n classes, as desired.

2.5 LOSS FUNCTIONS

Before training models in a supervised system, it is necessary to have a way to measure how close the model is to solve the problem it sets out to solve. The correct answers for a given input are known in advance for supervised learning. Thus, the role of loss functions, also known as cost functions, is to indicate the model's ability to solve the problem it is trying to solve by comparing the expected output to the predictions made by the model.

This will allow, during the training stage, the optimization algorithm to have an insight into the improvement, or not, of the performance of the system being trained. Thus, the total cost of the training set can be defined as follows:

$$J(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m} L(f(\mathbf{x}^{(i)};\boldsymbol{\theta}), \mathbf{y}^{(i)}), \tag{7}$$

where m is the number of training samples, L is the function that indicates the loss for each example, $\mathbf{y}^{(i)}$ is the expected output, and $f(\mathbf{x}^{(i)};\boldsymbol{\theta})$ is the prediction for input $\mathbf{x}^{(i)}$. Since this cost is calculated only on the training distribution and not for the entire data distribution for the problem, it is also known as empirical risk.

To better describe the problem for which the model is to be trained, there are different loss functions (L) depending on the model architecture, the activation function of the output layers, and the problem at hand. The loss functions used in this work are Categorical Cross-Entropy (CCE), Binary Cross-Entropy (BCE), Mean Squared Error (MSE), and Sum Squared Error (SSE). The subsections that follow detail these functions.

2.5.1 Sum Squared Error

Sum Squared Error is one of the simplest functions. It calculates the model error as the squared distance between the prediction made by the model and what was expected. Because of this property, this loss function is often used in regression problems, where the goal is to fit the predictions of a given model to a curve. This loss function can be defined by the following equation:

$$SSE = \sum_{i=1}^{N} (\hat{y}_i - y_i)^2, \tag{8}$$

where $\hat{y}_i$ is the model's prediction, y_i is the expected output, and N is the number of training examples.

2.5.2 Cross-Entropy

Cross-Entropy is a cost function aimed at classification models, appearing even in other machine learning techniques such as Logistic Regression (BISHOP, 2006). In its definition, it has the following form:

$$CE = -\sum_{n=1}^{N}\sum_{k=1}^{K} y_{kn} \cdot \log \hat{y}_{kn}, \tag{9}$$

where K represents the number of classes in the problem, N represents the number of samples, $\hat{y}_{kn}$ denotes the prediction for class k of the n sample, y_{kn} represents the expected prediction for class k of the n sample, and $y_{kn} \in \{0, 1\}$.

This cost type is intended for outputs with K possible classes, where only one is chosen at any given time. Because of this, this cost function is generally used with output layers with a softmax activation function. In this form, Cross-Entropy is also known as Categorical Cross-Entropy. Here, if the expected for a class is 0, there is no contribution to the total cost. If y_{kn} equals 1, the closer the prediction is to 0, the higher the cost. On the other hand, if the forecast is also 1, the result will be correct – therefore, there will be no contribution to the cost.

An important issue is that if the expected output y is 0, this cost function will not work well, as there would be no contribution to the total cost. To get around this problem, there is a modified version of this function, also known as Binary Cross-Entropy, for binary output problems. Since there will be only two classes for the same output, the idea already observed in Cross-Entropy is followed, adding a new term for the case where the expected prediction is 0. This modification can be described as follows:

$$BCE = -\sum_{n=1}^{N} [y_n \log \hat{y}_n + (1 - y_n) \log(1 - \hat{y}_n)]. \tag{10}$$

2.6 TRAINING

As presented in Section 2.5, loss functions provide a way to measure how well the model can generate the expected output. Thus, training aims to improve the system's performance on this metric, corresponding to minimizing this cost function. However, due to the nonlinearity in neural nets, the most relevant loss functions are characterized by being non-convex. This prevents the loss functions from being minimized directly by a straightforward solution of a system of equations, requiring iterative methods that reduce the cost based on its gradient.

These iterative methods are based on the mathematical property that says that the gradients of a function will point in the direction of the function's growth. Thus, the function can be minimized iteratively by taking small steps in the direction opposite to the direction indicated by the gradient. This procedure can be formally defined by the following equation:

$$\mathbf{x}' = \mathbf{x} - \epsilon \cdot \nabla_{\mathbf{x}} f(\mathbf{x}), \tag{11}$$

where $\mathbf{x}$ represents the current point of the function, $\mathbf{x}'$ represents the next point where the function should be evaluated, $\nabla_{\mathbf{x}} f(\mathbf{x})$ represents the gradient of the function at point $\mathbf{x}$, and ϵ, also known as the learning rate, is a positive scalar that will determine the size of the step to be taken in the attempt to find the minimum of the function.

2.6.1 Optimizers

Over the years, as a way to implement this gradient descent strategy, several algorithms have been developed with modifications that aim to improve the final performance of the optimization in some way. The following subsections present some of the most famous examples in more detail.

2.6.1.1 Stochastic Gradient Descent – SGD

Stochastic Gradient Descent (SGD) is one of the most widespread algorithms for optimization in deep learning. It follows the same gradient descent strategy already presented so far. However, it also proposes a modification to solve a problem encountered by gradient descent: performance. Deep learning models need a large amount of data to improve their generalization power. Thus, there will be many examples when calculating the cost function's value at each iteration of the optimization algorithm, negatively impacting performance.

To address this issue, the solution proposed by SGD is to interpret the gradient as an expectation, i.e., to assume that the total gradient can be approximately estimated by sampling only a small group of examples. Thus, for each iteration, a subset of samples is defined from the training group, also known as a mini-batch, where the gradient can be calculated as follows:

$$\mathbf{g} = \frac{1}{m'} \nabla_{\boldsymbol{\theta}} \sum_{i=1}^{m'} L(f(\mathbf{x}^{(i)}; \boldsymbol{\theta}), \mathbf{y}^{(i)}), \tag{12}$$

where m' represents the number of examples sampled for the mini-batch, and $\mathbf{x}^{(i)}$ and $\mathbf{y}^{(i)}$ represent an example and its correct label sampled for the mini-batch.

Once the gradient is determined, the optimization is performed using the same strategy as the gradient descent previously presented, taking steps in the opposite direction of the gradient and using a parameter known as the learning rate to control the step size performed at each iteration.

2.6.1.2 Adam

Among optimization algorithms, some also propose an adaptive learning rate. As presented before, the update during the optimization in SGD depends on two values: the gradient and the learning rate, which is a user-defined parameter. Thus, it can be complicated to define the best value for this parameter to get the best performance from the optimization algorithm. Also, optimization functions can be more sensitive to moves in one direction than the other, which is not considered within a fixed learning rate. One of the first attempts to solve this problem was the addition of momentum, which would move not only based on the current gradient but also consider the last movement made.

One of the most widespread algorithms among those adopting an adaptive strategy is Adam, a name derived from the expression “adaptive moments”. It uses a two-level momentum strategy. The first momentum takes into account the last calculated gradients and can be represented as follows:

$$\mathbf{s} = \beta_1 \cdot \mathbf{s} + (1 - \beta_1) \cdot \mathbf{g}, \tag{13}$$

where s is the first momentum, g represents the gradient of the function (estimated using the mini-batch strategy proposed by SGD), and β_1 is a decay rate for the momentum with value in the interval $[0,1)$.

The second momentum takes into account the square of the last gradients and is defined by the following equation:

$$\mathbf{r} = \beta_2 \cdot \mathbf{r} + (1 - \beta_2) \cdot (\mathbf{g} \odot \mathbf{g}), \tag{14}$$

where $\mathbf{r}$ is the second momentum, β_2 is a decay rate for the momentum with value in the interval $[0, 1)$ and $\mathbf{g} \odot \mathbf{g}$ represents the point-to-point multiplication of the gradient with itself, which is equivalent to the square of each of the partial derivatives that compose g.

However, since both momentums are composed of their previous value, there will be a more significant error during the first few iterations, where there will not yet be a history. To avoid this, Adam performs the following correction:

$$\hat{\mathbf{s}} = \frac{\mathbf{s}}{(1 - \beta_1^t)}, \text{ and} \tag{15}$$

$$\hat{\mathbf{r}} = \frac{\mathbf{r}}{(1 - \beta_2^t)}, \tag{16}$$

where ŝ is the first adjusted momentum, $\hat{\mathbf{r}}$ is the second adjusted momentum, and t represents the iteration in question. Thus, as the iterations go by, the effect of the correction tends to decrease.

Finally, with the two momentums adjusted, the movement can be calculated, with element-by-element operations, from the following equation:

$$\Delta\boldsymbol{\theta} = -\epsilon\frac{\hat{\mathbf{s}}}{\sqrt{\hat{\mathbf{r}}} + \delta}, \tag{17}$$

where ϵ represents the learning rate, δ is a small constant to avoid division by zero, and $\Delta\boldsymbol{\theta}$ is the move to be made in $\boldsymbol{\theta}$ to minimize the cost function $J(\boldsymbol{\theta})$ that is being optimized.

2.7 REGULARIZATION

During training, the model is only aware of some examples sampled from the data distribution, the so-called training set. Thus, although training is based on optimizing a cost function calculated over the training data, decreasing this value too much may cause the model to over-adapt to the training set data. This phenomenon known as overfitting. That means that instead of learning the training data distribution, the model will adhere to it, thus reducing its generalization ability and, consequently, its performance on the test set.

Regularization techniques exist to reduce such model overfitting. They modify the learning algorithm to reduce the generalization error, even if this means increasing the error in the training set. For this purpose, there are several techniques based on different approaches, such as reducing the weights of the model, discarding connections during training, and stopping training early, among other strategies. The following subsections will present some of the most common techniques in more detail.

2.7.1 Dropout

One of the most famous forms of regularization is Dropout, proposed in the work of Srivastava *et al.* (2014). This strategy uses as inspiration the principle that combining multiple models can result in improved generalization on the test set (BISHOP, 2006). However, training multiple models is expensive in the deep learning scenario. Thus, Dropout proposes discarding some neurons from the network during training to have different networks from the original architecture.

This discarding strategy consists simply of temporarily removing connections to a particular node. The choice of this node is made randomly so that at each training iteration, new neurons are dropped, resulting in a different net. Thus a Dropout probability p represents the chance of a neuron being discarded each time, creating a new hyperparameter for the model. This dropout, however, occurs only during the training phase. For the evaluation phase on the test set, the net uses all neurons in its architecture.

The main advantages of this method, as presented in Srivastava *et al.* (2014), are its simplicity, low computational cost, since there is no need to train n models separately, and its capability of being used in conjunction with other regularization methods.

2.7.2 Early-Stopping

The Early-Stopping strategy uses a simple principle. As the name implies, this technique is about stopping the model's training early when a model is seen to be overfitting to the training set. That is because training consists of reducing the cost calculated for the data in the training set. During training, it is expected that the same will happen for the test set, showing that the model is adapting to the data distribution. However, as it begins to overfit the training data, the model's error on the test data may rise, representing a decrease in the model's generalization ability.

Within this context, the idea behind Early-Stopping is to monitor the model's cost on the test set to stop training when it gets too high. To avoid using the test set during training, the ideal solution is to split the training set itself into two parts: training and validation. The new training will be used to optimize the model, while the validation will be used to monitor the model on a set not used directly in the optimization. So training will be stopped when the validation cost starts to rise for a given number of consecutive epochs, also called patience.

2.8 VALIDATION

The simple division of the dataset into training and testing groups can create a problem during the model training. This problem arises from the need to evaluate the model's performance during optimization, either because of a hyperparameter selection or by using an early-stopping strategy. Because, in this case, assessing its performance against either set would be problematic.

To begin with, scoring on the test set would present a methodological problem. After all, this would aim to increase the performance on this set directly, biasing the model and preventing the evaluation on the test set from actually representing what would be found in a real scenario: unseen data. On the other hand, scoring directly on the training set is not a better solution.

Because maximizing the performance on the training set can lead the model to over-specialize in this data set, leading to the so-called overfitting. Thus, the solution is to split the data into a third set, called the validation set.

This new validation set can be used to evaluate the models during training, either in the choice of hyper-parameters, where it is possible to choose the values that yield better performance on the validation set or using a strategy like early-stopping, where the model training could be halted when observed an increase in the error on the validation set, maximizing the model's performance on that set. Thus, it allows the test set to be kept isolated, allowing the model scoring to be performed on an as-yet-unseen data set.

2.8.1 K-Fold

However, the need for a new group can lead to a new and prevalent problem in deep learning: the amount of available data. This can result in a validation set that is too small, which can lead to a noisy model performance estimate during training.

To solve this problem, cross-validation solutions propose using the entire training set for validation in exchange for a higher computational cost. Among them, one of the most famous is K-Fold. This technique splits the data set into K partitions without overlapping data, with $K - 1$ allocated for training and one for validation.

This technique allows the model's performance on the validation set to be expressed by averaging the scores of each model on its own validation set. The downside of this strategy is the computational cost. Since one model must be trained for each partition as a validation set, K models must be trained using this technique.

Also, K-Fold has variations according to how the partitioning is performed. One of the most common is the stratified division of the partitions. This variation aims to create an even division of the examples between each partition (FORMAN; SCHOLZ, 2010). This means that each partition will have an approximately equal number of samples for each class, avoiding some imbalance that could occur during the division using only K-Fold.

2.9 MULTI-TASK LEARNING

As suggested by Caruana (1993), it may be easier for a model to learn multiple complicated tasks simultaneously than to learn them separately. This technique is also known as

Multi-Task Learning. In practice, this can be achieved by solving different problems with a model that shares parameters between tasks. According to Goodfellow *et al.* (2016), models of this nature can be divided into two parts:

1. Task-specific parameters. These only benefit the specific output of a task, and are therefore located near the output layers of that task;
2. Generic parameters. These are shared by all the other outputs of the model and, for this reason, are located at the very beginning of the network.

This parameter-sharing strategy can increase the model's generalization ability by bringing more information about the data to train the system. This benefit, however, can only be obtained if the tasks for which the model is being trained are related, i.e., if the problems the model proposes to solve share some information.

2.10 HIGH ACCURACY NILM DETECTOR – HAND

The detection step is critical for a large portion of NILM systems, allowing the identification of the appliances to be performed later. In this context, one of the existing detection methods is the HAND (NAIT MEZIANE *et al.*, 2017). It stands out mainly due to the simplicity of its operation and speed of execution.

The first step of its operation is to detect the signal's $e_d(t)$ envelope. The proposed method for this is to find the signal peaks at each power grid cycle (approximately 16.67 ms for a 60 Hz grid, for example). After this, the points between each consecutive pair of peaks found are calculated using interpolation. Since this can smooth the signal, eliminating abrupt changes that can help identify events, the algorithm also proposes that sets of values between two consecutive peaks with a meager median value be set to zero.

With the envelope prepared, the signal is analyzed. To do this, the envelope signal is divided into windows of size L, where this value represents the number of network cycles within each window. Then, for each window, a method is applied to iteratively compute the mean $\mu_d(t)$ and variance $\sigma_d^2(t)$ of the signal, as proposed by the algorithm itself, as follows:

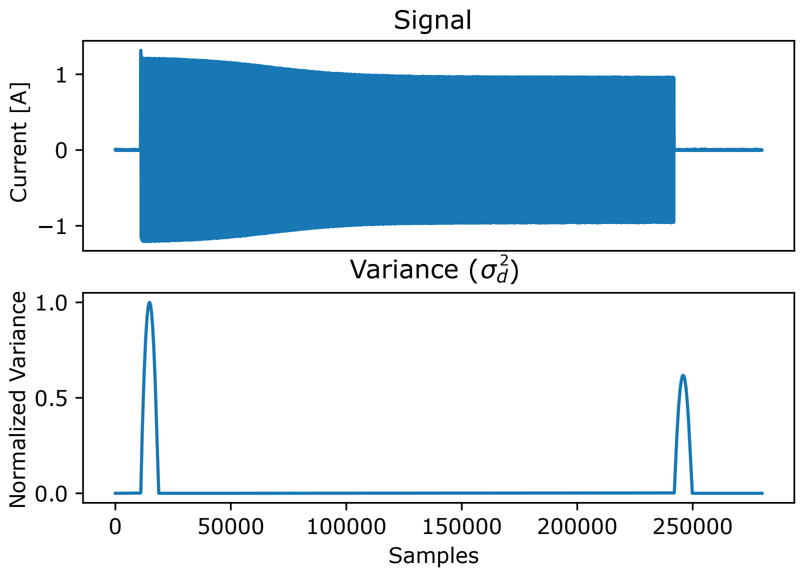
$$\mu_d(t_k) = \mu_d(t_{k-1}) + \frac{1}{L}\left[e_d(t_k) - e_d(t_{k-L})\right], \text{ and} \tag{18}$$

$$\sigma_d^2(t_k) = \sigma_d^2(t_{k-1}) + \frac{1}{L-1}\left[e_d^2(t_k) - e_d^2(t_{k-L})\right] + \frac{L}{L-1}\left[\mu_d^2(t_{k-1} - \mu_d^2(t_k)\right], \tag{19}$$

where $k = L + 1, ..., N$. The average will be used to calculate the window variance, which will be used to identify events in the signal.

An example of this resulting variance curve is presented in Figure 3, where the normalized variance curve for a waveform containing the current of a fan is shown. In this figure, one can see that the variance calculated from the envelope of the current signal is related to the events that occurred during that acquisition. When the appliance is turned on, where there is a greater variation in the current, there is a greater variation in the variance curve as a response. At the moment the load is turned off, although the variation in the signal is small, it is still possible to notice a change in the variance curve, allowing the identification of the switching event. Thus, HAND will identify as an event any variation in the variance curve at which $\sigma_d(t) > \gamma$, where γ is a user-defined threshold.

Figure 3 – Example of the proposed variance curve in HAND calculated for a fan signal.



Source: Own work.

This approach, however, is subject to a good definition of the parameters L and γ, the latter being the most important because a low threshold can result in the detection of a large number of false positives. A high threshold, on the other hand, can result in a large number of false negatives. Also, being a method based on the signal's amplitude variation, this strategy can present difficulties in scenarios with non-linear load combinations, which may result in a narrower amplitude variation. Another challenge is the power difference between loads. The variation caused by a low-power appliance may go unnoticed if the power demand of other devices on the power grid is too high.

2.11 ADDITIVE WHITE GAUSSIAN NOISE – AWGN

Additive White Gaussian Noise is a famous type of noise with some specific properties. As defined in Kuo (1996), an essential characteristic of white noise is the mean 0. It also holds the property that for the noises n_t and n_s, where n_t is the noise value at time t and n_s the value at time s, they must be independent for $t \neq s$.

Thus, white noise is composed of random and independent values, which results in a signal where all frequencies have a similar amplitude. This idea is what justifies the term “white” in its name. After all, the same behavior can be observed for the spectrum of white light. Visually, this noise can be represented as a flat signal without significant variations in amplitude.

On the other hand, the Gaussian part of the acronym AWGN represents that the noise can be modeled using a Gaussian distribution. Thus, since Gaussian is a probability distribution function defined by the mean and variance, a given Gaussian white noise can be defined as follows:

$$n \sim \mathcal{N}(0, \sigma^2), \tag{20}$$

where $\mathcal{N}(0, \sigma^2)$ represents a Gaussian distribution with mean 0 and standard deviation σ. This will result in a signal around 0 with an amplitude defined by the variance σ^2.

Meanwhile, the amplitude of this noise will define the Signal-to-Noise Ratio (SNR), which states the quality of the signal in relation to the noise present. This ratio is measured in decibels (dB) and can be calculated as follows:

$$\text{SNR} = 20 \cdot \log_{10}\left(\frac{A_s}{A_n}\right), \tag{21}$$

where A_s represents the signal amplitude and A_n represents the noise amplitude. Thus, the smaller the SNR value, the lower the signal quality since there will be a smaller proportion of signal to noise. Finally, the additive part of AWGN represents the property that this noise can be combined with the signal by a simple addition. This makes this noise a great option for experimentation because it can be easily generated and added to the original signal.

2.12 SCATTERING TRANSFORM

Proposed by Mallat (2012), Scattering Transforms (STs) represents the modulus and the mean of a Continuous Wavelet Transform (CWT). For this reason, its filters are computed beforehand, i.e., they are not learned during the training phase of the model, which results in a model with fewer trainable parameters, thus reducing the amount of data required to train it.

The ST layer consists of a convolution, a modulus (nonlinearity), and an averaging operation, and has a structure similar to a CNN, which usually has a convolution, an activation function (nonlinearity), and a pooling operation. Like CNNs, STs can be stacked in multiple layers. Thus, the coefficients of a layer q of a ST can be expressed by the following equation:

$$S_q = ||x * \Psi_{1,k}| \ldots * \Psi_{q,k}| * \Phi_J, \tag{22}$$

where Φ_J is a low-pass filter, Ψ is a wavelet function, x is the input signal, and S_q are the coefficients of the q-th layer. However, the output coefficients of a scattering network are a set of the coefficients for all the layers, as summarized by the following equation:

$$S^k = \{S_q(x)\}, \tag{23}$$

where $0 \leq q \leq m$, and m is the total number of layers in the scattering network.

Furthermore, as presented by Bruna and Mallat (2013), the scattering transform layers have information loss, with this information being propagated to higher orders (layers). Thus, the more orders in a scattering transform, the less energy is lost from the input signal. However, most of the information is conserved in the first orders, with up to 99% conserved in the 0th and 1st orders alone. This property allows numerical applications to limit the number of layers with a negligible loss of signal energy.

Finally, regarding the interpretation of the coefficients, due to the fact that the scattering transforms are based on wavelet filters, they are able to transform a time series input signal to the time-frequency domain (AGUIAR *et al.*, 2021). This property gives ST layers an advantage over CNN layers, making their output coefficients explicable, which allows the development of better strategies for dealing with the computed features.

2.13 ROC CURVES AND CONFUSION MATRICES

Receiver Operating Characteristic (ROC) Curves provide a visual way to analyze a classifier's balance between hits and false alarms (FAWCETT, 2006). For this purpose, the Receiver Operating Characteristic (ROC) curve uses the true positive rate ($\text{TP}_{\textit{rate}}$) and the false positive rate ($\text{FP}_{\textit{rate}}$). The following equation defines the true positive rate:

$$\text{TP}_{\textit{rate}} = \frac{tp}{p}, \tag{24}$$

where tp represents true positives, which indicates the number of instances where the model correctly identifies the load, and p represents total positives, which refers to the number of instances where the load is connected. The false positive rate, on the other hand, can be calculated as follows:

$$\text{FP}_{rate} = \frac{fp}{n}, \tag{25}$$

where fp indicates the number of false positives, which accounts for instances where the model mistakenly identifies a load as connected, and n indicates the number of negatives, which accounts for the total number of examples where the load is not connected.

For models that predict a continuous classification probability, it is necessary to set a threshold, usually 50% in problems with only two classes, to convert that information to a discrete value that indicates the predicted class. In the case of the ROC Curve, however, the curve is formed by multiple points obtained when varying this classification threshold, each point consisting of pairs (FP_{rate}, TP_{rate}), allowing the classifier evaluation for different threshold values. Thus, the ideal case for a classifier is to be at the point $(0, 1)$, indicating that the model identifies all positives correctly and no false alarms occur, so the closer to the upper left diagonal, the better the performance of the classifier. Finally, another performance measure from the ROC Curve is the Area Under the Curve (AUC), which has a maximum value of 1.

