Development of an Open Source Energy Disaggregation Tool for the Home Automation Platform Home Assistant

Raphael M. Grund[1], Lena C. Altherr[1]

Abstract

In order to reduce energy consumption of homes, it is important to make transparent which devices consume how much energy. However, power consumption is often only monitored aggregated at the house energy meter. Disaggregating this power consumption into the contributions of individual devices can be achieved using Machine Learning. Our work aims at making state of the art disaggregation algorithms accessibe for users of the open source home automation platform Home Assistant.

Keywords

Energy Disaggregation, Machine Learning, Open Source, Home Assistant, Home Automation Platform

1 Introduction

In times of climate change and rising energy costs, saving electricity becomes more and more important. In order to save energy, an important aspect is to know what home devices are actually using energy and how much. Power disaggregation achieves this without the need for sensors at each device [1]. Power consumption data is only collected at the already present household power meter. The power consumption of individual devices can then be extracted from the collected data, e.g. using machine learning models. Knowing the power consumptions of the different devices, the end user can direct his or her attention to the main power consumers and can try to optimise usage patterns and observe if they result in energy savings. While numerous works deal with power disaggregation [1, 2, 3], few approaches have been integrated into home automation platforms so that they can be used easily and hassle-free by the end user. Goal of our project is to integrate an approach for power disaggreation based on two special types of Neural Networks into the open source home automation platform Home Assistant[4]. The whole approach is depicted in Fig. 1.

1.1 Home Assistant

Home Assistant is a well-known open source home automation platform. It is the project on GitHub with the highest contributor growth and the second most contributors overall [5]. With more than 2200 integrations [6] it is possible to connect many devices from the field of smart home, and it is also possible to connect energy meters [7]. Per default, Home Assistant records the measurements and provides an API to access the data. Home Assistant also comes with an OAuth 2.0 provider to connect external apps, with OAuth 2.0 being an industry-standard protocol for authorization [8].

[1] Faculty of Electrical Engineering and Information Technology, Aachen University of Applied Sciences Aachen, Germany

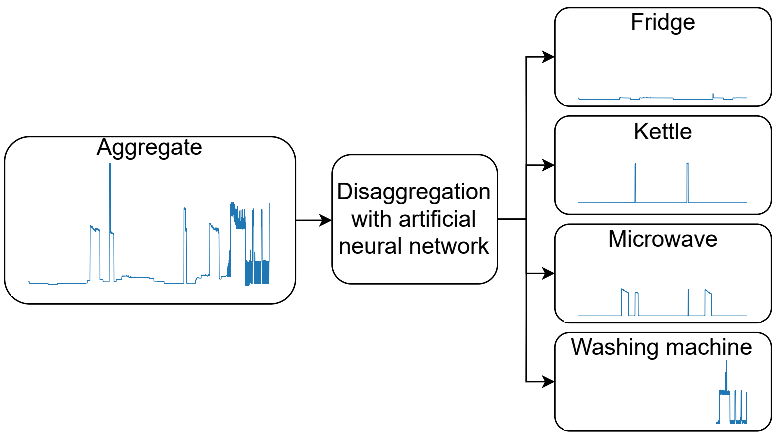

Figure 1: Overall approach: The aggregated power consumption measured at the house energy meter is disaggregated into the contributions of different household devices.

1.2 UK-DALE Dataset

For training and testing disaggregation algorithms, a publicly available dataset is used: The UK-DALE dataset [9] is comprised of the power consumption of five households in the United Kingdom. The power consumption data are recorded for each device separately. Moreover, the house-wide aggregated consumption of all appliances is given. The house-wide aggregate data are measured each second, while the per appliance data are only measured every 6 seconds. The UK-DALE dataset is available as a 2015 and a 2017 edition. The 2017 edition which is used in this work contains two additional years of data in house 1 and is available as hdf file [10].

2 Training Data

For training, we use the UK-DALE 2017 dataset provided as a hdf file. The data processing and pre-processing pipeline is adapted from [2]. The python package Pandas [11], version 1.4.3, is used for data manipulation, loading and storing.

2.1 Pre-processing

For each device, we load the power consumption data for the device as well as the aggregate power consumption of the whole house. Using the pandas resample function, we first resample the time series of the aggregated and single device consumptions to one measurement every six seconds. In a next step, both of the time series are scanned for gaps of missing data. Gaps smaller than 60 seconds are padded with the last known value, while larger gaps are padded with zeros. With both of the series being prepared in the described manner, they can be merged into one data structure, a pandas dataframe. For some devices in some of the households there is excess aggregate data at the end or beginning of the combined series. With no ground truth available, this period is removed. The dataframe containing the prepared device power consumption with aggregate power consumption is saved to disc. So there is no need to run the process more than once per device and household.