As a complement, confusion matrices are helpful in evaluating the occurrence of false positives in detail. This technique allows the visualization of the successes and errors of a classifier by relating the actual labels with those predicted by the model. For multi-label models, which allow the output for each class to be interpreted as an independent binary classifier, there will be an independent confusion matrix for each class.

3 LITERATURE REVIEW

This chapter presents the state of the art of using deep learning techniques in the context of non-intrusive load monitoring. For this, it first presents works using similar technology and their results. Finally, it presents the contributions of this work to the NILM field.

3.1 RELATED WORKS

This Section presents the literature review and briefly discusses correlated works from the NILM setting and their results. Since this work focuses on using deep learning techniques, which have achieved superior results in the recent literature (SONG *et al.*, 2021; PICCIALLI; SUDOSO, 2021), this Section will emphasize correlated research within this area in the NILM field. The advantage of deep learning techniques is to provide an end-to-end architecture, allowing feature extraction and load classification using only the sampled signal as input to the models, learning optimized parameters across layers, and increasing the final classification result.

In this context, Kelly and Knottenbelt (2015) presented one of the pioneering works in using deep learning in this area for low-frequency disaggregation. That work compares three deep neural network architectures. Using the Long Short-Term Memory (LSTM) architecture, the model based on recurrent neural network technology presented promising results for loads with two states, i.e., loads whose mode of operation can be summarized as on or off. Following this idea, several studies have been presented to disaggregate loads at low frequencies, mainly using models based on Recurrent Neural Networks (RNN) (SONG *et al.*, 2021; GOMES; PEREIRA, 2020; PENG *et al.*, 2020; D'INCECCO *et al.*, 2020; JIA *et al.*, 2020; PICCIALLI; SUDOSO, 2021; YANG *et al.*, 2020; NALMPANTIS; VRAKAS, 2020).

In a different approach, Li *et al.* (2023) proposes using a combination of CNN and RNN to disaggregate loads on low-frequency NILM signals. This strategy achieved a Mean Absolute Error (MAE) of approximately 4.26 in the REFIT dataset. However, despite the superior performance on public datasets, such as UK-DALE, REDD, and REFIT, the main limitation of low-frequency methods is the low resolution for the measured features (RUANO *et al.*, 2019), which prevents the identification of particular load types (ZOHA *et al.*, 2012).

High-frequency methods, on the other hand, allow a more detailed analysis of the extracted features, improving the identification of loads and, consequently, the disaggregation of the measurements (IQBAL *et al.*, 2021). In Baets *et al.* (2018), the authors used images generated by different V-I trajectories as input for a model based on CNNs. For this strategy, the accuracy obtained for the PLAID and WHITED datasets, both public, was less than 80%.

Liu *et al.* (2018) proposes a similar approach. A deep learning model, pre-trained on a dataset for visual recognition, is transferred to train a classifier for NILM using images generated by color-coded V-I trajectories. This coding allowed adding more information to the generated image, such as the ratio between active and apparent powers. In contrast, the transfer learning technique allowed reducing the amount of data needed for training the models and still significantly increased the accuracy, achieving an F1-Score higher than 95%. Similarly, in Guo *et al.* (2020), a method for load identification based on Active Deep Learning (ADL) and the Discrete Wavelet Transform (DWT) is proposed. The authors presented a combination of three public datasets that reduced the number of samples required by 33% while maintaining the F1-Score greater than 90%.

Chen *et al.* (2022) propose a solution with temporal and spectral representations of the appliance's power signature. Combining both, the authors presented the Two-Stream Convolutional Neural Network (TSCNN). Both representations are image-based, requiring extra preprocessing steps to extract the measured signal's temporal and spectral load signatures. That also implies 2-D CNN layers, increasing the model complexity, making training more expensive, and increasing the data needed to achieve good performance results. Despite that, the authors achieved an F1-Macro superior to 95% for both PLAID and WHITED.

In Mukaroh *et al.* (2020), a Generative Adversarial Network (GAN) based model removes background noise from multiple complex waveforms. By combining this model with a CNN-based classifier, the classification accuracy reached approximately 92%. On the other hand, Luo *et al.* (2023) propose a metric-based meta-learning strategy using 1-D CNNs, which allows the model to train using only a few examples for each appliance. Its goal is to improve the model's generalization on practical applications where labeled data is a limited resource. Using this strategy, the authors achieved an approximate F1 Score of 92% for WHITED and 76% for PLAID. Despite the promising results for classification, the main limitation for Baets *et al.* (2018), Guo *et al.* (2020), Liu *et al.* (2018), Mukaroh *et al.* (2020) is the absence of a detection stage or an analysis of the impact of event detection on the load identification process.

To overcome the limitation imposed by detection, a method capable of detecting loads in the electrical grid is discussed in Baets *et al.* (2019), including the detection and identification of devices not previously seen. Electrical appliances are represented by their V-I trajectory and mapped to the feature space created by a Siamese neural network so that examples of the same load form compact groups in this designed space. Then, grouping the data with a clustering algorithm known as Density-Based Spatial Clustering of Applications with Noise (DBSCAN) resulted in an F1-Macro score within the 80-90% range for the PLAID and WHITED datasets.

Nevertheless, in this context, Zhou *et al.* (2021) proposes a complete framework for NILM, including detection, feature extraction, and classification. This framework also implements an online learning method to infer varied loads using a transfer learning method on a model with limited measurements. For this, two neural networks were used in two different stages: (i) LSTM to extract the spatiotemporal features from the gray-scale image generated by the measurements and (ii) a Probabilistic Neural Network (PNN) to classify the load type. Both applied models used transfer learning and deep neural network techniques. Finally, the proposed algorithm was implemented on an embedded device to present the effectiveness of its use in practice.

A typical limitation observed in the methods presented so far is that, although obtaining adequate identification of the loads is possible, a disaggregation step is still required. A viable strategy for this disaggregation step is using a simple current subtraction, as discussed in Lazzaretti *et al.* (2020), Wang *et al.* (2018), or a filter-based approach, such as the GAN-based model presented in Mukaroh *et al.* (2020). These solutions aim to isolate each load in an acquisition with multiple appliances. However, nonlinearity and overlapping devices with significant power differences may preclude such strategies' use. In addition, this strategy's operation is highly dependent on the accuracy of the detection method employed. As a solution for this problem, some recent studies have pointed out that an alternative to such approaches is using a multi-label classification model, as shown in Tabatabaei *et al.* (2017).

Following this trend, recently, in Faustine and Pereira (2020), an approach based on CNNs was presented for the multi-label classification of NILM current signals. In this proposal, the current signal is represented in image form using Fryze power theory, which allowed this model to achieve an F1-Score on the order of 90-95% for public datasets. However, the authors did not discuss the applicability of this method in a real-world scenario through an analysis of the computational performance of the proposed architecture. Similarly, Manca *et al.* (2022) propose a multi-label strategy using V-I trajectories and Weighted Recurrence Graph (WRG). To improve the multi-label performance, the authors propose to combine a single-label model with a multi-label model during the training stage. This solution achieved an F1-Score of approximately 91% on the PLAID dataset.

In a general overview, the practical use of deep learning-based approaches is under-explored in the literature. In this context, it is worth highlighting the research presented in Baptista *et al.* (2018). The authors proposed a model based on convolutional neural networks implemented in hardware to identify loads using voltage and current measurements employing the V-I trajectory. For the implementation, the authors chose as the platform a Field Programmable Gate Array (FPGA) board aiming to use parallel processing. This strategy achieved an absolute accuracy of approximately 78% on the PLAID dataset, with a processing time of about 5.7 ms. However, this work used only single-loaded waveforms.

Mulinari *et al.* (2022) present another work that explores the practical implementation of a NILM model. In this solution, the authors propose representing the loads using features extracted by a Fourier 2-D series from the signal's V-I trajectory. The authors then tested those features with different machine learning classifiers, such as Ensemble, Support Vector Machines (SVMs), and Decision Trees (DTs) – with the Ensemble yielding the best performance. This strategy achieved an F1 Macro of approximately 96% when evaluated on the LIT-Dataset, with an average processing time of 96.6 milliseconds. However, the authors did not explore the detection problem.

3.2 WORK CONTRIBUTIONS

Based on the criteria exposed up to this point, one can observe that the correlated works presented have several positive points and some limitations, as summarized in Table 1. Considering the limitations primarily, the main original contributions of this work can be summarized as follows:

- A new architecture based on convolutional neural networks to perform detection, feature extraction, and multi-label classification of the loads in high-frequency signals with a single model. As will be presented later, this proposed model takes inspiration from the YOLO (REDMON *et al.*, 2016) architecture, which performs multi-label detection and classification of objects in images;
- The results obtained are equivalent or superior (in most of the cases analyzed) to the state-of-the-art methods for the dataset evaluated;

- The proposed technique can be embedded in an electronic system, showing its applicability in a real-world use scenario for a NILM system;
- Discussed the applicability of Scattering Transforms (STs) by proposing and evaluating a deep learning model with ST layers to perform event detection and multi-label load identification in NILM signals. Although this specific research related to ST is at an early stage, the experiments performed so far were able to show how scattering transforms can help reduce the amount of data required to achieve good performance in such tasks.

Table 1 – Related work and contributions of this research. Concerning notation, "Y" indicates that the paper contains the feature in question, while "–" indicates the absence of such a feature.

Work	Detection	Multi Label	High Frequency	Embedded
Li *et al.* (2023)	–	Y	–	–
Song *et al.* (2021)	–	Y	–	–
Gomes and Pereira (2020)	–	Y	–	–
Peng *et al.* (2020)	–	Y	–	–
D'Incecco *et al.* (2020)	–	Y	–	–
Jia *et al.* (2020)	Y	Y	–	–
Piccialli and Sudoso (2021)	Y	Y	–	–
Yang *et al.* (2020)	–	Y	–	–
Nalmpantis and Vrakas (2020)	–	Y	–	Y
Luo *et al.* (2023)	Y	–	Y	–
Chen *et al.* (2022)	–	–	Y	–
Baets *et al.* (2018)	–	–	Y	–
Aguiar *et al.* (2021)	–	–	Y	–
Guo *et al.* (2020)	–	–	Y	–
Liu *et al.* (2018)	–	–	Y	–
Mukaroh *et al.* (2020)	–	–	Y	–
Faustine and Pereira (2020)	–	Y	Y	–
Baets *et al.* (2019)	Y	–	Y	–
Zhou *et al.* (2021)	Y	–	Y	–
Mulinari *et al.* (2022)	–	–	Y	Y
Baptista *et al.* (2018)	–	–	Y	Y
Manca *et al.* (2022)	–	Y	Y	–
This work	Y	Y	Y	Y

Source: Own work.

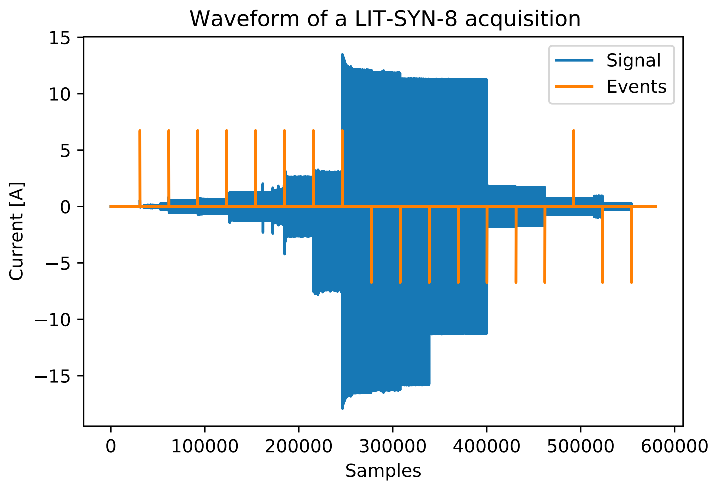
4 METHODOLOGY

This chapter describes the methods used to develop the solution proposed in this work. It consists of six sections. The first describes the database used to evaluate the model, the second details the architecture proposed to solve the problem, and the third section presents the procedure used to train and evaluate the model created. The fourth section describes the evaluation of the robustness to noise of the proposed solution, while the fifth section details the tests on an embedded system. Finally, the last section presents the architecture proposed for on the initial tests using scattering transforms.

4.1 LIT-DATASET

For model training and evaluation, the chosen data source was the LIT-Dataset[1]. This database, proposed by the Laboratory of Innovation and Technology in Embedded Systems and Energy (LIT), comprises three subsets: Natural, Simulated, and Synthetic. For this work, however, only the Synthetic subset, also known as LIT-SYN, was used since the Natural subgroup, which is also composed of real load acquisitions, was not yet fully available. Figure 4 shows an example of a waveform present in this subset of data.

Figure 4 – Example of an octuple waveform containing the loads: (i) Cell phone charger (Motorola), (ii) Fume Extractor, (iii) Soldering Station, (iv) LED Panel, (v) Vaio Notebook, (vi) Incandescent Lamp, (vii) Eleganza Hair Dryer, and (viii) Parlux Hair Dryer. In blue is the aggregate current of all loads. In orange are the load-switching events, where up indicates a turn-on and down indicates a turn-off.



Source: Own work.

[1] Available at: http://dainf.ct.utfpr.edu.br/~douglas/LIT_Dataset

The Synthetic subset of the LIT-Dataset (LIT-SYN) comprises acquisitions performed in a controlled environment using a bench. Here, actual loads are connected, but the activation of the devices happens in a controlled manner. This subset has 1 664 waveforms of up to 40 seconds, sampled at 15 360Hz. Concerning the loads, this subset comprises 16 appliances in different configurations, which result in 26 distinct classes, as detailed in Table 2. This dataset further provides waveforms of acquisitions with multiple loads connected simultaneously, including the following scenarios: one load (LIT-SYN-1), two loads (LIT-SYN-2), three loads (LIT-SYN-3), and eight loads (LIT-SYN-8).

Table 2 – Classes present in the synthetic subset of the LIT Dataset (LIT-SYN).

ID	Device	Power (W)
0	Microwave Standby	4,5
1	LED Lamp	6
2	CRT Monitor Standby	10
3	LED Panel	13
4	Fume Extractor	23
5	LED Monitor	26
6	Cell phone charger Asus	5
7	Soldering Station	40
8	Cell phone charger Motorola	10
9	Laptop Lenovo	70
10	Fan	80
11	Resistor	80
12	Laptop Vaio	90
13	Incandescent Lamp	100
14	Drill Speed 1	165
15	Drill Speed 2	370
16	Oil heater power 1	520
17	Oil heater power 2	750
18	Microwave on	950
19	Air heater Niko	1120
20	Hair dryer Eleganza - Speed 1	365
21	Hair dryer Eleganza - Speed 2	500
22	Hair dryer Super 4.0 - Speed 1 - Heater 1	660
23	Hair dryer Super 4.0 - Speed 1 - Heater 2	1120
24	Hair dryer Parlux - Speed 1 - Heater 1	660
25	Hair dryer Parlux - Speed 2 - Heater 1	885

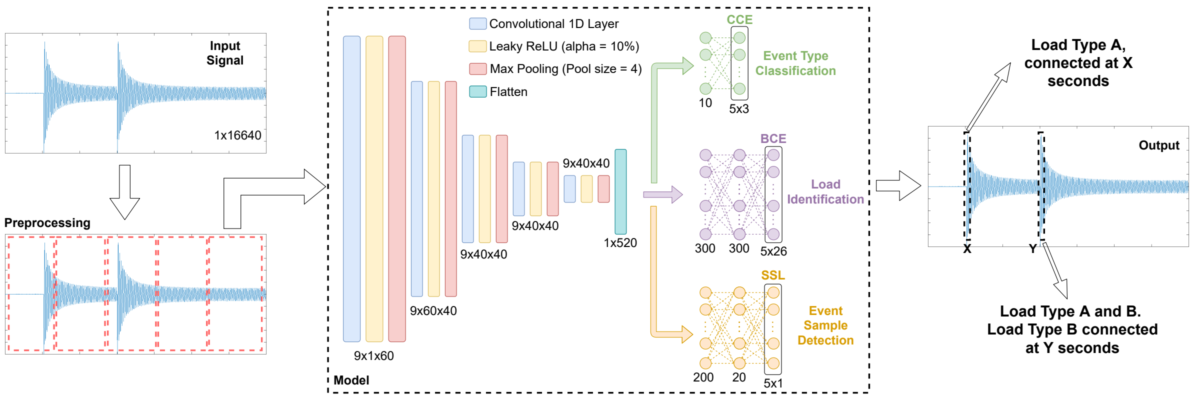
Source: Adapted from Renaux *et al.* (2020).

Another essential feature of this database is the existence of precise annotations (< 5 ms) of the load-triggering events, including the load responsible for such an event. That allows training not only the classification part of the events but also the detection part, something necessary for the proposed architecture.

4.2 PROPOSED METHOD (DEEPDFML-NILM)

Figure 5 presents the model proposed to solve the problems described so far. This model processes the measured aggregate electric current by a 5-layer block, each composed of a 1D convolutional layer, a Leaky ReLU activation function, and a Max Pooling layer. In the end, the resulting signal, after passing through this block, is transformed into an array by a Flatten layer, resulting in the features extracted from that signal until that point. These features, in turn, are redirected to three fully connected subnets that indicate the event type (turn-on or turn-off), the connected appliances, and the moment where an event occurred. During the training step, each of these sub-networks is optimized, respectively, by the Categorical Cross-Entropy (CCE), Binary Cross-Entropy (BCE), and Sum Squared Error (SSE) cost functions. The codes for reproducing the proposed method are publicly available[2]. It is worth noting, however, that the preprocessing step does not split the signal. The illustration represents how the model interprets the input, dividing it into grid regions, as detailed in the following sections.

Figure 5 – Overview of the proposed method.



Source: Own work.

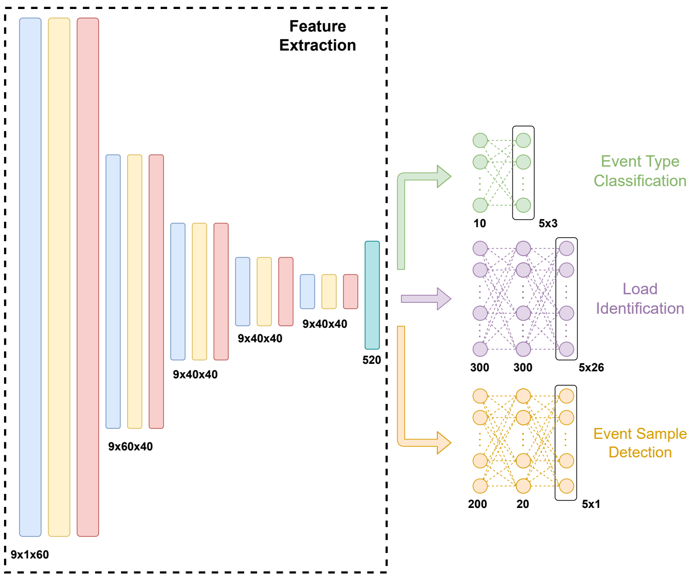
4.2.1 Architecture

The proposed model has three main parts, each responsible for predicting different information. The first of these parts is the event type classification, which indicates whether there is a load-switching event. If an event exists, the model classifies it as a turn-on or turn-off. The second part identifies the appliances, which, considering the analyzed signal excerpt, indicates the devices connected to the monitored electrical grid. Finally, the last part is event detection. This

[2] https://github.com/LucasNolasco/DeepDFML-NILM

step identifies in which specific signal sample an event occurred, in case there is an event in the signal excerpt. All three parts share the same convolutional layers to perform feature extraction, as shown in Figure 6. Sharing features among the outputs aims to improve the generalization ability of the model by proposing to solve multiple subproblems simultaneously, a strategy also known as Multi-Task Learning (see Section 2.9). All these combined results in a model containing 464 990 trainable parameters, also known as weights, to perform all these tasks. The base architecture, number of layers, and kernel sizes were inspired by the work presented in Ince *et al.* (2016), where the authors propose a 1-D CNN-based model for fault detection in motor current signals. From this base architecture, the model was empirically updated until obtaining the architecture presented in this work.

Figure 6 – Highlighted are the convolutional layers used to extract features shared by the three subproblems. The features extracted by this set of layers pass through each subnetwork, which will compute features for its specific problem.



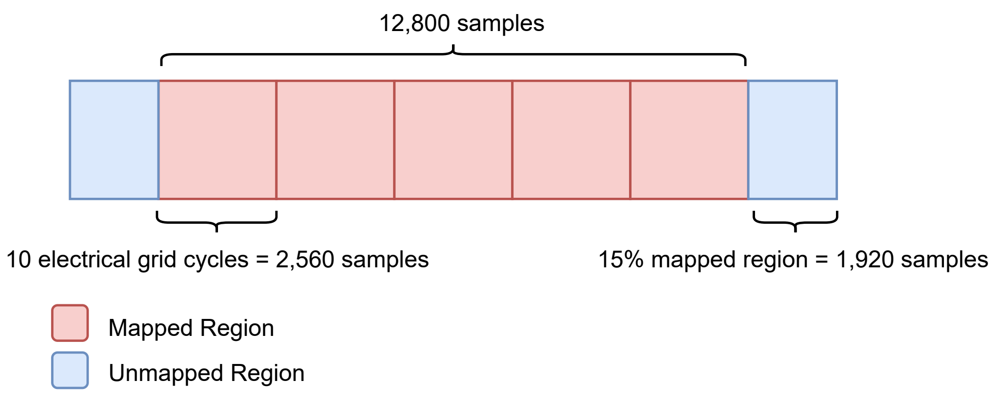
Source: Own work.

4.2.1.1 Model input

The architecture in question considers a division of the input signal into five segments, thus composing a grid of five positions, each containing ten cycles of the power grid. This division strategy increased the number of samples in the input signal without impacting the accuracy of event detection. The increase in the size of the input signal means that there will be more information about the connected loads, improving the quality of identifying the loads in the aggregated signal. However, if there is an event, it also means that there will be a more significant number of samples when it comes to defining which one is responsible for the detected event, something mitigated by dividing the signal into grids. Thus, this proposed division of the input signal enables the model to analyze the signal locally while having a broader view.

In addition to these five segments, the input signal comprises unmapped edges at the beginning and end of the analyzed signal, each corresponding to 15% of the total mapped samples, i.e., of the five segments already mentioned. These edges have this name because they are not mapped to the output labels, so the model is not trained to detect events or identify appliances that only appear in these signal portions. The idea is to mitigate problems caused by loads triggering too close to the edge of the analyzed signal, which could result in a low amount of samples to identify the load responsible for any possible event. With all this, the input signal will consist of five snippets containing ten network cycles each and the unmapped edges, totaling 16 640 samples, as summarized in Figure 7.

Figure 7 – Representation of the grid division used by the model to interpret the input signal.


12,800 samples
10 electrical grid cycles = 2,560 samples
15% mapped region = 1,920 samples
Mapped Region
Unmapped Region

Source: Own work.

4.2.1.2 Feature extraction

The feature extraction step, meanwhile, is the stage where the input signal is analyzed, and the model computes the features used by the fully connected layers. This stage is composed of five 1D convolutional layers, where the first one contains 60 filters, and the others have 40, all of them with a kernel size equal to 9. These convolutional layers use as activation function Leaky ReLUs with a leakage factor equal to 0.1. It means that for values greater than zero, the function will behave like the standard ReLU, while for values less than zero, the function will allow only 10% of the input value to pass to the output. Furthermore, after each convolutional layer, the signal is subsampled using max-pooling layers with a factor of 4. That implies that out of every four samples, only one will move to the output – the one with the highest value.

4.2.1.3 Model output

With the features computed by the convolutional layers, it is possible to proceed to the predictions. For the identification of the appliances, two fully connected layers containing 300 nodes each are used, with both using as activation function Leaky ReLUs with a leakage factor equal to 0.1 – the same used for the convolutional layers. In addition, between these two fully connected layers, there is a dropout layer with a rate of 25%. That will make the training process randomly drop a quarter of the values before they reach the second fully connected layer. The output layer's number of nodes is equal to the number of appliances in the database multiplied by the number of grid positions of the input signal, i.e., the output will have dimensions 5×26. Finally, since the identification of the loads happens in a multi-label scheme, sigmoid was used as the activation function because it does not assume a relationship between the nodes, which makes any combination of loads possible. This output was trained using Binary Cross-Entropy as the cost function, which can be described as follows:

$$BCE = -\frac{1}{M}\sum_{i=1}^{M} w_1 \cdot p(x_i)\log q(x_i) + w_0 \cdot (1 - p(x_i))\log(1 - q(x_i)), \tag{26}$$

where M is the total of examples, $p(x_i)$ is the expected probability for the load x_i, $q(x_i)$ is the probability calculated by the model, w_1 is the weight for when $p(x_i)$ is 1, and w_0 is the weight for when $p(x_i)$ is 0. The inspiration for these weights came from the work proposed by King and Zeng (2001). Here, they are used to lessen the impact of the imbalance between the number of examples for each load (for the calculation of these weights, see Subsection 4.3.2).

For the event type classification, the extracted features go through a fully connected layer with ten nodes, which uses as an activation function the same Leaky ReLU with 0.1 leakage factor already found in other parts of the model. This step's output uses one-hot encoding to represent its predictions, which, very briefly, means that the possible scenarios predicted by this output are mutually exclusive. Thus, three nodes are needed to represent the possible scenarios: (i) turn-on, (ii) turn-off, and (iii) no events. With this, the last layer presents the total number of positions in the grid multiplied by three, resulting in a dimension of 5×3. This output uses a softmax layer as the activation function and Categorical Cross-Entropy as the cost function. This function, meanwhile, can be represented as follows:

$$CCE = -\sum_{i=1}^{M} p(x_i) \cdot \log q(x_i). \tag{27}$$

Finally, the sample detection stage indicates precisely where the event has occurred. This subnetwork presents a similar configuration to the one in the load identification subnetwork – two fully connected layers separated by a dropout layer. The difference, in this case, is due to the number of nodes in each fully connected layer, with the first layer containing 200 nodes and the second 20 nodes. The output layer has dimensions 5×1 and uses Sigmoid as the activation function. Thus, this subnetwork predicts a value between 0 and 1 for each grid position in the input signal. This last layer uses Sum Squared Error as the cost function, which can be calculated as follows:

$$SSE = \sum_{i=1}^{M} \mathbb{E}_i \cdot (\hat{y}_i - y_i)^2, \tag{28}$$

where $\mathbb{E}_i$ denotes whether there is an event at position i of the *grid*, $\hat{y}_i$ is the model prediction for detection, and y_i is the expected output for this stage. Unlike the other outputs, which solve classification problems, detection aims to solve a regression problem, predicting a value between 0 and 1 to represent the sample where the event occurred.

4.2.2 Similarities and differences to YOLO

As stated earlier, this proposed architecture takes inspiration from YOLO (REDMON *et al.*, 2016), specifically YOLOv3 (REDMON; FARHADI, 2018). The first similarity is the use of convolutional layers. However, YOLOv3 uses 2D layers, while the proposed architecture features 1D layers. This change reduces the number of parameters in the model, making training less expensive.

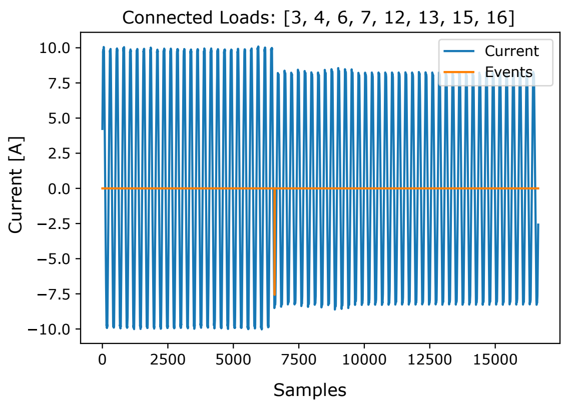
Another feature in both architectures is multiple outputs sharing the same convolutional layers for feature extraction. In YOLO's case, there is only one output for identifying objects and one for detecting their bounding boxes. In the case of the architecture proposed in this work, in addition to the outputs for identification and detection, there is also an output to indicate the type of event found, something necessary for the NILM problem. One more detail common to both models is the multi-label strategy classification. However, the system proposed by YOLO uses a hierarchical multi-label scheme based on the hierarchy proposed by WordTree, containing directly related classes, such as "dog" and "golden retriever". This concept creates a relationship between types, e.g., all golden retrievers will also be dogs. For the NILM scenario, however, that does not occur, which means that the model proposed in this work should be able to face any appliance combinations in a signal.

Figure 8 shows a comparison between the output of both models. For the output of the architecture presented in this work, given the nature of the problem, the detection consists of only one point. Another piece of information is the event type, which the event mark pointing downwards indicates as a turn-off event. Finally, regarding identification, the model correctly identified the eight independent loads present in this example.

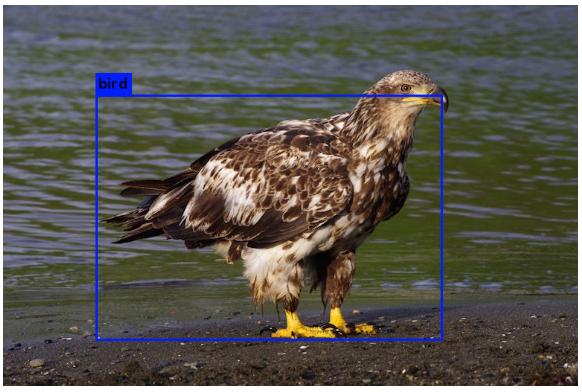
On the other hand, YOLO detects objects by predicting a bounding box that defines a region in the image. In the example presented in Figure 8, the model also identified the detected object as a bird. Nevertheless, a human being would be able to recognize that this bird is, in fact, an eagle. That represents the hierarchical concept used for the labels mentioned above. Although the image shows an eagle, the model can identify it with a more comprehensive but correct available class. Thus, the model can recognize an eagle as a bird simply by using features shared by other bird species in the dataset.

Figure 8 – Comparison between predictions generated by the proposed model and by YOLO. On the left is shown a detection performed by the architecture presented in this work, and on the right is shown a detection performed by YOLOv3.

(a) This work's prediction.



(b) YOLO's prediction.

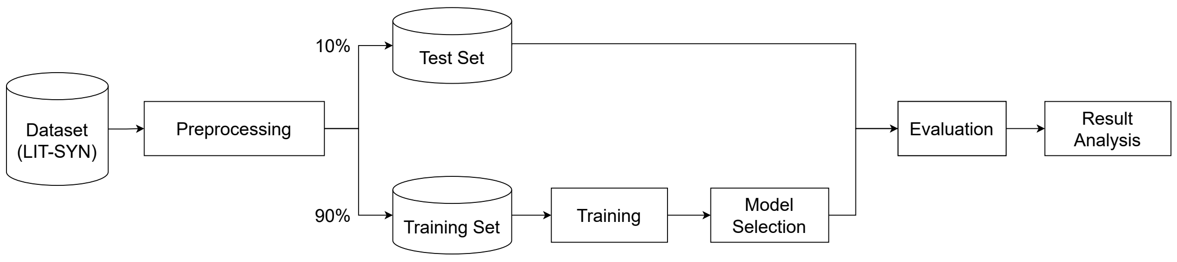


Source: Own work.

4.3 TRAINING

Figure 9 describes the steps required to train the architecture presented in Subsection 4.2.1. The first step is preprocessing the dataset to create the signal clippings and their labels. After that, the examples are divided into training and testing groups. With the set selected for training, the model is trained on a cross-validation scheme with ten folds, allowing a less biased evaluation of the split between training and validation. After this, the best-performing model is chosen during validation to evaluate the system's performance on the test set. The following subsections present these steps in more detail.

Figure 9 – Overview of the training and evaluation processes of the proposed model.



Source: Own work.

4.3.1 Preprocessing

Given that the acquisitions have millions of samples, which is much more than the model's input can handle, the first step in this phase is to create clippings of the signal. Therefore, it is necessary to choose which portions of the signal to use for model training and evaluation. Selecting these excerpts can be divided into two parts: (i) with events and (ii) without events.

4.3.1.1 Signal excerpts with events

The choice of clippings containing switching events is a simple task since the database already offers precise markings indicating the sample of occurrence of these events. Thus, to perform each clipping, the start of the selection window is positioned over the sample where an event occurs. This window is then moved using a pseudo-random distance smaller than the number of samples in the window. This procedure aims to improve the distribution of the position of the events within the clipping made.

4.3.1.2 Signal excerpts without events

For the choice of sections without load-switching events, as most of the signal will not have events, it is necessary to choose sections that present some interesting behavior that can pose a more challenging scenario during the model training. Furthermore, this choice must be reproducible, allowing the idea to be replicated by other works for performance comparison.

The High Accuracy NILM Detector (HAND) (NAIT MEZIANE *et al.*, 2017) algorithm was employed to solve this problem. This algorithm aims to detect events in NILM signals using the signal envelope based on a threshold of signal variation. However, defining such a threshold is a challenging task and may result in a large number of false positives or false negatives. In order to take advantage of this characteristic, parameters were chosen to ensure a high sensitivity of this algorithm to increase the number of false positives found, i.e., allowing the algorithm to detect variations in the signal that are not actual events.

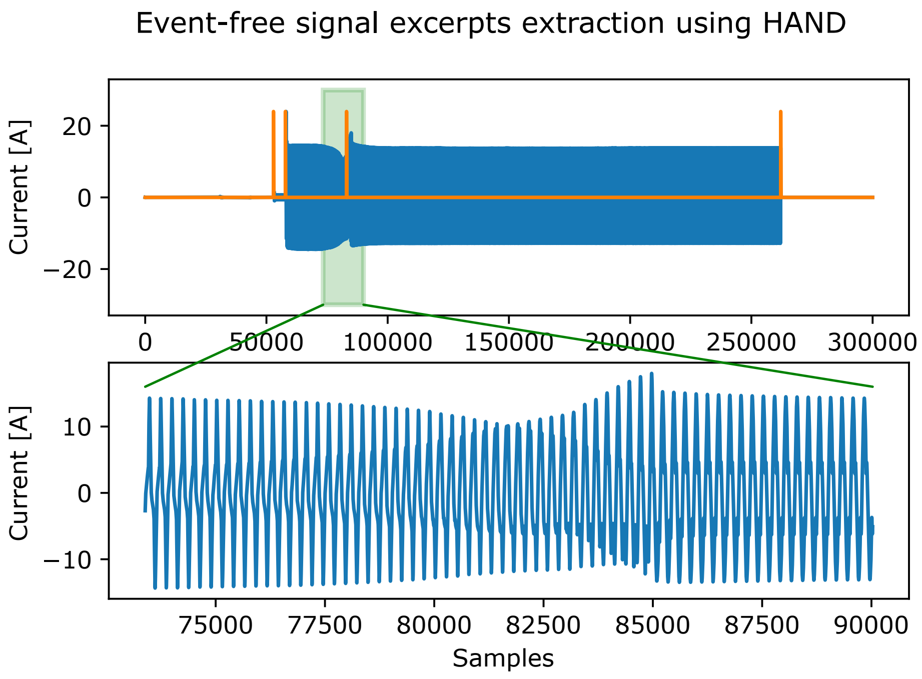
A window size $L = 1$ power grid cycle and a detection threshold $\gamma = 1 \cdot 10^{-5}$ were used as parameters (see Subsection 2.10). Also, to avoid clippings without actual current consumption, only detections with at least one load connected are chosen among the detected events. Furthermore, to avoid creating a clipping that is simply a translation of an existing example, only points with a distance of at least 50 grid cycles from other already chosen sections or load-switching events are selected. Finally, the same procedure described previously for the regions with events (see Subsection 4.3.1.1) is applied to create the clippings for the sections without events.

Figure 10 shows an example of a clipping obtained from applying HAND using the strategy presented here. The upper part of the figure shows an acquisition with a single load. However, as can be seen, after turning it on, the waveform still exhibits a significant current variation before finally entering the steady state. This current variation can be considered a switching event even by a human unaware of the load's behavior, making it a useful example for model training. This strategy generates 8 316 instances with events and 12 396 without events.

4.3.1.3 Creating the labels

With these signal excerpts, the labels provided for the model during training are created. Due to the architecture already presented, these labels will have three components: Classification of the event type, Identification of the loads, and Detection of the Event Samples. Detection consists of an array with a size equals to the number of grid positions, and its output is calculated

Figure 10 – Example of applying the HAND algorithm in choosing event-free excerpts in the signal.



Source: Own work.

as follows:

$$d_i = \begin{cases} 0, \text{if the grid does not have an event,} \\ \frac{S_e - G_i}{G_l}, \text{if the grid has an event,} \end{cases} \tag{29}$$

where d_i represents the value of the detection array for grid position i, S_e indicates the index of the event sample that occurred inside that grid position, G_i represents the first sample of that grid position, and G_l represents the total number of samples in a grid. For the event type, an array with three slots for each grid position is defined using one-hot coding, where the first slot is for turn-on events, the second is for turn-off events, and the third is for no events.

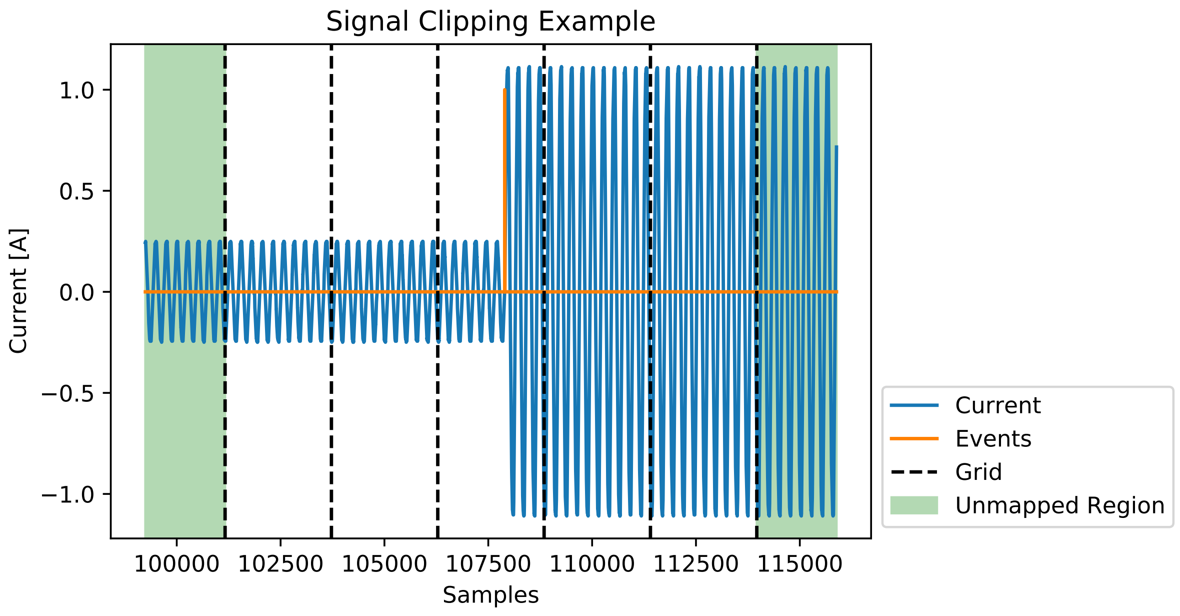
In load identification, an array with 26 positions represents the connected loads during the signal portion of a grid. Each of these positions corresponds to one of the loads present in the dataset (see Section 4.1). Given that the dataset provides the sample where an appliance is connected and disconnected, the proposed strategy to create the labels consists of calculating the intersection between a signal excerpt and each load. Then, for each grid position, if there is an intersection between that signal portion and an appliance, that appliance is considered connected in that grid position. Thus, if the appliance is connected even for a single sample of a grid position, the model will treat it as connected in that grid position. Finally, after creating the labels, the signal clipping window is extended by 15% on both sides to create the unmapped edges the proposed architecture considers.

Figure 11 presents a final example of the clipping created. In this figure, the orange indication shows the event marking provided by the database. At the same time, the black divisions represent the separation by grids, containing five central positions in addition to the two edges, indicated in the figure by the green regions. This excerpt of the signal in question comes from an acquisition with two loads, more specifically, from when the second load turns on. For this generated clipping, the label for detection would be:

$$y_{\text{det}} = \begin{array}{c} \text{Grid 1} \\ \text{Grid 2} \\ \text{Grid 3} \\ \text{Grid 4} \\ \text{Grid 5} \end{array} \begin{pmatrix} 0.00 \\ 0.00 \\ 0.63 \\ 0.00 \\ 0.00 \end{pmatrix}, \tag{30}$$

where the center position will be the only one with a non-zero value because it is the only grid position with a load-switching event, as defined by Equation 29.

Figure 11 – Example of a signal excerpt created using the proposed method.



Source: Own work.

As for event types, the training labels will be:

$$y_{\text{type}} = \begin{array}{c|ccc|} & \text{On} & \text{Off} & \text{None} \\ \text{Grid 1} & 0 & 0 & 1 \\ \text{Grid 2} & 0 & 0 & 1 \\ \text{Grid 3} & 1 & 0 & 0 \\ \text{Grid 4} & 0 & 0 & 1 \\ \text{Grid 5} & 0 & 0 & 1 \end{array}, \tag{31}$$

where each line represents a grid position with three possible events, namely: appliance turn-on (represented by $[1 \quad 0 \quad 0]$), appliance turn-off (represented by $[0 \quad 1 \quad 0]$), and no switching events (defined by $[0 \quad 0 \quad 1]$).

Finally, the equivalent label for identification will be:

$$y_{\text{class}} = \begin{array}{c|ccccccccccc|} & \rotatebox{90}{Microwave Standby} & ... & \rotatebox{90}{Cell phone charger Asus} & \rotatebox{90}{Soldering station} & \rotatebox{90}{Cell phone charger Motorola} & \rotatebox{90}{Laptop Lenovo} & \rotatebox{90}{Fan} & \rotatebox{90}{Resistor} & \rotatebox{90}{Laptop Vaio} & ... & \rotatebox{90}{Hair dryer Parlux} \\ \text{Grid 1} & 0 & ... & 0 & 1 & 0 & 0 & 0 & 0 & 0 & ... & 0 \\ \text{Grid 2} & 0 & ... & 0 & 1 & 0 & 0 & 0 & 0 & 0 & ... & 0 \\ \text{Grid 3} & 0 & ... & 0 & 1 & 0 & 0 & 0 & 1 & 0 & ... & 0 \\ \text{Grid 4} & 0 & ... & 0 & 1 & 0 & 0 & 0 & 1 & 0 & ... & 0 \\ \text{Grid 5} & 0 & ... & 0 & 1 & 0 & 0 & 0 & 1 & 0 & ... & 0 \end{array}, \tag{32}$$

where each row represents the connected loads in a given grid position, and each column position represents one of the loads in the dataset. From the third grid position, where the switching event occurs, the second load is considered connected and remains so until the end of the signal excerpt. In this representation, some columns have been hidden simply for easy visualization. Finally, Algorithm 1 summarizes the procedure for creating these labels.

4.3.1.4 Creating the training and test sets

Finally, with all the clippings done, these examples are divided into two parts, 90% for the training set and 10% for the testing set. That guarantees a test group, also known as a

Algorithm 1 – Algorithm for label creation

```
Input: the sample array $current$, the sample $S_e$ where an event occurs in the signal, and the list $C$ with the loads
    present in the signal and the samples where their switching events occur.
Output: current excerpt $x$ and labels $yDet$ (detection output), $yType$ (event type output), and $yClass$(classification
    output).
 1: $G_l \leftarrow$ number of samples for a grid position
 2: allocates yType, yDet, yClass
 3: $start \leftarrow$ applies the offset on $S_e$ and places the window over the event
 4: $nGrids \leftarrow$ number of grids used by the model
 5: for $g \in [0, 1, 2, ..., nGrids]$ do
 6:     $G_i \leftarrow start + g * G_l$
 7:     if there is an event on the grid $g$ then
 8:         yDet[$g$] $\leftarrow \frac{S_e - G_i}{G_l}$
 9:         if turn-on event then
10:             yType[$g$] $\leftarrow [1, 0, 0]$
11:         else
12:             yType[$g$] $\leftarrow [0, 1, 0]$
13:         end if
14:     else
15:         yDet[$g$] $\leftarrow 0$
16:         yType[$g$] $\leftarrow [0, 0, 1]$
17:     end if
18:     for load $l \in C$ do
19:         if $l$ connected in the grid position $g$ then
20:             yClass[$g$][$l$] $\leftarrow 1$
21:         else
22:             yClass[$g$][$l$] $\leftarrow 0$
23:         end if
24:     end for
25: end for
26: $x \leftarrow$ creates the expanded excerpt to add the unmapped margins
27: return $x, yDet, yType, yClass$
```

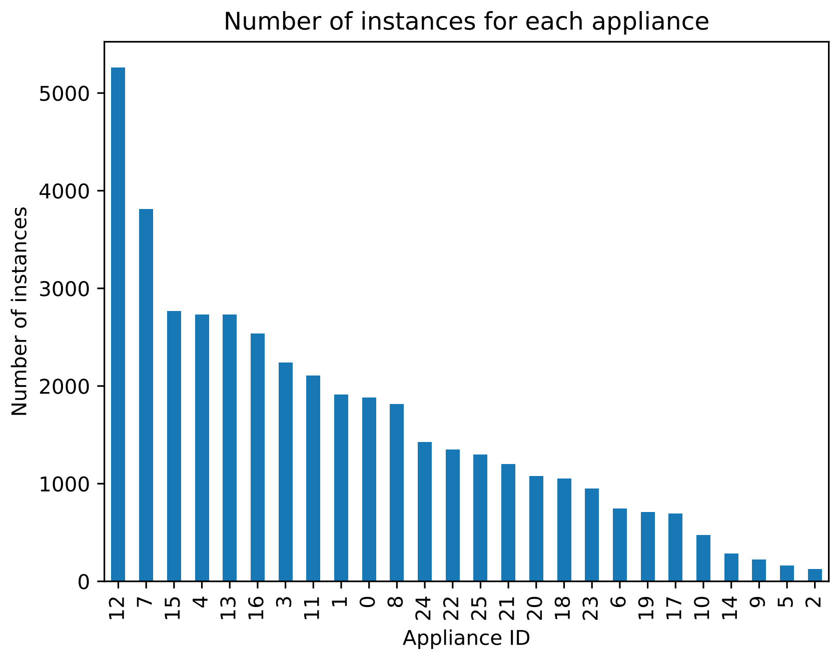
Source: Own work.

hold-out, that will not be used during training, reproducing what would happen during the use of the model in a real scenario. This division is done in a stratified manner based on the connected loads, aiming to create a balance between the number of examples for each load in the training and test sets. This choice is due to the significant imbalance among the loads, in which many loads have many examples while, at the same time, there are other loads with a small number of occurrences. That gives the distribution a characteristic known as long-tailed (CUI *et al.*, 2019). Figure 12 presents the distribution of the number of examples in which each load in the dataset appears.