2.2 Selection of Windows

For the training of the Neural Network, data windows are selected from the dataframe that has been prepared as described in the previous section. The length of each window is determined by the device type and needs to be longer as the time the device is active.

On the other hand, the data windows should not be too long in order to conserve computing resources.

Table 1: Device Parameters used to extract activations.

Device	max power [W]	on-threshold [W]	min-on-time [s]	min-off-time [s]	window size
Kettle	3100	200	12	0	128
Fridge	300	50	60	12	512
Washing machine	2500	20	1800	160	1536
Microwave	3000	200	12	30	288
Dishwasher	2500	0	1800	1800	1536

E.g., for the kettle, a window length of only 13 minutes was chosen as boiling water with the kettle only takes a few minutes. In contrast, other devices such as a dish washer have long running programs spanning multiple hours and require a window length of 2.5 hours to have a good chance of the Neural Network to witness the complete activation of the device. The specific window sizes for the different devices are given in Table 1.

2.3 Device Activation Detection

In order to train and evaluate the performance of the Neural Networks, the data are prepared so that we can add the ground truth of when single devices were turned on or off. Fig. 2 depicts this process.

To gather when a device was on, we select all points where the device's power consumption is above a specified level which we call on-threshold, c.f. Table 1. In this list of selected timestamps we choose the first timestamp as the point in time when the device was first turned on. To derive the point when the device is turned off again or finished with its program, we iterate through the selected points until a time gap between them larger than a threshold – which we call min-off-time, c.f. Table 1 – is found. We select the last point in time before this gap as timestamp when the device was turned off.

The point in time right after the gap is the next starting point, from which we again search for the turn off time in the described manner. Additionally, if the turn on and turn off times are too close together, i.e. yielding a time span smaller than the min-on-time, the found turn on and turn off times are entirely discarded. Additionally there is a min-off-time, c.f. Table 1, this is necessary to capture the full cycle of dishwashers and washing machines, even though these devices have longer periods of low power consumption. The resulting turn on and off tuples are saved in order to save computing time later on.

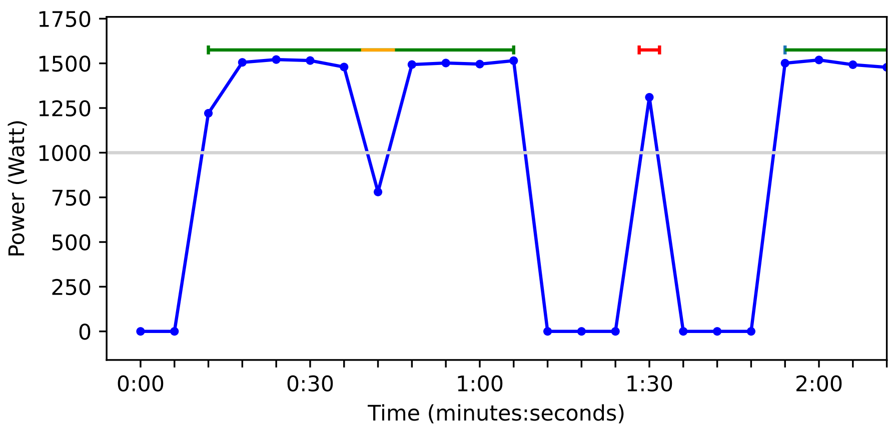

Figure 2: The detected activation of an exemplary device with an on-threshold of 1000 W (gray line), a min-off-time of 12 seconds and a min-on-time of 18 seconds. For the green segments the device's power consumption lies above the on-threshold. For the orange segments, the power is below the on-threshold, but only for a time period shorter than the min-off-time. Thus, during the green and orange segments the device is detected as being activated. The red segment is also a detected activation, but other than the green and orange time periods, this activation is discarded as its activation length is shorter than the min-on-time.

2.4 Generation of a Training Epoch

For generating the training data, we collect time spans in which the respective device was switched on. In addition, we select random time spans from the dataset, during which the device was switched off, but other devices may be switched on, yielding an aggregate power consumption unequal to zero. In a next step, the selected time spans have to be cut or padded in order to achieve uniform sequence lengths. This is done in a random manner. E.g., selected time spans being shorter than the desired sequence length are not positioned in the middle of the time window and then padded with the same number of values before and after, but positioned randomly. Like this, for each epoch, the training data is slightly different. In our approach, we opted to normalize by simply dividing both input and output data by the maximum power consumption of the respective device. Standardization by dividing by the series' standard deviation and subtracting the window's mean did not improve the results.