4.3.2 Training and model selection

With the signal clippings and labels already prepared, it is finally possible to start training the model. For that, the k-fold cross-validation strategy is used, with $k = 10$. That

Figure 12 – Distribution of the number of instances for each load present in the dataset.



Source: Own work.

means that the training set will be divided into ten subgroups, where the model will be trained 10 times using each group once as a validation set, while the others will be used for training. After defining the training set, the input signal is normalized within this subset, keeping the values within the $[-1, 1]$ range to keep the model weights lower during training, avoiding a high sensitivity to variations present in the signal, such as noise. Then, to reproduce what would happen in the actual application of the model, the signals of the validation and test groups are normalized based on the thresholds obtained for the training set in question.

Still, in the training group, the weights used for training the classification are calculated (see Equation 26). Since identifying the loads is treated as a binary classification for each device, the weights are calculated individually. Thus, for each of the loads, there will be a weight for each of the two possible states (on or off). These weights, meanwhile, are calculated based on the following equation:

$$w_i = \frac{N}{2 \cdot n_i}, \tag{33}$$

where w_i indicates the weight for the i state of the load, n_i represents the number of examples containing the load in the i state, N represents the total number of instances, and i represents the states of the load (on or off). This formula is based on the work proposed by King and Zeng (2001), consisting of the inverse of the frequency for each state. It also divides the inverse frequency by the number of possible states to decrease the influence of the weights on the magnitude of the loss function.

Next, the training step starts. This procedure will use the early stopping strategy as a form of regularization, aiming to prevent the model from overfitting to the training data, which could reduce its performance on the test set. For this, the training process will monitor the loss function on both the training set and the validation set, and if its value for the validation set shows no reduction after a fixed number of epochs, the training stops. To ensure that the training process will always stop using the early stopping strategy, the number of epochs is 100 times greater than the patience. Table 3 lists all the parameters used for training the model. These parameters were empirically defined, being tuned during the experimentation phase.

Table 3 – Model training parameters.

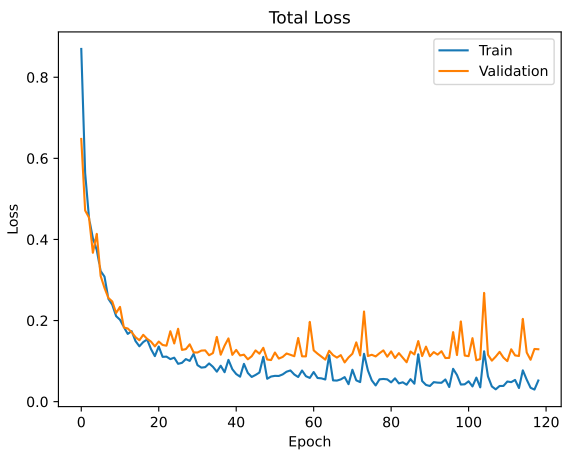
Parameter	Value
Epochs	5 000
Patience (early stopping)	50
Optimizer	Adam
Learning rate	10^{-3}
β_1 (Adam)	0.9
β_2 (Adam)	0.999
Batch size	32
Folds	10

Since this training step aims to optimize three cost functions simultaneously, it was possible to notice that some parts of the model optimized before others. More specifically, the load identification output, a more complex problem than the others, could still be more optimized at the end of training. That happens because the cost function increases faster in the other outputs, causing the total cost of the model to rise, leading to a stop in training due to the early stopping strategy. That becomes clear when looking at Figure 13. There, one can see how the cost of identifying the event type in Figure 13 rises faster than that of the other outputs. A similar behavior, but to a lesser extent, can also be observed for detection in Figure 13(d). In identification, on the other hand, the training and validation curves remain close, indicating that it may be possible to optimize that output further.

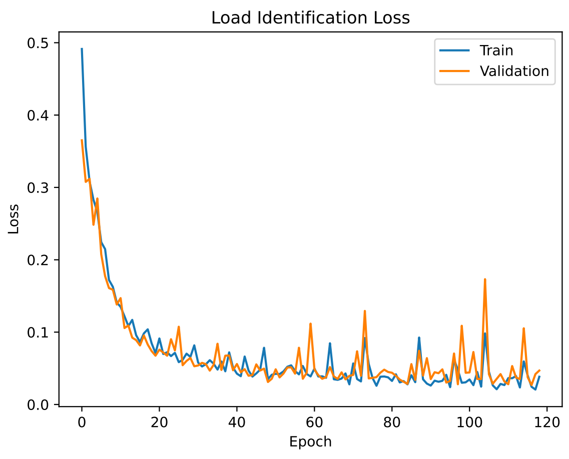
To solve this problem, after this first training step, all model layers, except for the fully connected layers of the load identification output, were frozen for a second training round. That allows training part of the model without interfering with the other layers. Because of its characteristic, this technique is usually found in transfer learning works, where layers with pre-trained weights are used, and in multitask learning works, where layers are shared among multiple subtasks (KHAN *et al.*, 2020; JIA *et al.*, 2019; SPANGHER *et al.*, 2021). In this second training stage, where only the identification of the loads is optimized, the same parameters

Figure 13 – Losses for training and validation sets during model training.

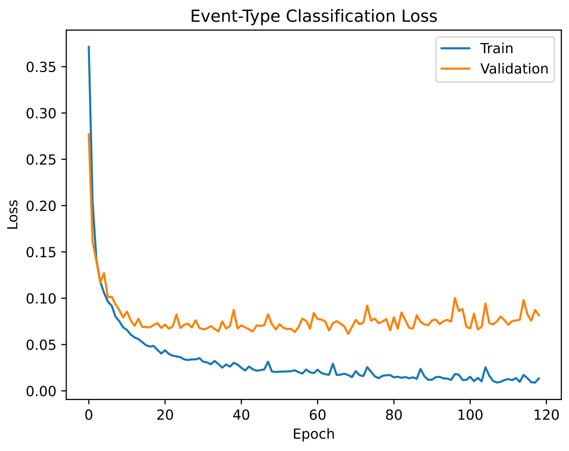
(a) Total Loss.



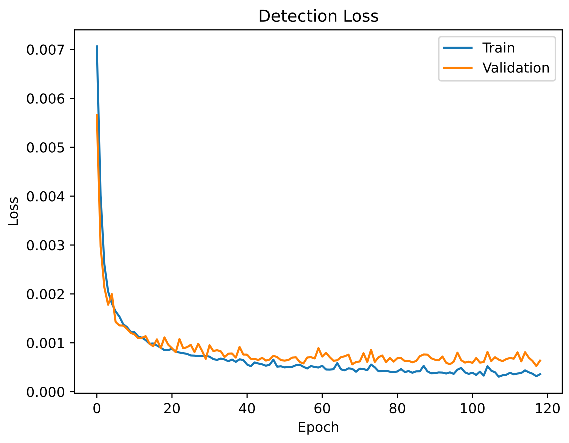
(b) Load Identification Loss.



(c) Event-Type Classification Loss.



(d) Detection Loss.



Source: Own work.

already listed in Table 3 were used, with a modification only in the learning rate, which decreased to 10^{-5}.

After training the model using cross-validation, it is time to choose the best-performing model within its validation set. However, since the architecture discussed here proposes to solve multiple problems simultaneously, it was necessary to decide how to evaluate which model performed best among all those trained during cross-validation. Thus, since detection showed similar performance among all the models trained, only the performance in identifying the loads was considered to simplify the decision. This performance, in turn, was calculated as follows:

$$F = \frac{F_{\text{on}} + F_{\text{off}} + F_{\text{noev}}}{3}, \tag{34}$$

where F_{on} represents the F_1 Macro (see Section 5.1) calculated for the instances containing a turn-on event, F_{off} represents the F_1 Macro considering only the instances presenting a turn-off of a load, and $F_{\text{no ev}}$ represents the F_1 Macro for the instances without events. The idea behind this averaging is to increase the influence of the examples with load-triggering events since only they present the transient characteristics of the devices in the signal.

4.3.3 Evaluating and processing the model output

As previously explained, the model has three outputs, each with different dimensions: the detection has dimensions of 5×1, the event type classification has 5×3, and finally, the identification of the loads has 5×26. Therefore, this subsection aims to describe the steps needed to create the event indications from all the information provided by the model.

The first step is determining the connected loads during the signal excerpt analyzed. That is done by identifying as connected loads all those that have a probability greater than 50% in at least one of the five grid positions. In other words, the load is considered connected if $max(P_L) > 0.5$, where P_L represents the array with the probabilities for a given load L in each of the grid positions that, in this way, will have dimensions 5×1.

The next step is to determine the events detected by the model. To accomplish this, the predictions of the output that indicates the type of event are checked. Then, in case of switching events, the sample where this possible event may have occurred is also checked for each grid position where a load-switching event occurred. The exact sample of the event occurrence is calculated from the detection output using the following formula:

$$S_e = \lfloor (D[g] + g) \cdot s_g \rfloor, \tag{35}$$

where S_e represents the sample number within the mapped region of the clipping, D represents the array with the prediction for detection calculated by the model, g indicates the grid position where an event was detected, and s_g represents the number of signal samples per grid position. Thus, the combination of the event sample, event type, and connected loads constitutes a detected event.

However, this strategy still allows the detection of the same event multiple times by neighboring positions in the grid. Therefore, event suppression is performed to avoid this situation. Hence, if more than one event is found for the same clipping, a relevance coefficient (C) is calculated for each of these events as follows:

$$C = \sum_{i=0}^{T-1} P_{CL}[i], \tag{36}$$

where P_{CL} represents the probability of the loads considered connected at the grid position where the event was found, and T represents the total number of loads considered connected.

Thus, the relevance coefficient will sum up the probability of the loads believed to be connected at the grid position where the event was detected. After calculating it for all detected events, the event with the highest C value is kept. Algorithm 2 summarizes this whole procedure of creating events from the model outputs.

Algorithm 2 – Algorithm to create the events from the predictions

```
Input: $d$ array with the detection predictions, $t$ array with the event type predictions and $c$ array with the connected
    load predictions.
Output: prediction $event$, in case there is an event that will contain the event occurrence sample, the event type,
    and the connected loads.
 1: $G_l \leftarrow$ number of samples per grid
 2: $l \leftarrow$ loads on $c$ with maximum probability greater than 0.5
 3: $events \leftarrow [\,]$
 4: $nGrids \leftarrow$ number of grids used by the model
 5: for $i \in [0, 1, 2, ..., nGrids]$ do
 6:     $type \leftarrow argmax(t[i])$
 7:     if $type \neq 2$ then
 8:         $ev \leftarrow \lfloor (d[i] + i) \cdot G_l \rfloor$
 9:         adds ($ev$, $type$, $l$) in $events$
10:     end if
11: end for
12: if there is more than one event in $events$ then
13:     $event \leftarrow$ chooses the event with the highest combined probability (see Eq. 36)
14: else
15:     $event \leftarrow$ the only event present in $events$
16: end if
17: return $event$
```

Source: Own work.

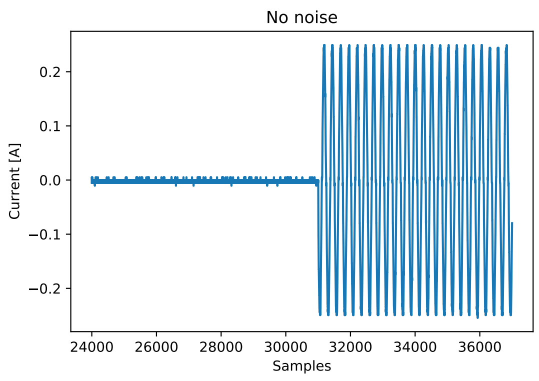
4.4 NOISE

This work includes a set of tests using different noise levels on the aggregated current signal to verify the proposed architecture's robustness to variations in the input signal's quality. These tests comprise five scenarios in which the signal degradation gradually increases using Additive White Gaussian Noise (AWGN), going from 50 dB to 10 dB SNR. Figure 14 shows the comparison of the effect of each of the noise scenarios on the original signal. In it, it is possible to see a drop in the signal's quality when decreasing the signal-to-noise ratio. Up to the scenario with 40 dB SNR there is a slight variation in the signal's shape. At 30 dB, however, it is possible to see a more significant influence of the added noise. At 20 and 10 dB SNR, the result is a considerably modified signal, creating a challenging scenario for the operation of the proposed model.

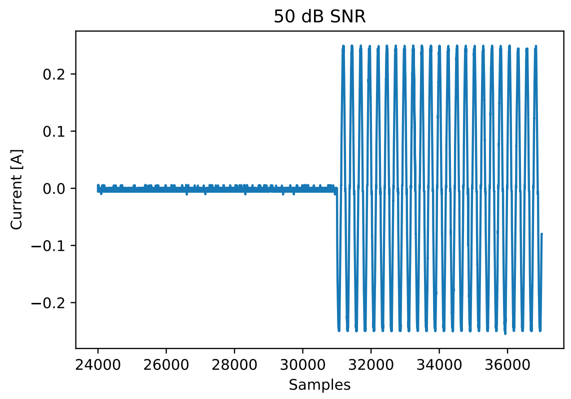
This test aims to evaluate the quality required for the sample acquisition system in the case of developing a system that uses the model proposed in this work. If the system performs

Figure 14 – Comparison of noise levels applied during the tests.

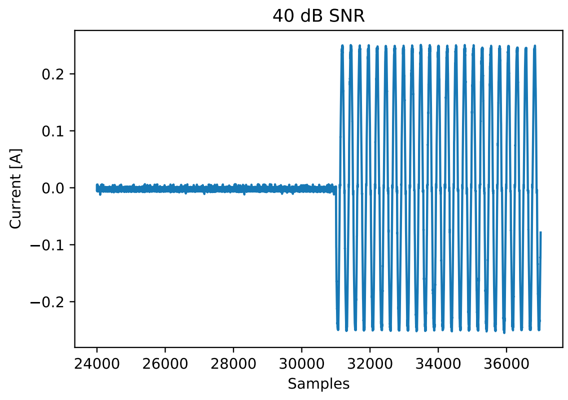
(a) No noise.



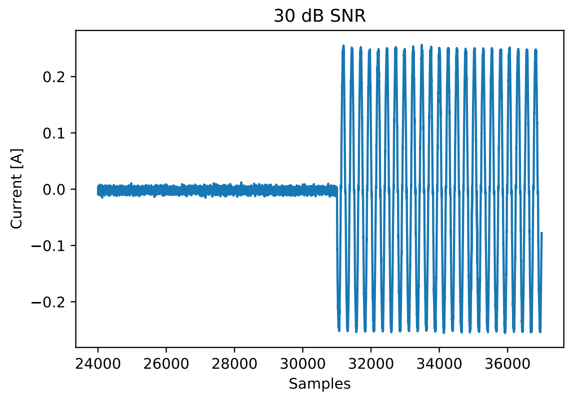
(b) 50 dB SNR.



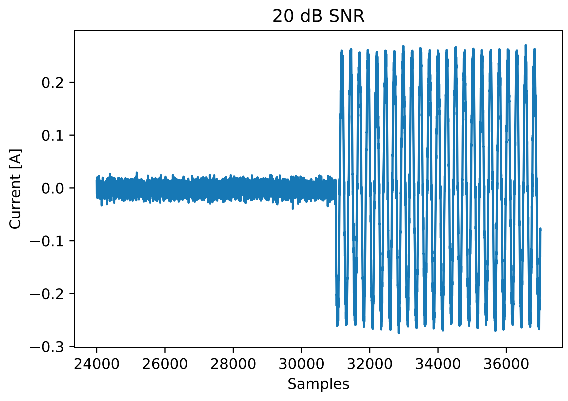
(c) 40 dB SNR.



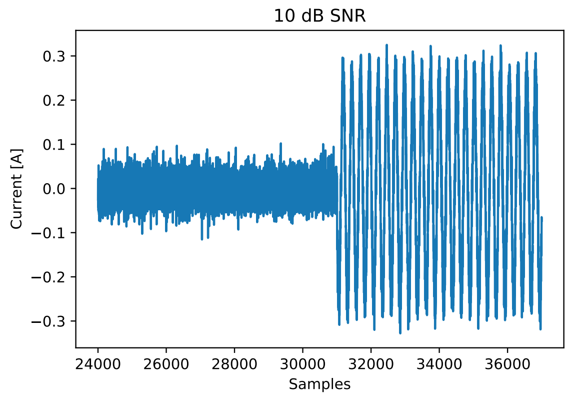
(d) 30 dB SNR.



(e) 20 dB SNR.



(f) 10 dB SNR.



Source: Own work.

poorly for more deteriorated signals, a better acquisition system will be required, consequently costing more. On the other hand, if the model proves robust in these scenarios, this would allow the use of a lower-level acquisition device, reducing costs and helping to make it more commercially viable. It is also worth noting that this type of noise analysis is recurrent in the NILM literature (WANG *et al.*, 2018).

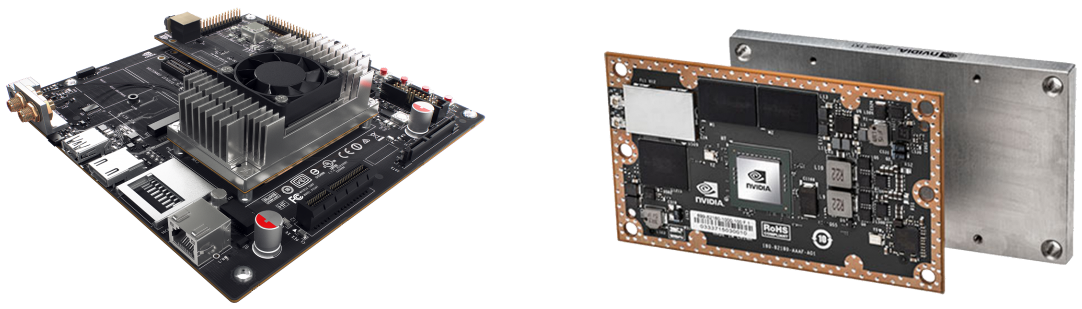
4.5 EMBEDDED SYSTEM

Another critical aspect of the applicability of such a system in a real scenario is its performance. That is because, for the model to be viable, it should be able to process a signal clipping in less time than it takes to get all those samples. Otherwise, the system would overload with more data than it can process. Thus, this work includes tests of the proposed architecture on an embedded system to assess this aspect[3].

The NVIDIA Jetson TX1 Developer Kit card was the platform chosen for these tests. It has a 256-core GPU, an ARM Cortex A-57 processor, 4GB LPDDR4 RAM, and a Linux environment (NVIDIA, 2015). These features made this board a good choice for the tests performed since it presents enough computational power to run the proposed model. Moreover, although the development kit offers considerable dimensions, measuring $170.18 \times 171.45 \times 15.58$ mm, as shown in Figure 15(a), its processing module, presented in Figure 15(b), has dimensions comparable to a credit card, measuring 87×50 mm. That makes this platform a viable option for prototyping and, consequently, an ideal scenario to verify the performance of the proposed model.

Figure 15 – Platform used for performance testing in an embedded system.

(a) Development kit. **(b) Processing module.**



Source: NVIDIA (2015).

4.6 SCATTERING TRANSFORM – ST

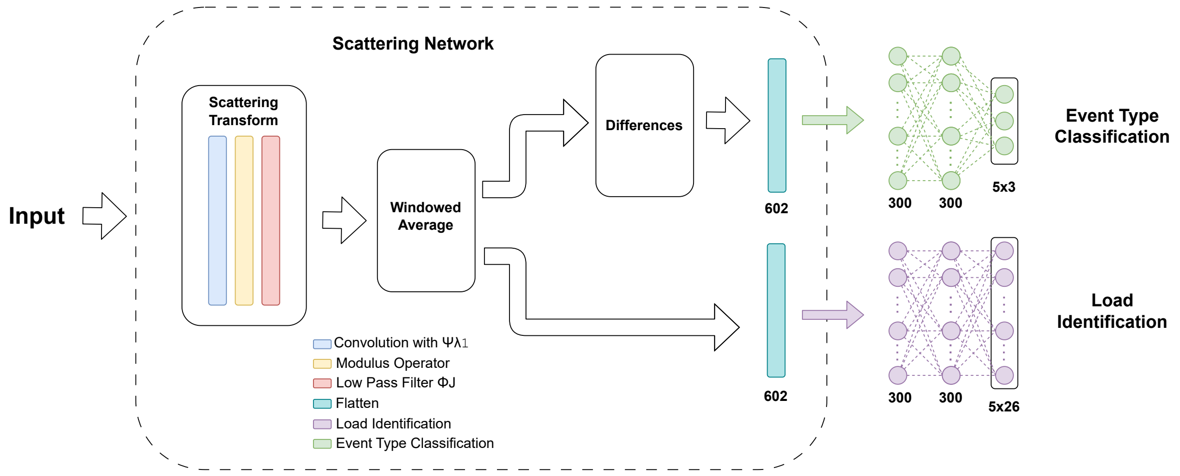
Deep learning models, such as the architecture presented in Section 4.2.1, typically require a large amount of data to achieve good performance. In an attempt to address this issue, this work proposes an alternative architecture based on Scattering Transforms (STs). The main

[3] The source code and instructions for reproducing the tests on the same embedded system are publicly available. More information can be found at: https://github.com/LucasNolasco/DeepDFML-NILM/blob/master/EmbeddedSystem.md

idea consists in replacing the block of shared CNN layers by a ST layer, reducing the number of trainable parameters and thus requiring less data to achieve good performance. However, it is important to emphasize that this is an ongoing research, which means that the results are still preliminary.

Figure 16 presents the proposed architecture using ST layers. Compared to the model based on CNNs presented in Subsection 4.2.1, this architecture based on ST layers present some important differences. The first and most prominent one is the replacement of the shared CNN block by the scattering network, which consists of a ST layer and some feature selection strategies.

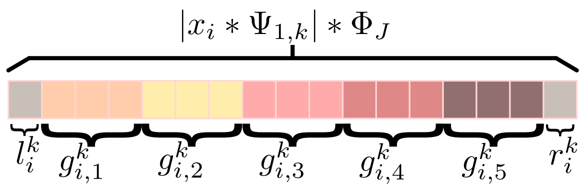
Figure 16 – Overview of the proposed model using Scattering Transforms (STs).



Source: Own work.

The output of the ST layer is a decomposition of the input signal and time and frequency (see Section 2.12). Thus, it allows the combination of these coefficients to reduce the output size, which consequently reduces the amount of parameters in the fully connected subnetworks. This combination consists of a windowed averaging, using the grid division concept used to interpret the signal, introduced in Subsection 4.2.1.1. Thus, the output coefficients of the ST layer are divided as shown in Figure 17.

Figure 17 – Grid division applied for the windowed average.



Source: Own work.

Then, the coefficients from each region are averaged and recombined as follows:

$$Sf_i^k = \begin{bmatrix} f_{i,l}^k & f_{i,1}^k & \cdots & f_{i,5}^k & f_{i,r}^k \end{bmatrix}, \tag{37}$$

where $f_{i,l}^k = \overline{l_i^k}$, $f_n^k = \overline{g_{i,n}^k}$, and $f_{i,r}^k = \overline{r_i^k}$. For the event-type classification task, there is another operation performed on the extracted features, the difference between coefficients.

The purpose of this operation is to highlight the variation in the signal caused by a load switching event. For this reason, the difference for a grid position is calculated as a simple difference between its two neighbors. The edge case is for the unmapped regions, which have only one neighbor. For them, the difference is calculated as the difference between them and their only neighbor. It is important to note that this should not affect performance, since the model is not trained to detect events on these regions. Therefore, the differences for the k-th wavelet filter and the i-th sample can be calculated as follows:

$$D_i^k = \begin{bmatrix} d_{i,0}^k \\ d_{i,1}^k \\ d_{i,2}^k \\ d_{i,3}^k \\ d_{i,4}^k \\ d_{i,5}^k \\ d_{i,6}^k \end{bmatrix} = \begin{bmatrix} f_{i,1}^k - f_{i,l}^k \\ f_{i,2}^k - f_{i,l}^k \\ f_{i,3}^k - f_{i,1}^k \\ f_{i,4}^k - f_{i,2}^k \\ f_{i,5}^k - f_{i,3}^k \\ f_{i,r}^k - f_{i,4}^k \\ f_{i,r}^k - f_{i,5}^k \end{bmatrix}. \tag{38}$$

Another difference is the output layers of the model. One can notice that the model has been reduced to only two outputs: Event-Type Classification and Load Identification. The regression output that predicts the time at which an event occurs has been removed for a couple of reasons. The first is the fact that the event-type output already provides information about where an event occurs in an interval of less than 167 ms. This leads to the second reason, which is to reduce the number of parameters in the model. Finally, the third reason was the poor performance in the earlier experiments, probably due to the averaging strategy used by the scattering layers.

The averaging strategy also created a challenging scenario for the event-type output, which motivated the calculation of the differences between features for each grid position. For the same reason, the subnetwork responsible for the event-type classification task was increased to two layers with 300 nodes – the same number of layers and nodes used for the load identification task.

Regarding the training process, the ST-based model uses the same procedure presented in Subsection 4.3.2. However, since the ST layer is not trainable, the subnetworks can be trained separately. This eliminates the need for the weight freezing strategy to retrain the load identification subnetwork. To make these experiments easily reproducible, the code for this proposed architecture is publicly available[4].

[4] https://github.com/LucasNolasco/ST-NILM.

5 RESULTS

This chapter presents the results obtained from the experiments using the methodology described in Chapter 4. In addition, it discusses an analysis of these results, as well as a comparison with other recent results from the literature. First, this chapter introduces the measures used to evaluate this performance. Then, it presents the results in the following order: (i) Detection performance; (ii) Classification performance; (iii) Visual example; (iv) Comparison with related works; (v) Noise robustness tests; (vi) Performance tests on an embedded system; and (vii) Preliminary results using Scattering Transforms (STs).

5.1 PERFORMANCE MEASURES

This section presents the measures used to assess the model's performance. For this purpose, it presents measures that calculate the model's ability to detect load-switching events and identify the loads composing the measured aggregate signal (classification).

5.1.1 Detection Performance Measures

This work proposes using two measures to evaluate the system's performance in the load detection task, as proposed in Lazzaretti *et al.* (2020). The first one aims at measuring the number of correctly detected events. In other words, it informs if the detected events are of the correct type (turn-on or turn-off) and within a predefined tolerance interval concerning the correct sample. Meanwhile, the other performance measure aims to calculate the average error of the correct detections, i.e., how far these predictions are from the actual event.

5.1.1.1 Percentage of Correct detections – PC

This work uses the Percentage of Correct detections (PC) measure to evaluate the correct detection of load-switching events. The main reason is that it has been used in other works in the NILM literature (LAZZARETTI *et al.*, 2020), enabling comparison with different approaches. This measure has two parts: PC_{on}, focused on turn-on events, and PC_{off}, which calculates the percentage of correct detections for turn-off events.

The measure PC_{on} can be defined as the ratio of the number of correctly detected turn-on events (A_{on}) to the total number of turn-on events recorded in the waveforms (N_{on}). This definition can be summarized in the following equation:

$$PC_{\text{on}} = \frac{A_{\text{on}}}{N_{\text{on}}}. \tag{39}$$

A tolerance window of ± 10 half-cycles of the power grid (60 Hz) relative to the event's actual occurrence has been used to define an event as successfully detected. That corresponds to a maximum absolute error of $1\,280$ samples. Thus, any detection within this range is considered correct (LAZZARETTI *et al.*, 2020).

Similar to the turn-on events, the PC_{off} is defined as the ratio of the number of correctly detected turn-off events (A_{off}) to the total number of turn-off events recorded in the waveforms (N_{off}). This definition results in:

$$PC_{\text{off}} = \frac{A_{\text{off}}}{N_{\text{off}}}, \tag{40}$$

where again a tolerance window of ± 10 half-cycles of the electric grid is used to define a detection as correct.

5.1.1.2 Average Distance – D

Another performance measure used in this work is the average distance (D) between a load-switching event and its detection by the model. Like the Percent of Correct detections (PC), this measure has two parts: D_{on}, which measures the average distance of detections for turn-on events, and D_{off}, which measures the average distance of detections for turn-off events.

The average distance to switch-on events, D_{on}, can be defined as the ratio of the absolute sum of the distances (in half-cycles) between detections and the actual event to the total number of correct detections, A_{on} (LAZZARETTI *et al.*, 2020). The following equation summarizes this definition:

$$D_{\text{on}} = \frac{\Sigma_{i=1}^{A_{\text{on}}} |d_{\text{on}}(i) - ev_{\text{on}}(i)|}{A_{\text{on}}}, \tag{41}$$

where $d_{\text{on}}(i)$ is the half-cycle in which the detection is located and ev_{on} is the half-cycle in which the event actually occurred. Note that only correct detections are taken into account for this measure, i.e., only those that increased A_{on} according to the definition presented for the calculation of PC_{on}.

The measure D_{off} can be defined in a similar way. It corresponds to the ratio between the absolute sum of the distances between turn-off event detections and the total number of correct turn-off detections, A_{off}. Thus, this measure can be represented by the following equation:

$$D_{\text{off}} = \frac{\Sigma_{i=1}^{A_{\text{off}}} |d_{\text{off}}(i) - ev_{\text{off}}(i)|}{A_{\text{off}}}, \tag{42}$$

where $d_{\text{off}}(i)$ is the half-cycle of the power grid in which the event was detected, and $ev_{\text{off}}(i)$ is the half-cycle in which the event occurred.

5.1.2 Classification Performance Measures

In this work, classical performance measures for classification problems, such as accuracy and F$_1$-score, are used to evaluate the identification of the loads that compose a given aggregated current signal. However, as mentioned in Subsection 4.3.1.4, the number of examples for the loads is unbalanced. Therefore, the performance measures employed use a macro-averaging strategy, where the measures are computed individually for each load and averaged to obtain the final performance estimate.

5.1.2.1 Macro Averaged Accuracy – ACC

Although not an ideal performance measure for multi-label classification problems, this work uses accuracy because it has been employed in other works in recent literature (MUKAROH *et al.*, 2020; LAZZARETTI *et al.*, 2020; MULINARI *et al.*, 2022). As a macro averaged measure, it is calculated as the average of the individual accuracies of each class, reducing the effects of imbalance present in the dataset (PEREIRA; NUNES, 2017). The following equation summarizes this definition:

$$ACC = \frac{1}{Y} \sum_{i=1}^{Y} ACC_i, \tag{43}$$

where Y is the total number of loads in the dataset, and ACC_i is the individual accuracy for each load. The accuracy of a particular device can then be calculated as follows:

$$ACC_i = \frac{CCE_i}{TNE_i}, \tag{44}$$

where ACC_i is the individual accuracy for each device i, CCE_i is the number of correct classified events for the device i, and TNE_i is the total number of load events for each device i.

5.1.2.2 F$_1$-Score

The F$_1$-Score, on the other hand, is a measure more commonly found in multi-label classification scenarios (FAUSTINE; PEREIRA, 2020; TABATABAEI *et al.*, 2017; YANG *et al.*, 2020; NALMPANTIS; VRAKAS, 2019). As seen for the accuracy, the macro average of the F$_1$-Score is calculated to deal with the problem of imbalance between the number of examples for each load. Thus, this measure can be defined by the following equation:

$$\text{F}_1 \text{ Macro} = \frac{1}{Y}\sum_{i=1}^{Y} \frac{2 \cdot tp_i}{2 \cdot tp_i + fp_i + fn_i}, \tag{45}$$

where tp_i is the number of true positives for load i, fp_i is the number of false positives, and fn_i is the number of false negatives.

5.2 DETECTION

Table 4 summarizes the results obtained for the detection of load-switching events using the proposed model and applying the methodology previously presented in this work. As can be seen, the proposed architecture can correctly identify most of the events, achieving a $PC_{\text{on}} > 95\%$ and $PC_{\text{off}} > 95\%$ when evaluating the entire database, represented in the table as the LIT-SYN subset. Furthermore, these detections have a low error regarding the correct event sample, showing $D_{\text{on}} < 1$ and $D_{\text{off}} < 1$, meaning that the event detections are majorly within less than a power grid half-cycle of the actual event.

Table 4 – Detection results.

Subset	PC$_{\text{on}}$ (%)	PC$_{\text{off}}$ (%)	D$_{\text{on}}$	D$_{\text{off}}$
LIT-SYN-1	97.1	100.0	0.9	0.7
LIT-SYN-2	97.6	97.8	0.6	0.6
LIT-SYN-3	96.2	95.7	0.9	0.7
LIT-SYN-8	88.5	90.3	1.0	1.0
LIT-SYN	95.1	95.9	0.8	0.7

Source: Own work.

However, when evaluating individually the groups that compose the LIT-SYN dataset, one can identify that the performance is superior for the cases where there is a smaller amount of connected loads, with very close values of PC_{on} and PC_{off} for the groups LIT-SYN-1, LIT-SYN-2, and LIT-SYN-3. On the other hand, there is a drop in performance for the case of the eightfold acquisitions present in the LIT-SYN-8 set, achieving a PC_{on} below 90% for the first time. Such a drop is to be expected, since more connected loads make it more challenging to identify a newly connected device.

Figure 18 shows an example of this problem. It presents a scenario where the system could not correctly detect the load-switching event. In this figure, blue represents the aggregated current, orange is the detected position for the event, and green is the event marking provided by the dataset. This signal is composed of eight loads, these being: (i) LED Panel; (ii) Fume Extractor; (iii) Soldering Station; (iv) Cell phone charger Motorola; (v) Vaio Laptop; (vi) Incandescent Lamp; (vii) Hair Dryer Eleganza; and (viii) Hair Dryer Parlux.

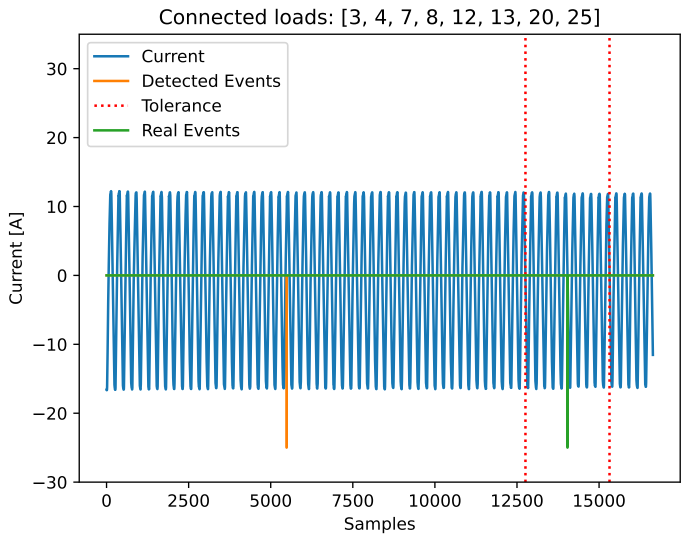
This error occurs because, besides the high number of connected devices, the load responsible for the turn-off event, the Soldering Station, has a lower power rating than most other connected loads. As already presented in Table 2, the Soldering Station has a power of only 40 W. Meanwhile, other connected loads in this example, like the Hair Dryers, stand out among the devices with the highest power consumption in the database, with the Parlux consuming 660 W and the Eleganza consuming 365 W.

The result is that the Soldering Station shutdown causes a much lower variation in the signal, making detecting this event a challenging task, even visually. Despite this, the system can identify the event and recognize a power off, as shown by the event mark pointing down in the figure. However, this detection is outside the tolerance of ± 10 half-cycles set previously, which makes this detection incorrect. For a visual intuition of this range, the example delimits the region where detection would be correct using the dotted red lines.

5.3 CLASSIFICATION

Table 5 summarizes the results obtained for the load identification task using the methodology proposed in this work. The first point to note in this table is the division of the calculation of the performance measures into four distinct scenarios based on the event types: (i) On, (ii) Off, (iii) No events, and (iv) All. The On scenario represents the subset of clippings with a power-on event. Consequently, the Off scenario comprises instances that have power-off events. Since this database has no events within a distance smaller than the input size of the proposed architecture, it is impossible to have examples of signal clippings with two events. Therefore, the turn-on and turn-off subsets do not overlap. The non-event set, on the other hand, consists of all instances that do not present an event, i.e., the signal snippets obtained by applying the HAND algorithm (see Subsection 4.3.1.2). Finally, scenario (iv) counts all examples in the test

Figure 18 – An example scenario where the model could not correctly detect an event. This clipping represents an acquisition of the LIT-SYN-8 at the time of the Soldering Station (40 W) shutdown, while other loads, such as the Hair Dryer Eleganza (365 W) and Hair Dryer Parlux (660 W), were connected. The indication of the connected loads uses the appliance ID defined in Table 2.



Source: Own work.

set, representing the union of the three subsets previously mentioned.

This division aims to facilitate the comparison with related works in the following sections. One should note that the division by the number of loads in an acquisition (LIT-SYN-1, ..., LIT-SYN-8), as seen in the detection analysis, was not used because of the performance measures chosen for the classification problem. That is because, in this division per number of connected devices, not all loads may have positive examples in all subsets for the test set, making it impossible to calculate the measures individually for the load and, consequently, making it impossible to use the macro average. Therefore, to simplify the problem, since there is a division by event type, this strategy of division by the number of loads was discarded for the classification.

That said, based on the macro averaged accuracy (ACC) for all the scenarios evaluated, the model could correctly identify most of the connected loads, reaching $ACC > 95\%$ in all cases. The highlight goes to the instances with no events where the system approached the maximum accuracy. The possible reason for this is that this scenario presents the connected loads in all samples of the signal clipping, which can be a problem in the clippings with an event. An example of such a problem would be a power-on event very close to the end of the window or a power-off near the beginning, scenarios where the model would have a small number of samples with one of the loads connected, making it challenging to identify it.

Another point to note, however, is the superior performance for the turn-off scenario

Table 5 – Classification results.

Scenario	ACC (%)	F_1 Macro (%)
On	97.9	94.9
Off	98.7	96.6
No events	99.9	97.8
Total	99.4	97.1

Source: Own work.

compared to the turn-on scenario. A possible reason for this phenomenon is how the dataset was created – by alternating the activation angles for all the devices. In LIT-SYN, an acquisition for each load combination consists of 16 waveforms, each with a different triggering angle, ranging from 0^o to 337.5^o with steps of 22.5^o (RENAUX *et al.*, 2020). Figure 19 compares the waveform difference for the power-on and power-off events of a microwave in standby. The figure shows that the change caused by varying the trigger angle is noticeable when turning on the device. The waveform has a positive spike when triggered in 0^o but presents a negative spike when triggered in 225^o. On the other hand, the turn-off behavior is very similar even with the change in the activation angle, which makes it a more straightforward problem to solve, justifying the superior performance.

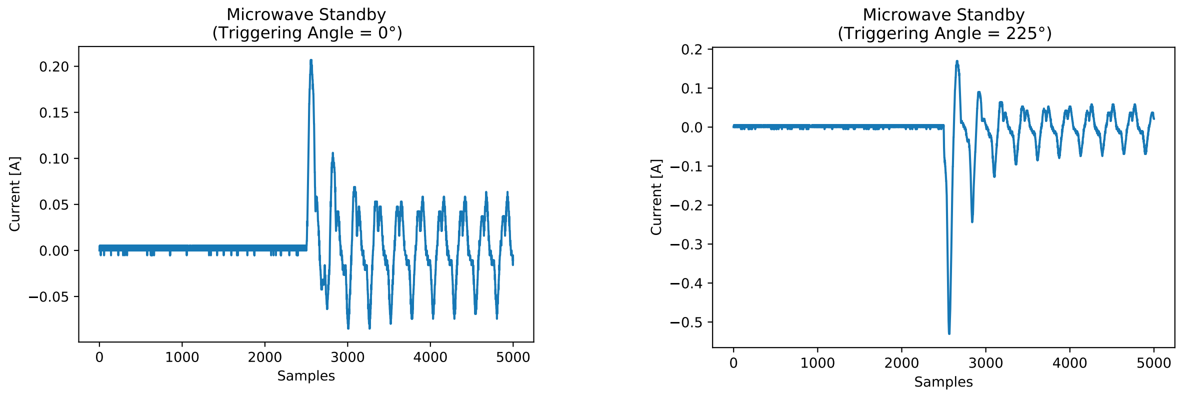
However, as mentioned before, accuracy may not be the best way to measure the model's performance, as it does not explicitly consider the occurrence of false positives. F_1-Score, on the other hand, does take this into account and, therefore, gives a better estimate of the system's performance. When comparing F_1 Macro with ACC, one can identify a slight drop in the values for all the evaluated scenarios. However, F_1 Macro remains above 94% in all cases, reinforcing the great performance of the proposed architecture. That further shows a low influence of false positives on the identification of the loads, which means that the proposed architecture can usually identify a load when it is connected.

Still, about false positives, Figure 20 presents the ROC Curves for the architecture proposed in the present work and their rounded AUC values. Since the output for identifying the loads indicates the individual probability of each device being on, each output acts as a binary classifier of two classes: on or off. Thus, each load will have its curve, all shown in this figure. In this same figure, one can see that the model presents a homogeneous performance among all loads evaluated, which is a positive aspect, especially considering the unbalance between the occurrences of the loads already mentioned in Subsection 4.3.1.4. Moreover, all loads presented good performance ($AUC > 0.95$), with curves that pass near the upper left diagonal.

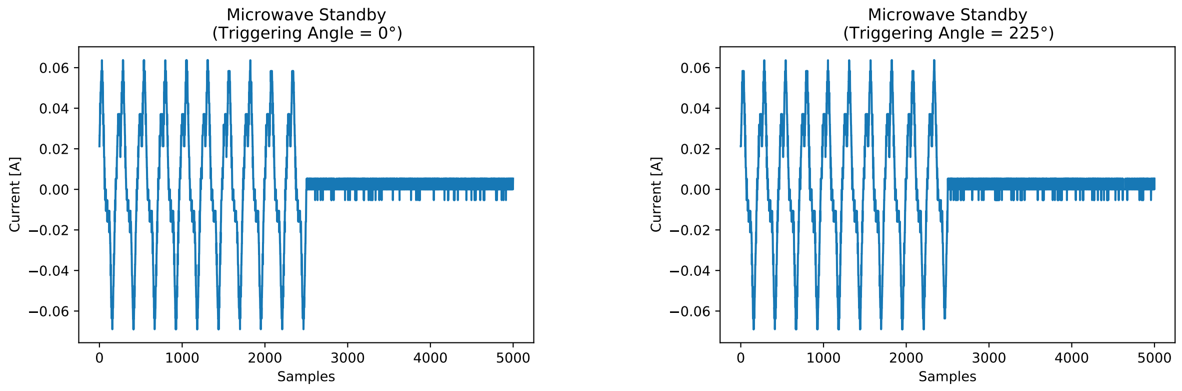
In addition to the information presented by the ROC Curves, Figure 21 shows the

Figure 19 – Comparison of the effect of the load triggering angle on the waveform for Microwave Standby switching events.

(a) Turn-on event when the triggering angle is 0^o. **(b) Turn-on event when the triggering angle is 225^o.**



(c) Turn-off event when the triggering angle is 0^o. **(d) Turn-off event when the triggering angle is 225^o.**

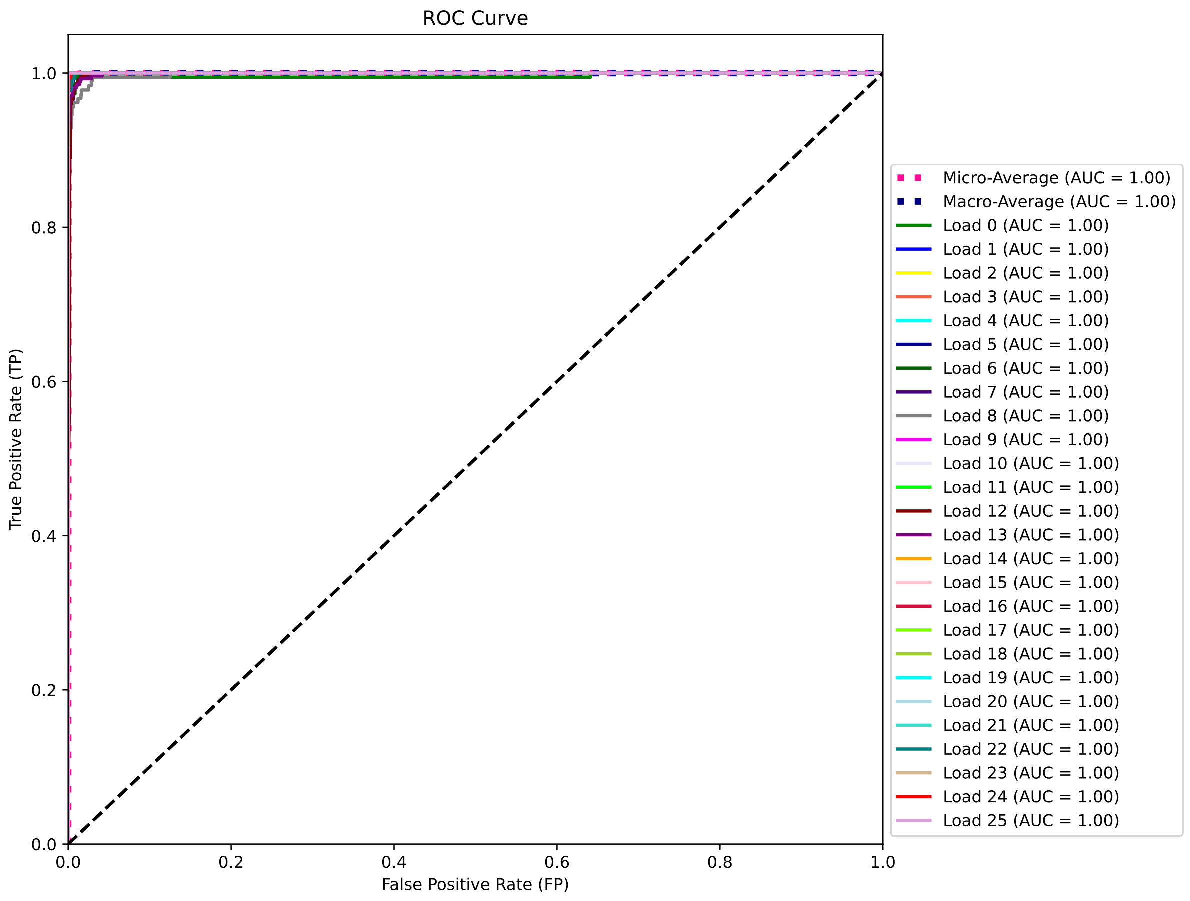


Source: Own work.

confusion matrices for the proposed model. For ease of visualization, it only presents a subset of the loads that stood out for some reason (positively or negatively). In this figure, one can observe the imbalance between positive and negative examples, as there are more examples where the loads are powered-off than examples where the loads are powered-on. Also, there are loads with much fewer positive instances, as previously mentioned. Despite this, the model can correctly identify connected loads most of the few times this occurs, indicating that the balancing strategy using weights may have collaborated in training the model.