3 Neural Network Architecture

We implemented two different Neural Network architectures: A Denoising Autoencoder [2], and a Convolutional Neural Network [12]. The following subchapters show their respective architectures.

3.1 Denoising Autoencoder

Autoencoders are normally used to create a compact representation of the input data by only extracting the most relevant information. Additionally, Denoising Autoencoders remove unwanted noise and return a cleaned signal as output.

For energy disaggregation we can make use of this denoising feature. Our goal is to derive the disaggregated power consumption of a specific device. However, only an aggregated signal is available. This aggregated signal can be interpeted as the signal of interest, i.e. the power consumption of the considered device, superimposed by noise of the other devices. Applying the denoising autoencoder should therefore remove this unwanted "noise" and yield the disaggregated power consumption of the device under consideration.

For the implementation, we use the Denoising Autoencoder described by Jack-Kelly [2]. It consists of 5 layers. The encoding layers consist of a convolutional layer with 8 filters and a kernel size of 4, followed by a fully connected layer, comprising 8 times as many neurons as the input sequence size minus 3. The bottleneck layer is a fully connected layer with 128 nodes. Finally, the dimensions of the decoding layers are the same as the ones of the encoding layers, but in reverse. The convolutional/deconvolutional layers use a linear activation function, while the dense layers use a rectified activation function. The whole approach is depicted in Fig. 3.

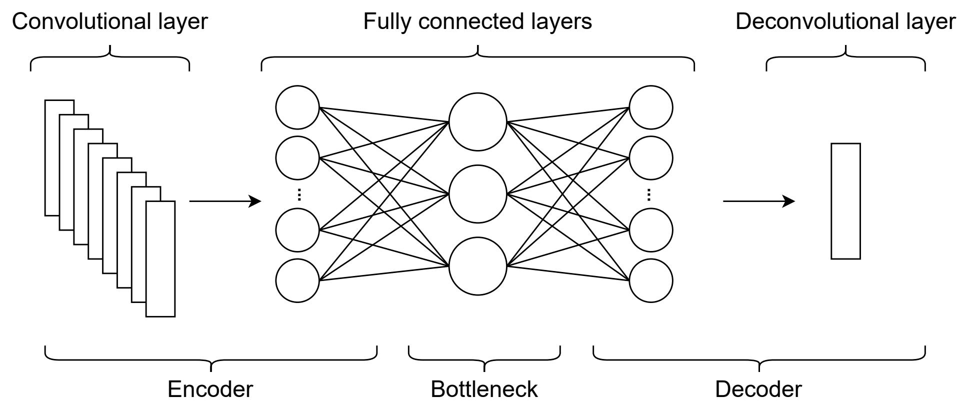

Figure 3: Structure of the Autoencoder Neural Network. A convolutional layer followed by three fully connected layers and a deconvolutional layer.

3.2 Convolutional Neural Network

Convolutional Neural Networks have shown excellent performance in pattern recognition and are most prominently used for speech recognition [13] and image classification [14]. They also have been successfully applied to energy disaggregation [15, 16].

For the implementation we use the network created by Pan et al [12]. It consists of 5 convolutional layers with 30, 30, 40, 50, and 50 filters, respectively, and kernel sizes of 10, 8, 6, 5 and 5. The convolutional layers are followed by two fully connected layers. The first fully connected layer comprises 1024 neurons, and the second one consists of as many neurons as time steps are chosen for the sequence length. All layers except the last layer use a rectified activation function. The last layer uses a linear activation. The whole approach is depicted in Fig. 4.