Another point presented by the confusion matrices is a low occurrence of false positives. In the case of the Lenovo Laptop and the Cell phone charger Asus, despite having more than 2 000 negative examples, the number of false positives is less than 10. Here, the positive highlight goes to the Air Heater Niko, which had no false positives. The negative highlight, meanwhile, goes to the Cell Phone Charger Motorola, with 34 false positives. However, although this value is higher than that obtained by the other loads presented here, this is still a low number compared to the total of negative examples, reinforcing the great performance.

Figure 20 – ROC curves for all appliances in LIT-SYN.



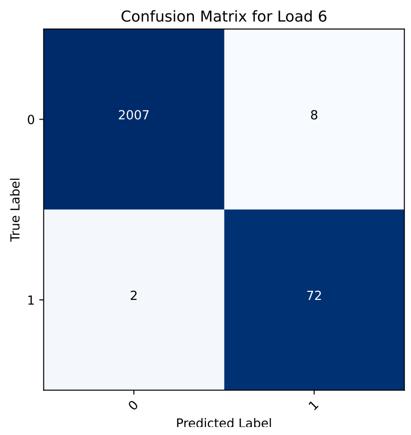
Source: Own work.

Regarding false negatives, the absolute numbers are also small. For this case, however, one should note a low total number of positive instances, which increases the impact of these false negatives. In this scenario, the Air Heater again stands out by not showing a single error. The negative example this time, however, is the Lenovo Laptop. Although it only has 2 false negatives, the low number of positives makes this value more significant than the other examples.

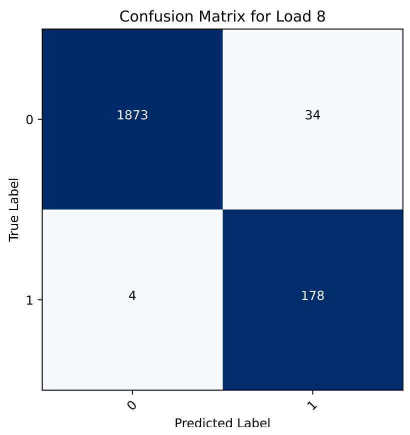
These errors are presented visually in Figure 22. Error 1 occurs in a shutdown scenario containing only the Lenovo Laptop. The model can identify the shutdown event correctly and within the tolerance limit. However, the identified load is 3, representing the LED Panel, as indicated in Table 2. In error 2, a similar scenario occurs, where the system can identify the event correctly but with a mistake in the device's identity. This time, however, the model can detect a laptop connected to the grid, represented by ID 12, which indicates the Vaio Laptop. Interestingly, the system again identifies the LED Panel as connected. A possible cause is the similar behavior of these loads, which may occur due to how the appliances are built, for reasons such as using semiconductor devices in their topologies.

Figure 21 – Confusion matrices for some of the devices in LIT-SYN.

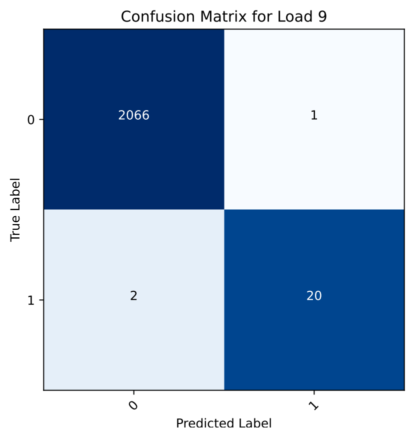
(a) Cell Phone Charger Asus.



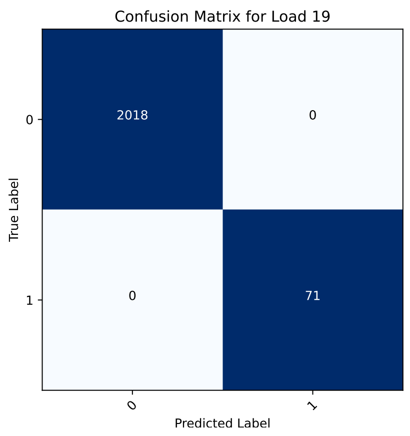
(b) Cell Phone Charger Motorola.



(c) Lenovo Laptop.



(d) Air Heater Niko.



Source: Own work.

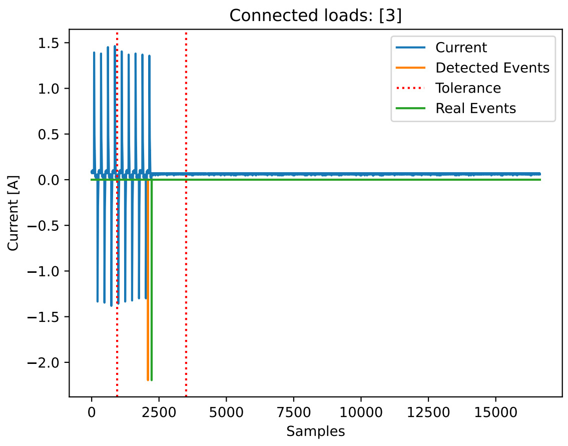
Checking these errors in more detail, one can see that the Lenovo Laptop and LED Panel waveforms share some similarities, as indicated by the mistakes made by the trained model. As shown in Figure 23, the LED Panel power-on response shows a current spike followed by a period of no consumption before returning to exhibit some signal, which one can also observe in the example presented for the Lenovo Laptop. For switching off events, the waveform is similar in both cases, at least visually.

Despite the similarity in shape, there is a difference in the scales. Looking at the figures, one can see that the laptop has a higher current consumption, especially at power-on. There is also a difference in amplitudes for the switch-off, but it is less than 1A. This more significant difference at power-on can justify why the model identified a laptop connected in addition to the LED panel. When turning the load off, only the panel is detected, as there is a smaller current amplitude difference.

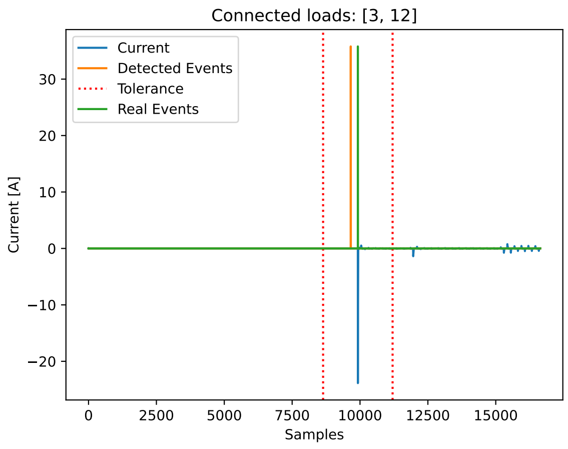
Finally, besides helping to identify these errors, the confusion matrices help reinforce

Figure 22 – False negatives observed for the Lenovo Laptop.

(a) Error 1.



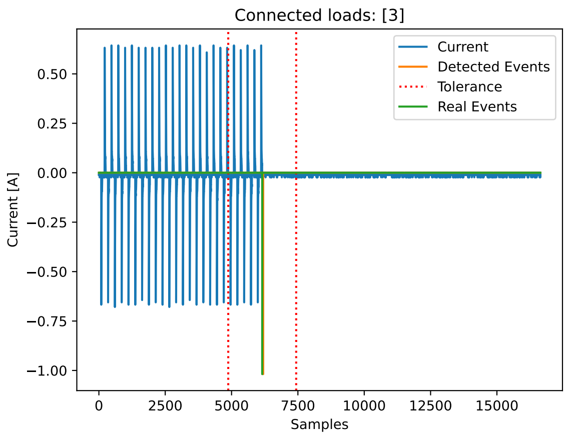
(b) Error 2.



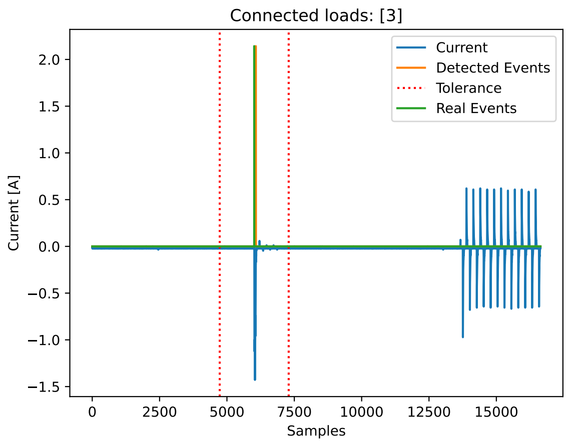
Source: Own work.

Figure 23 – Examples of waveforms containing the LED Panel.

(a) Power-off example.



(b) Power-on example.



Source: Own work.

the model's good performance in identifying the loads, already presented by the measures in Table 5 and by the ROC Curves and their area. Regarding the effect of using weights, adopted to balance the problem of load identification, Table 6 compares the results obtained for a model trained without using weights. To facilitate this comparison, Figure 24 presents the performance achieved with and without weights for all the scenarios evaluated and with both measures used for classification.

Table 6 – Classification results without using weights.

Scenario	ACC (%)	F_1 Macro (%)
On	90.8	91.0
Off	98.2	98.5
No events	99.3	98.9
All	98.1	98.0

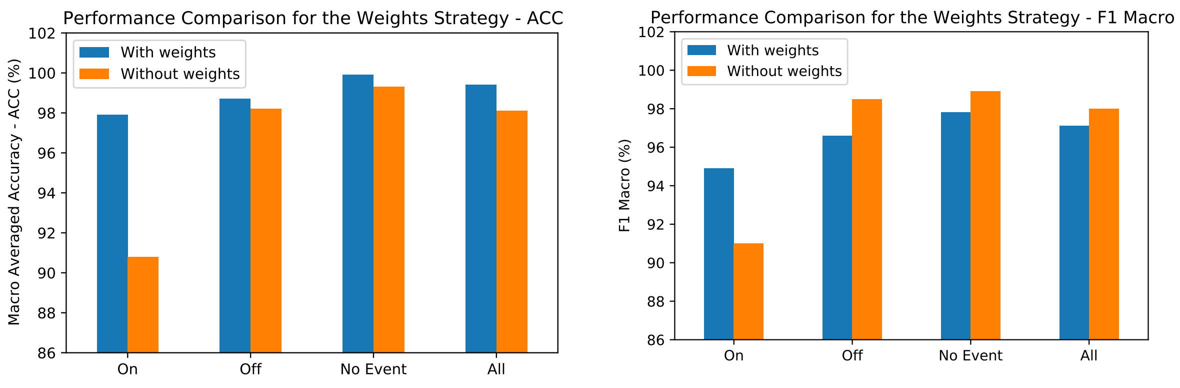
Source: Own work.

For ACC, one can see that using weights outperformed the model trained without weights. The highlight, however, is the turn-on scenario, where one can notice a performance increase of more than 5% when using the weights for training. Also, the performance between the scenarios for the model with weights is much more homogeneous when compared to the case without weights.

In the case of F_1 Macro, the difference is that the performance without weights outperforms the performance with weights in most scenarios but always with a slight advantage. The exception is the turn-on scenario, where the performance with weights is again superior. Furthermore, the power-on scenario has the most significant difference in performance between the two models (approximately 4%). That is important since the scenario with power-on events is the most challenging one for the model, as discussed earlier. That demonstrates this strategy as a simple and viable solution to balance the performance of the proposed architecture, something important in a multi-label classification scenario where problems tend to be unbalanced (ZHANG *et al.*, 2020).

Figure 24 – Comparison of the effects of using weights for training the load identification output.

(a) Macro Averaged Accuracy - ACC. **(b) F_1 Macro.**



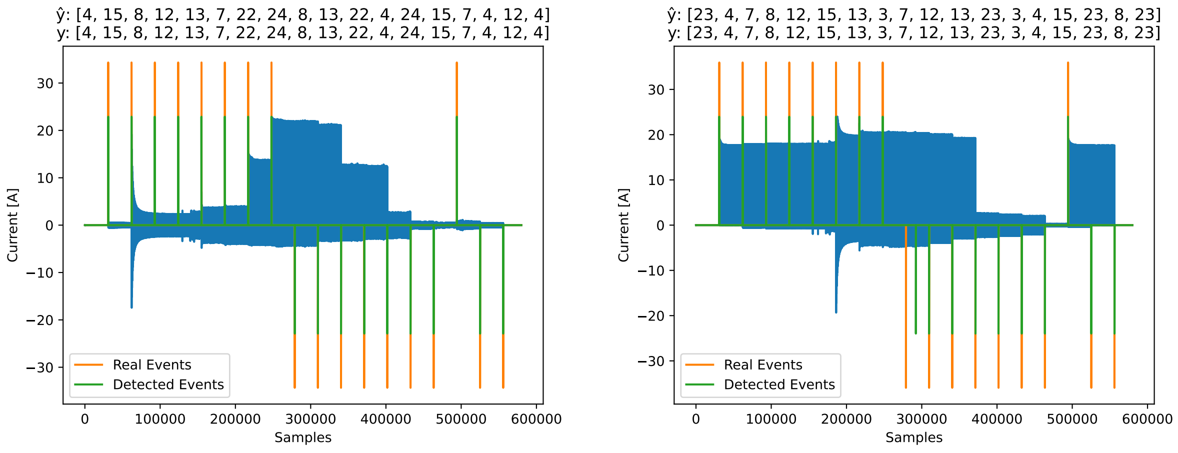
Source: Own work.

5.4 VISUAL EXAMPLE

To bring a more concise visual example of the operation of the proposed system, Figure 25 presents two examples of LIT-SYN-8 array waveforms processed by the architecture in question. The top of each figure shows the arrays with the identity of the loads responsible for each event in occurrence order, represented by $\hat{y}$, and those identified by the model, indicated by y. Only the devices responsible for the switching event are listed to facilitate visualization.

Figure 25 – Visual examples with acquisitions of the LIT-SYN-8 subset.

(a) Example 1. **(b) Example 2.**



Source: Own work.

As the model predicts all the simultaneously connected appliances, it is necessary to post-process that information to obtain the information presented in Figure 25. For that, the load with the highest probability difference was chosen between the grid position where the event occurred and a neighboring position. In the case of a power-on event, the comparison happens with the previous position, while for power-off, the comparison is with the position just ahead. The exception is in cases where a switch-on occurs at the first grid position or a switch-off at the last. For these scenarios, no load is chosen as responsible for the event.

In an effort to minimize this problem and improve the model's overall performance, these visual examples were generated by processing the signal in steps smaller than the model's input size. It used steps of the size of a grid position, ensuring that the same part of the signal was analyzed five times, each time from a different point of view. The multiple predictions are combined using the average for the event detection case and choosing the most frequently identified load as the one responsible for the events for identification.

The final result, presented in Figure 25, shows the actual events marked in orange and the detected events marked in green, where the upward markings indicate a turn-on while the downward markings indicate a turn-off. In these figures, one can see that the loads identified as the ones responsible for the events were correct in all cases for both examples, even though this is a more complicated scenario because it has a more significant number of loads. On top of that, the event detections remained close to what occurred, except for one event in Example 2, where the detection happened after the event. Despite this, the system could correctly identify the load responsible for such an event.

5.5 COMPARISON WITH RELATED WORK

This Section compares the proposed model's performance with related works from the literature to better understand the quality of the results achieved in this study. The comparisons are organized by detection and classification, which allows the comparison with works that do not approach the detection problem.

5.5.1 Detection

Table 7 summarizes the comparison with the Multi-Agent method proposed in Lazzaretti *et al.* (2020). For ease of visualization, the best performance on PC_{on} and PC_{off} for each of the load subsets has been highlighted in bold. For the case of acquisitions containing only one load, both models show similar performances, with the method proposed in this work excelling for most subsets in both turn-on and turn-off events.

Table 7 – Comparison of the detection results.

Method	Subset	PC_{on} (%)	PC_{off} (%)	D_{on}	D_{off}
This work	LIT-SYN-1	97.1	**100.0**	0.9	0.7
	LIT-SYN-2	**97.6**	**97.8**	0.6	0.6
	LIT-SYN-3	**96.2**	**95.7**	0.9	0.7
	LIT-SYN-8	**88.5**	**90.3**	1.0	1.0
Multi-Agent (LAZZARETTI *et al.*, 2020)	LIT-SYN-1	**97.8**	96.2	1.3	0.5
	LIT-SYN-2	96.3	92.5	1.2	0.7
	LIT-SYN-3	91.5	84.5	1.0	0.6
	LIT-SYN-8	64.9	60.5	1.0	0.9

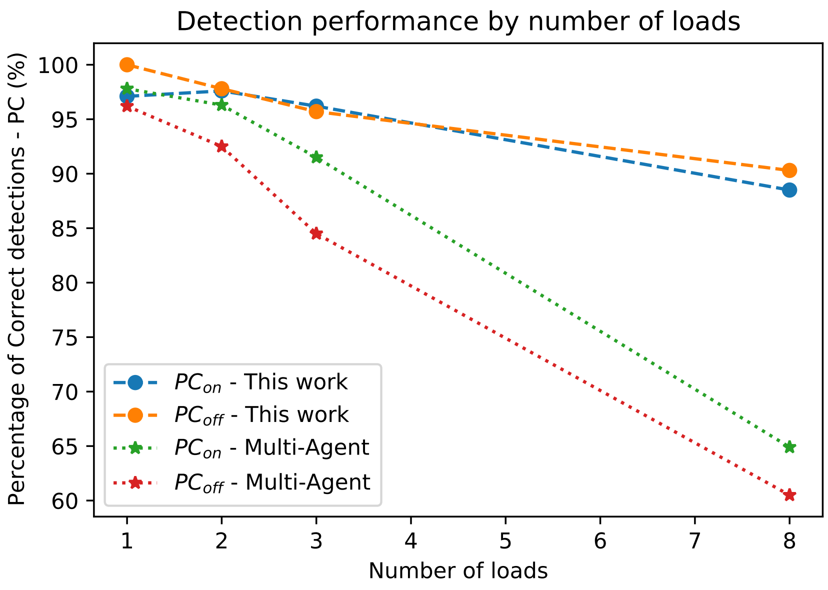
Source: Own work.

However, for two-load acquisitions, one can observe a more significant performance degradation in the correct detection of power-off events for the Multi-Agent method. At the same time, the architecture proposed in this study remains more stable. When increasing to three loads, the performance degradation for the Multi-Agent method intensifies, reaching for the first time a PC_{off} below 90%, which results in a 5% reduction for PC_{on} when compared to the performance with two loads. Meanwhile, the proposed architecture maintains performance close to that achieved with only two loads, outperforming Multi-Agent by more than 10% for PC_{off}.

For the LIT-SYN-8 acquisitions, on the other hand, the model proposed in this work obtains the most relevance. Despite presenting some performance decrease for this group of waveforms, the proposed architecture remains comparatively stable, reaching a percentage of correct detections close to 90% for both turn-on and turn-off. The Multi-Agent architecture, on the other hand, shows a pronounced decline for this group of loads. The result is a superiority of approximately 25% for PC_{on} and 30% for PC_{off} in favor of the system proposed in this study.

A graphical representation of this comparison is shown in Figure 26. In this figure, the performance of both models is plotted. For the case of one load, it is noticeable that both models perform equally well. However, for two loads, it is already possible to notice the model proposed in this work, represented by the blue and yellow lines, standing out. With three loads, the performance of the Multi-Agent model, represented by the red and green lines, shows a significant degradation, which intensifies when moving on to the set of acquisitions with eight loads.

Figure 26 – Graphical comparison of the detection performance of the methods in Table 7.



Source: Own work.

Finally, this comparison demonstrates that the system proposed in this work is more stable, allowing a superior performance even with an increase in the number of loads connected to the grid. Furthermore, for the case of turn-on events, the detection error represented by D_{on} proved to be lower for most scenarios. Although higher, the D_{off} obtained for the turn-off events was still close to that obtained with the Multi-agent system. This results in more correct and higher-quality detections, getting closer to where the event occurred.

5.5.2 Classification

Table 8 summarizes the comparison with other works that use the LIT-Dataset. The first notable aspect of this comparison is that the other works do not address the problem of identifying power-off events, showing a contribution made by the method proposed in this research.

Regarding the power-on events, using the macro averaged accuracy (ACC) as a measure, the result obtained by the proposed architecture is close to the works with the best performance in the state-of-the-art for this dataset, represented by the Multi-Agent method (LAZZARETTI *et al.*, 2020), the 2-D Fourier (MULINARI *et al.*, 2022) and the Scattering Transforms (AGUIAR *et al.*, 2021). Furthermore, the method proposed in this work outperforms other strategies in the literature, such as the Concatenated CNNs (WU; WANG, 2019) and the GAN System (MUKAROH *et al.*, 2020). Using the F_1 Macro as a measure, the performance of the proposed system remains close to the one presented by the Scattering Transform and the 2-D Fourier Transform, the only works that calculate such performance measure, showing a difference of less than 3% for both strategies.

Table 8 – Comparison of classification results between works that use the LIT-Dataset.

Method	Scenario	ACC (%)	F_1 Macro (%)
This work	On	97.9	94.9
	Off	98.7	96.6
Multi-Agent (LAZZARETTI *et al.*, 2020)	On	99.8	-
	Off	-	-
Scattering Transforms (AGUIAR *et al.*, 2021)	On	99.9	97.4
	Off	-	-
Concatenated CNNs (WU; WANG, 2019)	On	69.3	-
	Off	-	-
GAN System (MUKAROH *et al.*, 2020)	On	92.0	-
	Off	-	-
Fourier 2-D (MULINARI *et al.*, 2022)	On	96.8	96.2
	Off	-	-
V-I Trajectories (WANG *et al.*, 2018)	On	93.7	-
	Off	-	-

Source: Own work.

This comparison shows that the method proposed in this work can maintain load classification levels close to that observed in the state-of-the-art for this task while bringing other advantages, such as the capability of event detection and the ability to identify loads for switch-off events – features neglected by the other approaches. It is worth noting, however, that the LIT-Dataset is still a relatively recent dataset and therefore has a limited number of papers in the literature that have made use of it, making comparisons challenging.

Furthermore, some of these other works use the event markings provided by the database to test their classifiers. The result is that there is no analysis of the impact that poor quality detection, with a significant error compared to the correct event sample, can have on the classifiers in question. The proposal presented in this study, on the other hand, performs the classification with events that occur at several points in the analysis window, which can make this a more complicated task, as in the case of a power-on event very close to the end of the window (signal clipping) or a power-off event very close to the beginning. Both scenarios can result in a small number of samples with the load connected, making it difficult for the system to identify them. Thus, it enables the evaluation of the system in a scenario closer to reality.

Another issue is that all the works presented in Table 8 do not use the multi-label classification concept, meaning they identify only one device at a time. For this reason, Table 9 compares the proposed method and other methods in the literature that use the multi-label concept. However, it is worth noting that the methods presented in this table are not directly comparable because they use different databases. Nevertheless, this strategy, which can also be observed in other works in the literature (FAUSTINE; PEREIRA, 2020; LIU *et al.*, 2018), gives an idea of the order of magnitude of the results.

Table 9 – Comparison of classification results considering only multi-label models.

Method	Dataset	Results
This work	LIT	97.1% (F_1 Macro)
Faustine and Pereira (2020)	PLAID	94.0% (F_1 score)
Tabatabaei *et al.* (2017)	REDD-House1	61.9% (F_1 Macro)
Yang *et al.* (2020)	UK-DALE-house1	93.8% (F_1 score)
Nalmpantis and Vrakas (2019)	UK-DALE-house1	92.5% (F_1 score)
Manca *et al.* (2022)	PLAID	91.5% (F_1 Macro)

Source: Own work.

Considering only the reported measurements within this context, one can notice that the proposed method achieves comparable results to the other works presented, obtaining even superior results. That suggests its potential for NILM applications, reinforcing the good performance seen in the earlier comparisons. Finally, it is also worth mentioning that, due to the nature of the proposed architecture, the PLAID, REDD, and UK-DALE databases, used by the other works, could not be employed due to characteristics of these datasets, such as the fact that the load aggregations are created from a simple linear combination of acquisitions of a single load and the lack of precise marking for switching events.

5.6 NOISE

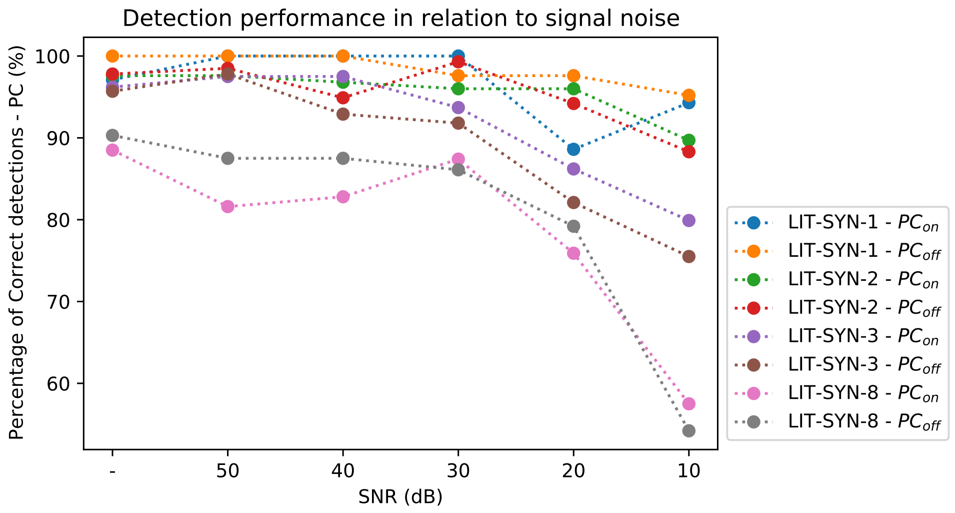
Regarding the noise robustness tests, the following subsections present the results obtained for both the detection and the load classification tasks when applying the proposed methodology to signals containing Additive White Gaussian Noise (AWGN).

5.6.1 Detection

Table 10 summarizes the detection results for the noise levels applied to the signal. For each of these scenarios, the model was retrained and evaluated using the signals modified by the noise level for that scenario. To facilitate the visualization and interpretation of these data, Figure 27 plots the evolution of the detection performance as the signal-to-noise ratio decreases.

In this figure, the PC_{on} and PC_{off} measures for each of the LIT-SYN subsets are arranged graphically, allowing the visualization of the effect of increasing the amount of noise contained in the signal for each measure. The observed result is a decrease in performance, at least in some groups, for the higher noise levels, which was to be expected given the significant decrease in signal quality.

Figure 27 – Evolution of detection performance as the signal quality degrades. The farther to the right on the x-axis, the higher the noise level relative to the signal. Regarding notation, "-" indicates the signal without adding noise.



Source: Own work.

However, a closer analysis reveals some other points about the behavior of the architecture in noisy scenarios. The first point is the performance stability for the first scenarios, such as the 50, 40, and 30 dB SNR cases, where there is already some noise. Within these three cases, i.e., up to an SNR of 30 dB, the detection performance remained high, with $PC_{\text{on}} > 80\%$ and $PC_{\text{off}} > 80\%$ for all groups in all scenarios. As shown earlier in Figure 2, up to this range of signal-to-noise ratio, the signal still retains its shape visually, justifying the good performance of the architecture.

On the other hand, for the 20 dB SNR scenario, a degradation in performance can be observed for acquisitions with a larger number of loads, represented by the LIT-SYN-3 and

Table 10 – Comparison of proposed architecture's event detection performance in noisy scenarios. Regarding the notation, "-" stands for the original signal, i.e., without added noise.

Subset	Signal-to-Noise Ratio (dB)	PC_{on} (%)	PC_{off} (%)	D_{on}	D_{off}
LIT-SYN-1	-	97.1	100.0	0.9	0.7
	50	100.0	100.0	1.0	0.5
	40	100.0	100.0	1.0	0.6
	30	100.0	97.6	1.2	0.4
	20	88.6	97.6	1.1	1.1
	10	94.3	95.2	1.2	1.1
LIT-SYN-2	-	97.6	97.8	0.6	0.6
	50	97.6	98.5	0.7	0.6
	40	96.8	94.9	0.8	0.6
	30	96.0	99.3	0.8	0.7
	20	96.0	94.2	1.0	0.8
	10	89.7	88.3	0.9	0.8
LIT-SYN-3	-	96.2	95.7	0.9	0.7
	50	97.5	97.8	0.9	0.9
	40	97.5	92.9	1.0	0.9
	30	93.7	91.8	1.0	1.1
	20	86.2	82.1	1.1	1.2
	10	79.9	75.5	1.2	1.4
LIT-SYN-8	-	88.5	90.3	1.0	1.0
	50	81.6	87.5	1.0	1.3
	40	82.8	87.5	1.0	1.0
	30	87.4	86.1	1.3	1.5
	20	75.9	79.2	1.1	1.5
	10	57.5	54.2	1.4	1.5
LIT-SYN	-	95.1	95.9	0.8	0.7
	50	94.3	96.6	0.9	0.8
	40	94.3	93.3	0.9	0.8
	30	93.6	93.8	1.0	0.9
	20	87.2	86.9	1.1	1.1
	10	79.4	77.9	1.1	1.2

Source: Own work.

LIT-SYN-8 sets. Meanwhile, the system performance remained stable for up to 2 loads, except for PC_{on} for the LIT-SYN-1 set. However, this variation may be related to how the final model is selected, focusing only on the classification performance within the validation set, creating the possibility of these slight variations in the detection performance.

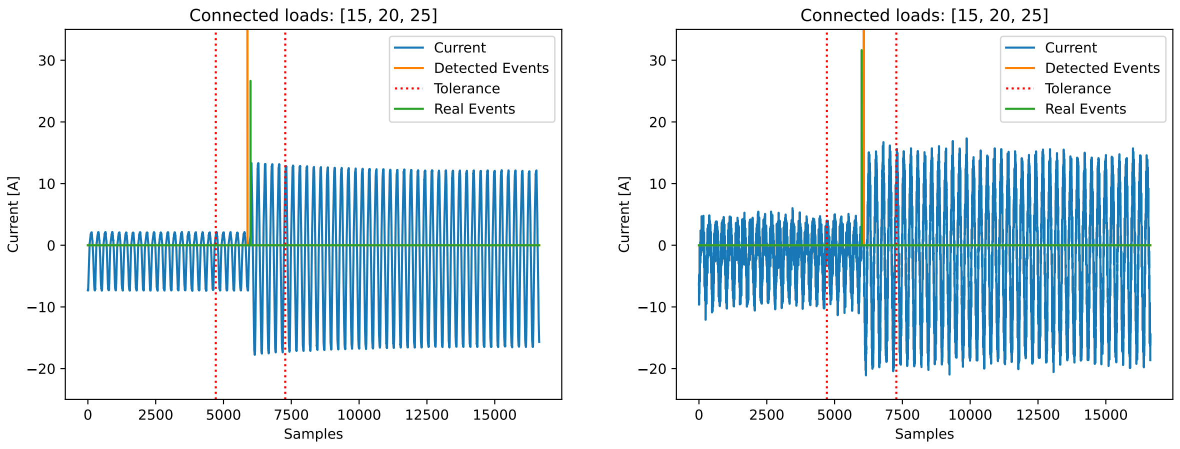
Finally, the most pronounced decrease occurs at 10 dB SNR, especially for the LIT-SYN-8 acquisitions. For this degraded signal, a detection performance below 70% can be observed for the first time. The triple waveforms of the LIT-SYN-3 also showed a decrease in performance but to a lesser extent. On the other hand, for the waveforms with up to two loads (LIT-SYN-1 and LIT-SYN-2), although there is a slight decrease, the values remain at a high level of performance, exceeding the 90% range. Considering the amount of noise present with an SNR of 10 dB, which can be visually observed in the example in Figure 14, these results become even more significant, showing the ability of the proposed architecture to perform well even under such adverse conditions.

Regarding the detection error D, it is possible to observe an increase as the signal quality degrades, which is to expect given the model performance degradation presented by the PC_{on} and PC_{off} measurements. However, even with this increase, an average error greater than 1.5 half-cycles of the power network was not observed. A possible reason for this phenomenon is that noise tends to affect detection primarily for lower power loads. Thus, for higher power loads, the switching event remains separated from the background signal, resulting in good quality detections with low error, keeping the error low even at higher noise levels.

An example of this scenario is shown in Figures 28 and 29. Figure 28 compares two signal snippets, the one on the left with no added noise and the one on the right with an SNR of 10 dB. There, one can see that the switching event still stands out in the signal. That allows the model to detect the event very close to the correct position in both signal snippets. However, this event stands out from the background signal because the load responsible for the switching event is load 25, the Parlux Hair Dryer at speed 2, which has a power consumption of 660W, one of the highest in the dataset.

Figure 28 – Comparison of switching a higher power load in scenarios with and without noise addition (10 **dB SNR). These signal excerpts include the following devices: Drill (15), Eleganza Hair Dryer (20), and Parlux Hair Dryer (25).**

(a) No noise added. **(b)** 10 **dB SNR.**

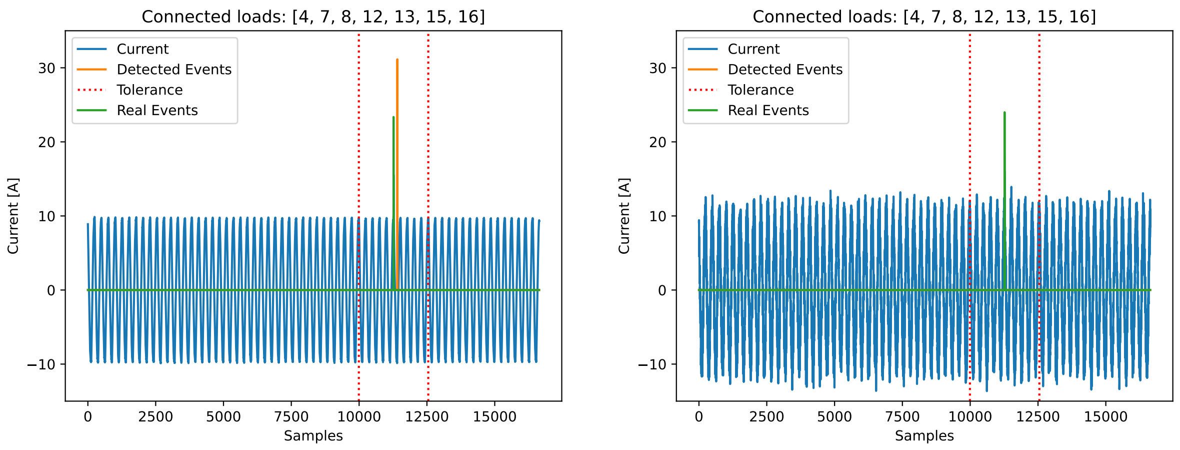


Source: Own work.

On the other hand, adding noise can cause the system to fail to detect a switching event for lower power loads. Figure 29 compares two signal snippets from the same section, one with no added noise and the other with an SNR of 10 dB. The Motorola cell phone charger, which has a power of 50W, is responsible for the switching event. For the waveform on the left, where no noise has been added, the model can identify the event, even though it is not easy to identify it visually. However, when noise is added, as shown in the waveform on the right, even the model cannot detect it.

Figure 29 – Comparison of switching a lower power load in scenarios without noise addition (SNR of 10 dB) and with noise addition. These signal sections include the devices: Fume Extractor (4), Soldering Station (7), Motorola Cell Phone Charger (8), Vaio Laptop (12), Incandescent Lamp (13), Drill (15), and Oil Heater (16).

(a) No noise added. **(b) 10 dB SNR.**



Source: Own work.

These tests showed that the proposed architecture maintains stable performance levels with an input signal up to 30 dB SNR. For lower SNR levels, the system can still maintain performance for aggregated signals with a smaller amount of loads. However, the performance degrades for scenarios with a larger amount of connected loads.

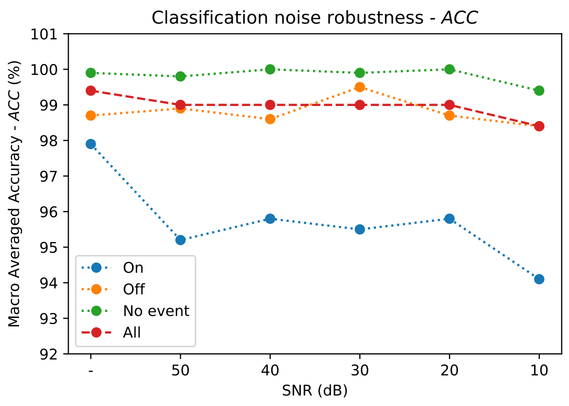
5.6.2 Classification

For the problem of identifying the loads that compose the aggregate current signal measured, Table 11 summarizes the results obtained in the noise robustness tests. To facilitate the interpretation and visualization of this data, Figure 30 presents this information visually.

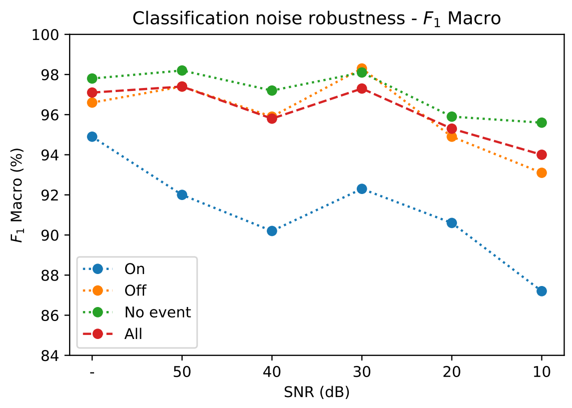
In this figure, one can see that the system does not present significant variations in performance when evaluated by the macro averaged accuracy (ACC), except for the power-on scenario. For this subset of examples, the performance shows a decrease of about 3% already in the first noise scenario. It remains stable until it drops again in the scenario with the highest signal degradation. Nevertheless, the accuracy remains high even with this reduction, with $ACC > 94\%$ for all noise levels. For the other scenarios, however, the performance variation is minimal, resulting in a consistently high performance, with $ACC > 98\%$ for all noise levels tested.

Figure 30 – Evolution of classification performance measures as signal degradation increases. The farther to the right on the x-axis, the higher the noise level relative to the signal.

(a) Macro Averaged Accuracy - ACC.



(b) F_1 Macro.



Source: Own work.

For the F_1 Macro measure, the observed behavior is similar to that seen for accuracy. The power-on scenario stands out negatively, showing a drop in performance already at the first noise level. It then remains stable between 40 dB and 20 dB SNR until it drops back again for 10 dB SNR, where the performance drops below 90% for the first time. For the other scenarios, it is also possible to observe a drop in performance starting at 20 dB SNR, which eventually intensifies when the SNR is reduced to 10 dB. Nevertheless, even at the highest noise level, the performance for these scenarios remains high, with F_1 Macro $> 94\%$.

For the 20 and 10 dB SNR cases, the power-on scenario showed the most significant drop in both performance measures. Furthermore, all event-type scenarios showed performance degradation when evaluated with the F_1 Macro for the same noise levels. Coincidentally, these noise levels also caused the most significant drop in the event detection model performance. That may indicate an optimal range of overall model performance stability up to 30 dB SNR – potentially decreasing as the signal-to-noise ratio decreases.

Despite these drops, the classification performance remains high even at the highest noise levels, with $ACC > 94\%$ and F_1 Macro $> 87\%$ for all scenarios. Examples of this good performance are shown in Figures 28 and 29, where detection errors are shown. The model correctly identified all loads connected to the grid for the signal clippings shown in both figures, including those in the noisiest proposed scenario (SNR of 10 dB). The main highlight is the one shown in Figure 29. Not only was the event coming from a lower power load, making it challenging to identify the event visually, but the signal also had a larger number of simultaneously connected loads.

Table 11 – Comparison of the performance of the proposed architecture for load identification in noisy scenarios.

Scenario	Signal-to-Noise Ratio (dB)	ACC (%)	F_1 Macro (%)
On	-	97.9	94.9
	50	95.2	92.0
	40	95.8	90.2
	30	95.5	92.3
	20	95.8	90.6
	10	94.1	87.2
Off	-	98.7	96.6
	50	98.9	97.4
	40	98.6	95.9
	30	99.5	98.3
	20	98.7	94.9
	10	98.4	93.1
No event	-	99.9	97.8
	50	99.8	98.2
	40	100.0	97.2
	30	99.9	98.1
	20	100.0	95.9
	10	99.4	95.6
Total	-	99.4	97.1
	50	99.0	97.4
	40	99.0	95.8
	30	99.0	97.3
	20	99.0	95.3
	10	98.4	94.0

Source: Own work.

5.7 EMBEDDED SYSTEM

This Section presents the results of the tests on the NVIDIA Jetson TX1 platform. For this purpose, inference tests were performed on the board system using the previously trained models, monitoring information such as execution time and system resource consumption. Approximately 4 000 examples were used to obtain the average execution time for each signal slice. The results of these tests are available in Tables 12 and 13.

Table 12 – Comparison of results on an embedded system. Regarding the notation, "Y" means that the paper implements this aspect, while "-" means that the paper does not consider this aspect.

Method	Detection	Classification	Window Size (ms)	Computational Time (ms)
This work	Y	Y	833	11.2
Baptista *et al.* (2018)	-	Y	333	5.7

Source: Own work.

As can be seen, the results are promising. One of the main positive points is that the average execution time is significantly lower than the window size. That means that its practical application is feasible since it takes less time to process a set of samples than to collect them, preventing the system from overloading and starting to accumulate unprocessed samples.

In addition, the significant difference between the processing window size and the execution time allows strategies that include window movements smaller than its size, allowing the same samples to be analyzed multiple times, resulting in more reliable predictions. A technique that follows this principle was presented in Section 5.4 and used to generate visual examples of complete acquisitions. In this case, with each new inference, the window was shifted by 10 power grid cycles, corresponding to the size of one grid position to the model's input. Thus, each of these 10 cycles is analyzed five times, once at each grid position, generating multiple predictions for the same signal segment.

Regarding the comparison with related works, Table 12 presents the results of a method proposed in the work by Baptista *et al.* (2018). It implements a model using 2D convolutional neural networks embedded in an FPGA. Unfortunately, the different platforms make direct comparison impossible. However, disregarding this issue, it can be seen that the method proposed in this work achieves a comparable performance, even superior, if the average time is considered together with the window size, analyzing the average time for processing 1 second of the signal.

In terms of resource consumption, Table 13 presents the results observed during the tests. As can be seen, the most used resource during these tests was RAM. This is due to the fact that during the tests, all signal clipping instances used to evaluate the model are loaded into memory at the same time, which would not happen in a practical implementation scenario. On the other hand, the CPU and GPU usage was far from the maximum capacity of the platform. That indicates that the proposed model could be implemented on lower-cost devices, reducing the price of the final system and making it easier to implement in practice. Overall, the use of 1D convolutional layers resulted in a small architecture that, in turn, generates a model with low hardware requirements while maintaining consistent performance on NILM tasks.

Table 13 – Resource consumption on the embedded system.

Resource monitored	System state	Value
Maximum CPU temperature	Idle	32.5^oC
	Running	39.5^oC
Maximum GPU temperature	Idle	29.0^oC
	Running	35.5^oC
RAM	Idle	$808/3984$ Mb (20.3%)
	Running	$3777/3984$ Mb (94.8%)
Average CPU load	Idle	8.2%
	Running	30.9%
Average GPU load	Idle	0.0%
	Running	50.6%

Source: Own work.

5.8 SCATTERING TRANSFORM – ST

The following subsections present the performance of the ST-based model and compare it to the CNN-based method also proposed in this work. To evaluate the effectiveness of the ST-based model when dealing with small amounts of data, the performance was measured with and without the data augmentation strategy proposed in Subsection 4.3.1.2. It is important to emphasize that the ST-based model is still a work in progress, and thus the results presented here are still preliminary.

5.8.1 Detection Performance

Table 14 compares detection results for both architectures. The best performance for each scenario (with and without data augmentation) is highlighted in bold for better visualization. One can observe that the ST-based model was able to achieve a good performance, reaching PC_{on} and PC_{off} above 80% for all subsets in every scenario. However, despite that, it was not enough to outperform the CNN-based model when using data augmentation. Nonetheless, for the scenario without data augmentation, the ST-based model shows an outstanding performance for turn-on events when there is a larger number of simultaneous devices, outperforming the CNN-based model by almost 20%.

Table 14 – Detection performance comparison between ST-based and CNN-based architectures.

Data Augmentation	Method	Subset	PC_{on} (%)	PC_{off} (%)
Yes	ST-based	LIT-SYN-1	88.6	**100.0**
		LIT-SYN-2	92.1	90.5
		LIT-SYN-3	88.7	94.0
		LIT-SYN-8	80.5	73.6
	CNN-based	LIT-SYN-1	**97.1**	**100.0**
		LIT-SYN-2	**97.6**	**97.8**
		LIT-SYN-3	**96.2**	**95.7**
		LIT-SYN-8	**88.5**	**90.3**
No	ST-based	LIT-SYN-1	**100.0**	**100.0**
		LIT-SYN-2	91.1	93.5
		LIT-SYN-3	**98.7**	94.3
		LIT-SYN-8	**96.5**	77.5
	CNN-based	LIT-SYN-1	**100.0**	**100.0**
		LIT-SYN-2	**97.8**	**98.3**
		LIT-SYN-3	92.6	**98.1**
		LIT-SYN-8	88.4	**94.4**

Source: Own work.

Furthermore, an interesting phenomenon observed in these tests is that the ST-based model performs worse when data augmentation is used. A possible explanation lies in how the data augmentation is performed. As presented in Subsection 4.3.1.2, the event-free signal snippets are selected using the HAND algorithm in an attempt to find regions of the signal with enough amplitude variation, i.e., signal sections that may look like an event but they are not. That creates a much more challenging scenario for the model, resulting in a weaker performance.

5.8.2 Classification Performance

For the classification task, however, the ST-based model shows promising results. Table 15 shows the classification performance comparison for both architectures presented in this work. As can be seen, the ST-based model outperforms the CNN-based model with and without data augmentation. For the scenario without data augmentation, the ST model is able to maintain its performance level while outperforming the CNN-based model by more than 15%.

Table 15 – Classification performance comparison between ST-based and CNN-based architectures.