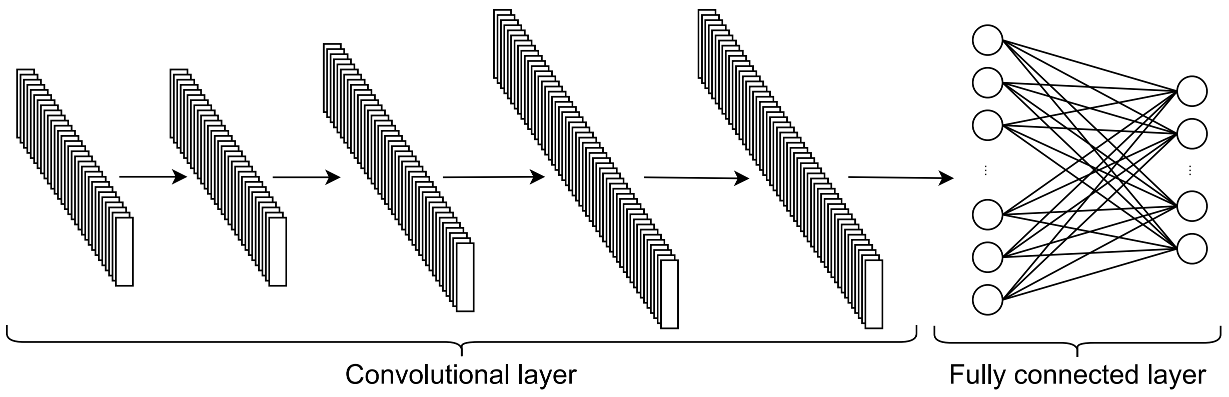

Figure 4: Structure of the used Neural Network: Five convolutional layers followed by two fully connected layers.

3.3 Neural Network Implementation

The models are implemented with Tensorflow [17] version 2.9.1 using Keras and subclassing [18]. Subclassing allows to create models like normal python classes which keep the design modular and easily extendable. The implementation is freely available in the project's GitLab repository [19]. Currently, from the GitLab Packet Registry [20] there are also trained model weights available for download for the devices microwave, kettle and fridge, with plans to upload weights for more devices.

3.4 Training and Testing of the Model

For training and testing the models, we follow an approach based on a leave-one-out cross-validation, i.e., for the five households given in the UK DALE dataset, we have five iterations. In each iteration, we choose one of the households as test data, and use the data of the remaining other households as training data. Note that if a device is not present in one of the households, the respective household is excluded from the cross-validation.

Since the time spans of available training data differ in the respective households, Table 2 shows the number of considered days in each household, as well as the number N of considered time steps after resampling the time series data to a frequency of 6 seconds.

During model training, for each epoch, the training dataset is regenerated with the training dataset generator described in Section 2.4. Furthermore, to detect overfitting, we also create a static validation set, consisting of the most recent 20% of time series data of each house in the training data set. We stop the training process, if the root mean squared error does not improve for 25 epochs. After training is completed, the checkpoint with the lowest loss on the validation set is loaded and the respective model configuration is used to disaggregate the aggregated power consumption of the test house. For house 1, due to the very long available time span, the disaggregation is restricted to the first 500 days, yielding

Table 2: Considered time spans and number of time steps N for the different households.

Household	Time Span [days]	Number of time steps N
House 1	1628	21837636
House 2	234	2780373
House 3	39	512327
House 4	205	2186446
House 5	137	1763101

$N = 6275689$ instead of $N = 21837636$ as given in Table 2. The disaggregation results are then compared to the ground truth and several metrics are calculated according to Eqs. (1) – (5).

The mean absolute error in Eq. (1) describes the average deviation of the predicted power consumption $prediction_t$ from the actual power consumption $truth_t$ at each point t in time in Watts.

For the other metrics recall, precision, f1-score, and accuracy given in Eqs. (2) – (5), the data are first discretized in the following manner: if the predicted disaggregated power consumption of a device lies above a certain device-specific minimum threshold, which we call on-threshold, the device is considered as being switched on, below this on-threshold it is considered as being switched off. The term “true positives” in Eqs. (2), (3), and (5) corresponds to the number of data points at which the device is switched on and is also correctly detected as switched on. “False positives”, on the other hand, is the number of data points at which the device is detected as switched on but is actually off. “True negatives” and “false negatives” is the equivalent for the off-state.

$$mean\_absolute\_error = \frac{\sum_{t=1}^{N} |prediction_t - truth_t|}{N} \tag{1}$$

$$recall = \frac{true\ positives}{true\ positives + false\ negatives} \tag{2}$$

$$precision = \frac{true\ positives}{true\ positives + false\ positives} \tag{3}$$

$$f1 = 2\,\frac{precision \cdot recall}{precision + recall} \tag{4}$$

$$accuracy = \frac{true\ positives + true\ negatives}{N} \tag{5}$$

4 Disaggregation Results for the Test Households

Fig. 5 shows exemplary power consumption patterns of the different devices to be disaggregated from an overall aggregated power consumption of the house. These activation patterns differ in form and time scale depending on the respective device. To disaggregate time series with an arbitrary length, one first has two split the time series into segments. For achieving this, we use windows of a device-specific constant length of several time steps, c.f. the parameter “window size” in Table 1. We create the first segment by placing the first window at the beginning of the time series. For creating the next segment, we move the window by a quarter of its width. This process is repeated until we reach the end of the time series. By segmenting the time series in this way, each window is overlapping with several other windows, and the model has several chances to observe a full device activation. After all windows are analyzed by the model, the resulting segments with the predicted devices’ power consumption are reassembled into one continuous time series. As there may exist multiple values for some of the timestamps, each set of duplicates is merged into one by computing the mean value.