Data Augmentation	Method	F_1 Macro (%)
Yes	ST-based	**97.7**
	CNN-based	97.1
No	ST-based	**97.5**
	CNN-based	81.0

Source: Own work.

These results show the potential of ST to perform well even under adverse conditions, such as scenarios with smaller datasets. For example, the model was able to achieve similar performance when trained with 8 416 instances (without data augmentation) and 20 812 (with data augmentation). It is also worth noting that the ST-based model was trained without the weights proposed in Subsection 4.3.2.

6 CONCLUSIONS AND FUTURE WORKS

In order to reduce energy waste, consumption monitoring techniques such as NILM play an essential role, allowing users to identify and reduce potential waste. One of the challenges of NILM techniques to perform this monitoring task is to detect load-switching events in the electrical grid and to be able to identify the devices that are connected at that moment.

However, the general structure of this technique presented in Section 2.1, with an event detection step followed by a load disaggregation of the background signal, moving to feature extraction, and ending with a load classification, has some limitations. The simplest disaggregation methods use a signal difference that can deform nonlinear load combinations, extraction methods can be time consuming, and detection methods show decreasing performance as the number of simultaneous devices increases. For these reasons, this work presents an architecture capable of simultaneously performing the tasks of detection, feature extraction, and classification.

Thus, all of the objectives listed in Section 1.2 were achieved through the steps listed in this document. The techniques and concepts presented in Chapter 2 have been studied in order to elaborate the final architecture. The state of the art of NILM solutions was also studied, focusing mainly on deep learning techniques, as presented in Chapter 3. Finally, a model capable of event detection, feature extraction, and classification in NILM signals was developed using the methodology presented in Chapter 4.

The results obtained for the proposed solution architecture, presented in Chapter 5, show that it outperforms other state-of-the-art methods using the same dataset in the detection task, achieving a general percentage of correct detections for the switch-on events (PC_{on}) of 95.1% and for the switch-off events (PC_{off}) a general percentage of 95.9%. As for the classification, the obtained performance was comparable to other methods in the literature, reaching an F1 Macro of 97.1%. Furthermore, in the noise robustness tests with AWGN, it was possible to observe that the system remains stable up to scenarios with an SNR of 30 dB, where the system was able to maintain a PC_{on} equal to 93.6%, a PC_{off} equal to 93.8% and an F1 macro of 97.3%. This indicates that the system should perform well even with a lower-quality capture system where the signal may contain more noise, thus reducing the cost of a potential end device.

Finally, the results of the tests on the NVIDIA Jetson TX1 platform, detailed in Subsection 5.7, demonstrated the feasibility of the proposed architecture in an embedded system, allowing its implementation in a real-world scenario. These tests also showed that the performance achieved by the proposed model is comparable to other methods in the literature that have also been embedded. In addition, the tests showed that the requirements for running the model presented in this work are below the platform's capabilities. This indicates that the system could run on a more basic platform at a lower cost, facilitating the creation and deployment of a final device using the architecture proposed in this work. The codes used in this research are publicly available[1] for testing and replication.

Despite the promising results, the proposed model requires a large amount of data, which is typical for deep learning techniques. That represented one of the major difficulties during the elaboration of this work. Although examples without load-switching events are helpful, the initial division of the signal into positions on a grid would allow the model to be trained without using them. That is because one example would have a grid position with events, while the other parts would serve as an example of an event-free signal. This idea of training without negative examples was tested with a data augmentation tactic that consisted of generating multiple clippings for the same event, changing the window's position over the annotated event in the database, and adding different levels of AWGN noise. This strategy yielded good results. However, it was discarded since data augmentation is not widely used for the NILM scenario, which could raise questions about its merit. That led to the solution presented in this work, where event-free segments of interest are selected based on the amplitude variation of the signal.

Another point observed during the development of this work, which can be improved in future works, is the need to use an output to indicate the sample where an event occurred. The event type output can already indicate a grid position where the event occurred. Meanwhile, the detection output is responsible for indicating in which sample the event occurred within that grid position. Although the detection output gives the model a better resolution in indicating where an event occurred, the resolution provided by the event type output may already be sufficient. That is because it can indicate that an event occurs at an interval of 10 electrical grid cycles (60 Hz), which is about 167 ms. For consumers, this is a small interval in terms of consumption detail and, therefore, could be used.

These concepts have been tested on some level in the ST-based model experiments. As presented in Section 5.8, the ST-based model provides an alternative to perform the detection and load identification tasks with a smaller amount of data. Despite the lack of an event sample

[1] Codes are available at https://github.com/LucasNolasco/DeepDFML-NILM and at https://github.com/LucasNolasco/ST-NILM.

regression output, this model is able to correctly detect more than 80% of the events. The highlight, however, is the classification task. Even when trained without data augmentation, with only 8 416 examples, the model is able to achieve a performance comparable to that of the CNN-based model with data augmentation (20 812 examples). Considering only the scenario without data augmentation, it outperforms the CNN-based model by more than 15%. This shows the potential of such a strategy, although the performance of the detection task still needs improvement.

6.1 FUTURE WORK

As a way to take this work forward, the following ideas are proposed for consideration:

- Perform tests in scenarios where there are simultaneous events within the same signal clipping to verify the behavior of the proposed system;
- With a database containing a larger number of examples, perform tests that remove the clippings without load-switching events;
- Investigate the simultaneity factor to learn how many simultaneous appliances are generally used by an average consumer. With that information, perform tests with a larger number of simultaneously connected loads to determine the maximum number of connected devices the model can identify at the same time;
- Investigate different feature selection strategies to improve the detection performance of ST-based models;
- Analyze the model performances in low-frequency scenarios by sub-sampling the signal. This test aims to understand the feasibility of implementing this model in a final device using cheaper low-frequency acquisition systems;
- Analyze the trained models using class activation maps to understand the contribution of each task in the multi-task learning strategy proposed;
- Perform tests combining the Scattering Transforms with CNN layers. This aims to investigate if the spatial characteristics from CNNs combined with the time-frequency signal decomposition from the Scattering Transforms can improve the model's performance.

REFERENCES

AGUIAR, Everton Luiz de; NOLASCO, Lucas da Silva; LAZZARETTI, André Eugenio; PIPA, Daniel Rodrigues; LOPES, Heitor Silvério. St-NILM: A wavelet scattering-based architecture for feature extraction and multi-label classification in NILM signals. Submitted to the IEEE Sensors Journal. 2023.

AGUIAR, Everton Luiz de; LAZZARETTI, André Eugenio; MULINARI, Bruna Machado; PIPA, Daniel Rodrigues. Scattering Transform for Classification in Non-Intrusive Load Monitoring. **Energies**, v. 14, n. 20, p. 6796, 2021. ISSN 19961073.

ARMEL, K. Carrie; GUPTA, Abhay; SHRIMALI, Gireesh; ALBERT, Adrian. Is disaggregation the holy grail of energy efficiency? the case of electricity. **Energy Policy**, v. 52, p. 213–234, 2013. ISSN 0301-4215. Special Section: Transition Pathways to a Low Carbon Economy.

BAETS, Leen De; DEVELDER, Chris; DHAENE, Tom; DESCHRIJVER, Dirk. Detection of unidentified appliances in non-intrusive load monitoring using siamese neural networks. **International Journal of Electrical Power and Energy Systems**, Elsevier Ltd, v. 104, p. 645–653, 1 2019. ISSN 01420615.

BAETS, Leen De; RUYSSINCK, Joeri; DEVELDER, Chris; DHAENE, Tom; DESCHRIJVER, Dirk. Appliance classification using vi trajectories and convolutional neural networks. **Energy and Buildings**, Elsevier Ltd, v. 158, p. 32–36, 1 2018. ISSN 03787788.

BAPTISTA, Darío; MOSTAFA, Sheikh Shanawaz; PEREIRA, Lucas; SOUSA, Leonel; MORGADO-DIAS, Fernando. Implementation strategy of convolution neural networks on field programmable gate arrays for appliance classification using the voltage and current (V-I) trajectory. **Energies**, MDPI AG, v. 11, 9 2018. ISSN 19961073.

BISHOP, Christopher M. **Pattern Recognition and Machine Learning (Information Science and Statistics)**. Berlin, Heidelberg: Springer-Verlag, 2006. ISBN 0387310738.

BRUNA, Joan; MALLAT, Stephane. Invariant scattering convolution networks. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 35, n. 8, p. 1872–1886, 2013.

CARUANA, Richard A. Multitask Learning: A Knowledge-Based Source of Inductive Bias. **Machine Learning Proceedings**, p. 41–48, 1993.

CHEN, Junfeng; WANG, Xue; ZHANG, Xiaotian; ZHANG, Weihang. Temporal and spectral feature learning with two-stream convolutional neural networks for appliance recognition in NILM. **IEEE Transactions on Smart Grid**, v. 13, n. 1, p. 762–772, 2022.

CUI, Yin; JIA, Menglin; LIN, Tsung Yi; SONG, Yang; BELONGIE, Serge. Class-balanced loss based on effective number of samples. **Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition**, v. 2019-June, p. 9260–9269, 2019. ISSN 10636919.

D'INCECCO, Michele; SQUARTINI, Stefano; ZHONG, Mingjun. Transfer learning for non-intrusive load monitoring. **IEEE Transactions on Smart Grid**, Institute of Electrical and Electronics Engineers Inc., v. 11, p. 1419–1429, 3 2020. ISSN 19493061.

EHRHARDT-MARTINEZ, K; DONNELLY, K A; LAITNER, J A. Advanced metering initiatives and residential feedback programs: A meta-review for household electricity-saving opportunities. 2010.

FAUSTINE, Anthony; PEREIRA, Lucas. Multi-label learning for appliance recognition in NILM using fryze-current decomposition and convolutional neural network. **Energies**, v. 13, n. 6, 2020. ISSN 19961073.

FAWCETT, Tom. An introduction to ROC analysis. **Pattern Recognition Letters**, v. 27, n. 8, p. 861–874, 2006. ISSN 01678655.

FORMAN, George; SCHOLZ, Martin. Apples-to-Apples in Cross-Validation Studies: Pitfalls in Classifier Performance Measurement. **ACM SIGKDD Explorations Newsletter**, v. 12, n. 1, p. 49–57, 2010. ISSN 19310145.

GOMES, Eduardo; PEREIRA, Lucas. PB-NILM: Pinball guided deep non-intrusive load monitoring. **IEEE Access**, Institute of Electrical and Electronics Engineers Inc., v. 8, p. 48386–48398, 2020. ISSN 21693536.

GOODFELLOW, Ian; BENGIO, Yoshua; COURVILLE, Aaron. **Deep Learning**. *[S.l.]*: MIT Press, 2016. http://www.deeplearningbook.org.

GUO, Luyang; WANG, Shouxiang; CHEN, Haiwen; SHI, Qingyuan. A load identification method based on active deep learning and discrete wavelet transform. **IEEE Access**, Institute of Electrical and Electronics Engineers Inc., v. 8, p. 113932–113942, 2020. ISSN 21693536.

HART, George W. Nonintrusive Appliance Load Monitoring. **Proceedings of the IEEE**, v. 80, n. 12, p. 1870–1891, 1992. ISSN 15582256.

IEA, International Energy Agency. **World Energy Outlook 2019**. *[s.n.]*, 2019. 810 p. Available at: https://www.oecd-ilibrary.org/content/publication/caf32f3b-en.

INCE, Turker; KIRANYAZ, Serkan; EREN, Levent; ASKAR, Murat; GABBOUJ, Moncef. Real-time motor fault detection by 1-d convolutional neural networks. **IEEE Transactions on Industrial Electronics**, v. 63, n. 11, p. 7067–7075, 2016.

IQBAL, Hafiz Khurram; MALIK, Farhan Hassan; MUHAMMAD, Aoun; QURESHI, Muhammad Ali; ABBASI, Muhammad Nawaz; CHISHTI, Abdul Rehman. A critical review of state-of-the-art non-intrusive load monitoring datasets. **Electric Power Systems Research**, v. 192, p. 106921, 2021. ISSN 0378-7796.

JIA, Ye; JOHNSON, Melvin; MACHEREY, Wolfgang; WEISS, Ron J.; CAO, Yuan; CHIU, Chung-Cheng; ARI, Naveen; LAURENZO, Stella; WU, Yonghui. Leveraging weakly supervised data to improve end-to-end speech-to-text translation. *In*: **ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)**. *[S.l.: s.n.]*, 2019. p. 7180–7184.

JIA, Ziyue; YANG, Linfeng; ZHANG, Zhenrong; LIU, Hui; KONG, Fannie. Sequence to point learning based on bidirectional dilated residual network for non-intrusive load monitoring. **International Journal of Electrical Power & Energy Systems**, v. 129, p. 106837, 5 2020. ISSN 0142-0615.

KELLY, Jack; KNOTTENBELT, William. Neural NILM: Deep neural networks applied to energy disaggregation. **BuildSys 2015 - Proceedings of the 2nd ACM International Conference on Embedded Systems for Energy-Efficient Built**, p. 55–64, 2015.

KHAN, Muhammad Raza; ZIYADI, Morteza; ABDELHADY, Mohamed. **MT-BioNER: Multi-task Learning for Biomedical Named Entity Recognition using Deep Bidirectional Transformers**. 2020.

KING, Gary; ZENG, Langche. Logistic regression in rare events data. **Journal of Statistical Software**, v. 8, p. 137–163, 2001. ISSN 15487660.

KRIZHEVSKY, Alex; SUTSKEVER, Ilya; HINTON, Geoffrey E. Imagenet classification with deep convolutional neural networks. *In*: PEREIRA, F.; BURGES, C. J. C.; BOTTOU, L.; WEINBERGER, K. Q. (Ed.). **Advances in Neural Information Processing Systems**. *[S.l.]*: Curran Associates, Inc., 2012. v. 25.

KUO, Hui-Hsiung. **White Noise Distribution Theory**. *[S.l.]*: CRC Press, 1996. ISBN 9780203733813.

LAZZARETTI, André Eugenio; RENAUX, Douglas Paulo Bertrand; LIMA, Carlos Raimundo Erig; MULINARI, Bruna Machado; ANCELMO, Hellen Cristina; OROSKI, Elder; PÖTTKER, Fabiana; LINHARES, Robson Ribeiro; NOLASCO, Lucas da Silva; LIMA, Lucas Tokarski; OMORI, Júlio Shigeaki; SANTOS, Rodrigo Braun dos. A multi-agent NILM architecture for event detection and load classification. **Energies**, v. 13, n. 17, p. 1–37, 2020. ISSN 19961073.

Le Cun, Y.; GUYON, I.; JACKEL, L. D.; HENDERSON, D.; BOSER, B.; HOWARD, R. E.; DENKER, J. S.; HUBBARD, W.; GRAF, H. P. Handwritten Digit Recognition: Applications of Neural Network Chips and Automatic Learning. **IEEE Communications Magazine**, v. 27, n. 11, p. 41–46, 1989. ISSN 01636804.

LI, Keqin; FENG, Jian; ZHANG, Juan; XIAO, Qi. Adaptive fusion feature transfer learning method for NILM. **IEEE Transactions on Instrumentation and Measurement**, p. 1–1, 2023.

LIU, Yanchi; WANG, Xue; YOU, Wei. Non-intrusive load monitoring by voltage-current trajectory enabled transfer learning. **IEEE Transactions on Smart Grid**, Institute of Electrical and Electronics Engineers Inc., 2018. ISSN 19493053.

LUO, Qingquan; YU, Tao; LAN, Chaofan; HUANG, Yi; WANG, Zihao; PAN, Zhenning. A generalizable method for practical non-intrusive load monitoring via metric-based meta-learning. **IEEE Transactions on Smart Grid**, p. 1–1, 2023.

MAAS, Andrew L; HANNUN, Awni Y; NG, Andrew Y. Rectifier nonlinearities improve neural network acoustic models. **in ICML Workshop on Deep Learning for Audio, Speech and Language Processing**, v. 28, 2013.

MALLAT, Stéphane. Group Invariant Scattering. **Communications on Pure and Applied Mathematics**, v. 65, n. 10, p. 1331–1398, 2012. ISSN 00103640.

MANCA, Marco Manolo; FAUSTINE, Anthony; PEREIRA, Lucas. Appliance recognition with combined single- and multi-label approaches. *In*: **Proceedings of the 9th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation**. New York, NY, USA: Association for Computing Machinery, 2022. (BuildSys ’22), p. 388–392. ISBN 9781450398909.

MCCULLOCH, Warren S; PITTS, Walter. A logical calculus nervous activity. **Bulletin of Mathematical Biology**, v. 6, n. l, p. 115–133, 1943. ISSN 00074985.

MUKAROH, Afifatul; LE, Thi Thu Huong; KIM, Howon. Background load denoising across complex load based on generative adversarial network to enhance load identification. **Sensors (Switzerland)**, v. 20, n. 19, p. 1–23, 2020. ISSN 14248220.

MULINARI, Bruna Machado; NOLASCO, Lucas da Silva; OROSKI, Elder; LAZZARETTI, André Eugenio; LINHARES, Robson Ribeiro; RENAUX, Douglas Paulo Bertrand. Feature extraction of v–i trajectory using 2-d fourier series for electrical load classification. **IEEE Sensors Journal**, v. 22, n. 18, p. 17988–17996, 2022.

NAIT MEZIANE, Mohamed; RAVIER, Philippe; LAMARQUE, Guy; LE BUNETEL, Jean Charles; RAINGEAUD, Yves. High accuracy event detection for Non-Intrusive Load Monitoring. **ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings**, p. 2452–2456, 2017. ISSN 15206149.

NALMPANTIS, Christoforos; VRAKAS, Dimitris. Machine learning approaches for non-intrusive load monitoring: from qualitative to quantitative comparation. **Artificial Intelligence Review**, v. 52, p. 217–243, 06 2019.

NALMPANTIS, Christoforos; VRAKAS, Dimitris. On time series representations for multi-label NILM. **Neural Computing and Applications**, Springer Science and Business Media Deutschland GmbH, v. 32, p. 17275–17290, 12 2020. ISSN 14333058.

NOLASCO, Lucas da Silva; LAZZARETTI, André Eugenio; MULINARI, Bruna Machado. DeepDFML-NILM: A new cnn-based architecture for detection, feature extraction and multi-label classification in NILM signals. **IEEE Sensors Journal**, v. 22, n. 1, p. 501–509, 2022.

NVIDIA. **Jetson TX1 Developer Kit**. 2015. https://developer.nvidia.com/embedded/jetson-tx1-developer-kit. Acessado em: 2021-10-22.

PENG, Ce; LIN, Guoying; ZHAI, Shaopeng; DING, Yi; HE, Guangyu. Non-intrusive load monitoring via deep learning based user model and appliance group model. **Energies**, MDPI AG, v. 13, p. 5629, 10 2020. ISSN 1996-1073.

PEREIRA, Lucas; NUNES, Nuno. A comparison of performance metrics for event classification in non-intrusive load monitoring. *In*: **2017 IEEE International Conference on Smart Grid Communications (SmartGridComm)**. *[S.l.: s.n.]*, 2017. p. 159–164.

PICCIALLI, Veronica; SUDOSO, Antonio M. Improving non-intrusive load disaggregation through an attention-based deep neural network. **Energies**, MDPI AG, v. 14, p. 847, 2 2021. ISSN 1996-1073.

REDMON, Joseph; DIVVALA, Santosh; GIRSHICK, Ross; FARHADI, Ali. You only look once: Unified, real-time object detection. **Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition**, v. 2016-Decem, p. 779–788, 2016. ISSN 10636919.

REDMON, Joseph; FARHADI, Ali. **YOLOv3: An Incremental Improvement**. 2018.

RENAUX, Douglas Paulo Bertrand; POTTKER, Fabiana; ANCELMO, Hellen Cristina; LAZZARETTI, André Eugenio; LIMA, Carlos Raiumundo Erig; LINHARES, Robson Ribeiro; OROSKI, Elder; NOLASCO, Lucas da Silva; LIMA, Lucas Tokarski; MULINARI, Bruna Machado; SILVA, José Reinaldo Lopes da; OMORI, Júlio Shigeaki; SANTOS, Rodrigo Braun dos. A dataset for non-intrusive load monitoring: Design and implementation. **Energies**, v. 13, n. 20, p. 1–35, 2020. ISSN 19961073.

RUANO, A.; HERNANDEZ, A.; UREÑA, J.; RUANO, M.; GARCIA, J. NILM techniques for intelligent home energy management and ambient assisted living: A review. **Energies**, MDPI AG, v. 12, n. 11, p. 2203, Jun 2019.

SONG, Jiateng; WANG, Hongbin; DU, Mingxing; PENG, Lei; ZHANG, Shuai; XU, Guizhi. Non-intrusive load identification method based on improved long short term memory network. **Energies**, MDPI AG, v. 14, p. 684, 1 2021. ISSN 1996-1073.

SOROWER, Ms. A literature survey on algorithms for multi-label learning. **Oregon State University, Corvallis**, p. 1–25, 2010.

SPANGHER, Alexander; MAY, Jonathan; SHIANG, Sz rung; DENG, Lingjia. **Multitask Learning for Class-Imbalanced Discourse Classification**. 2021.

SRIVASTAVA, Nitish; HINTON, Geoffrey; KRIZHEVSKY, Alex; SUTSKEVER, Ilya; SALAKHUTDINOV, Ruslan. Dropout: A simple way to prevent neural networks from overfitting. **Journal of Machine Learning Research**, v. 15, n. 56, p. 1929–1958, 2014. Available at: http://jmlr.org/papers/v15/srivastava14a.html.

TABATABAEI, Seyed Mostafa; DICK, Scott; XU, Wilsun. Toward Non-Intrusive Load Monitoring via Multi-Label Classification. **IEEE Transactions on Smart Grid**, v. 8, n. 1, p. 26–40, 2017. ISSN 19493053.

WANG, A. Longjun; CHEN, B. Xiaomin; WANG, C. Gang; HUA, D. D. Non-intrusive load monitoring algorithm based on features of V–I trajectory. **Electric Power Systems Research**, Elsevier B.V., v. 157, p. 134–144, 2018. ISSN 03787796.

WU, Qian; WANG, Fei. Concatenate convolutional neural networks for non-intrusive load monitoring across complex background. **Energies**, v. 12, n. 8, 2019. ISSN 1996-1073.

YANG, Yandong; ZHONG, Jing; LI, Wei; GULLIVER, T. Aaron; LI, Shufang. Semisupervised multilabel deep learning based nonintrusive load monitoring in smart grids. **IEEE Transactions on Industrial Informatics**, IEEE Computer Society, v. 16, p. 6892–6902, 11 2020. ISSN 19410050.

ZHANG, Min-Ling; LI, Yu-Kun; YANG, Hao; LIU, Xu-Ying. Towards class-imbalance aware multi-label learning. **IEEE Transactions on Cybernetics**, p. 1–13, 2020.

ZHOU, Zejian; XIANG, Yingmeng; XU, Hao; YI, Zhehan; SHI, Di; WANG, Zhiwei. A novel transfer learning-based intelligent nonintrusive load-monitoring with limited measurements.

IEEE Transactions on Instrumentation and Measurement, Institute of Electrical and Electronics Engineers Inc., v. 70, 2021. ISSN 15579662.

ZOHA, Ahmed; GLUHAK, Alexander; IMRAN, Muhammad Ali; RAJASEGARAR, Sutharshan. Non-intrusive Load Monitoring approaches for disaggregated energy sensing: A survey. **Sensors (Switzerland)**, v. 12, n. 12, p. 16838–16866, 2012. ISSN 14248220.