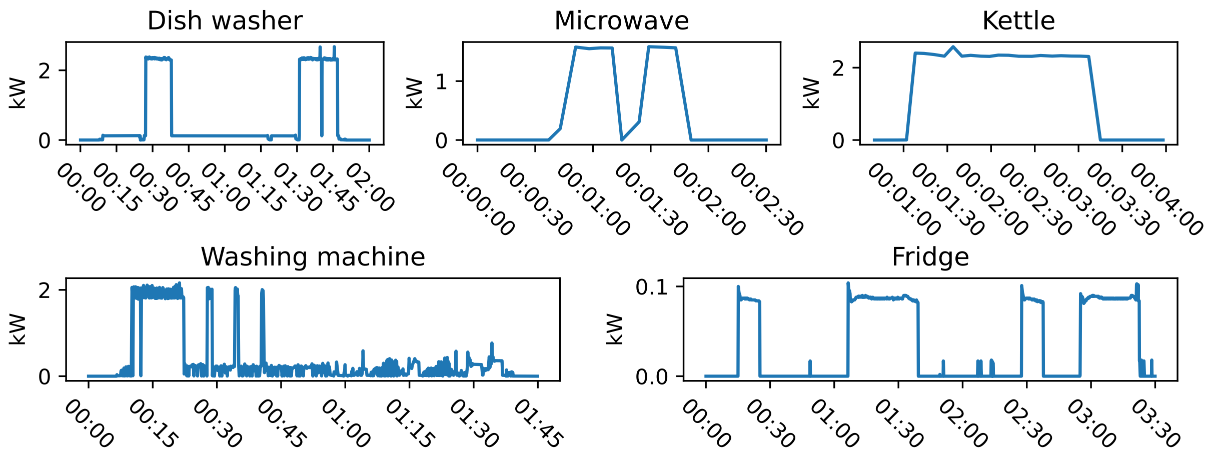

Figure 5: Examples of power consumption patterns from the devices. On the x-axis, the time is displayed in hours and minutes. For microwave and kettle, the seconds are also displayed.

4.1 Hardware and Software Configuration

For the model training and testing we used a personal computer (PC) with Linux operation system, equipped with a i7-4770k [21] processor from 2013, which worked fine for training the models for most devices. Only for the two devices washing machine and dishwasher we had to switch to a Windows PC with 32 GB of system memory due to larger window sizes and larger batch sizes.

For the end user, which will use our pre-trained models, only the time for testing, i.e. the time for disaggregation with a Neural Network model with fixed weights, is relevant. With the above mentioned hardware and software configuration, the disaggregation of 500 days with e.g. the kettle model from household 1 takes less than 5 minutes. However, the more complex model of the fridge runs for close to an hour for the disaggregation of the same time span.

The time for disaggregation can be shortened by using a more modern processor or by using a Tensor Processing Unit (TPU) coprocessor. TPUs are application-specific chips that enable high-speed data processing in pre-trained Artificial Neural Networks. There are also edge TPUs available, that can simply be connected to any system running Debian Linux (including Raspberry Pi), macOS, or Windows 10 via a USB port [22].

The disaggregation algorithm which was based on the fully trained models for the single devices was tested with Python 3.10 on Linux Fedora 36, Windows 10, as well as in the container engines Docker 20.10.16 on Ubuntu and Podman 4.2.1 on Fedora.

4.2 Results for the Denoising Autoencoder

The results of the disaggregation approach using a Denoising Autoencoder are depicted in Fig. 6. Satisfactory results in the prediction of the power consumption are obtained for the dishwasher, for which the mean absolute error stays below 30 Watts for all test houses, compared to its normal power consumption of around 2200 Watts. Also, high recall values, especially for houses 1 and 2 can be observed, showing that the disaggregation algorithm reliably detects if an activation of the dishwasher is present. However, there are also some false positives, yielding precision values unequal to one. Especially for house 5, activations of the dishwasher are detected but not actually present. The f1-score for the fridge shows little variance across the different households. The kettle model has good results in households 1 and 2, but performs worse in households 3 and 5. This could be due to the fact that in households 3 and 5 the kettle was only used about 250 times in the test period, whereas the kettle was used over 7000 times in household 1 and over 700 times in household 2. The microwave has a similar number of uses as the kettle, but the results are worse, probably due to the sawtooth like power consumption pattern of the device, cf. Fig. 5.

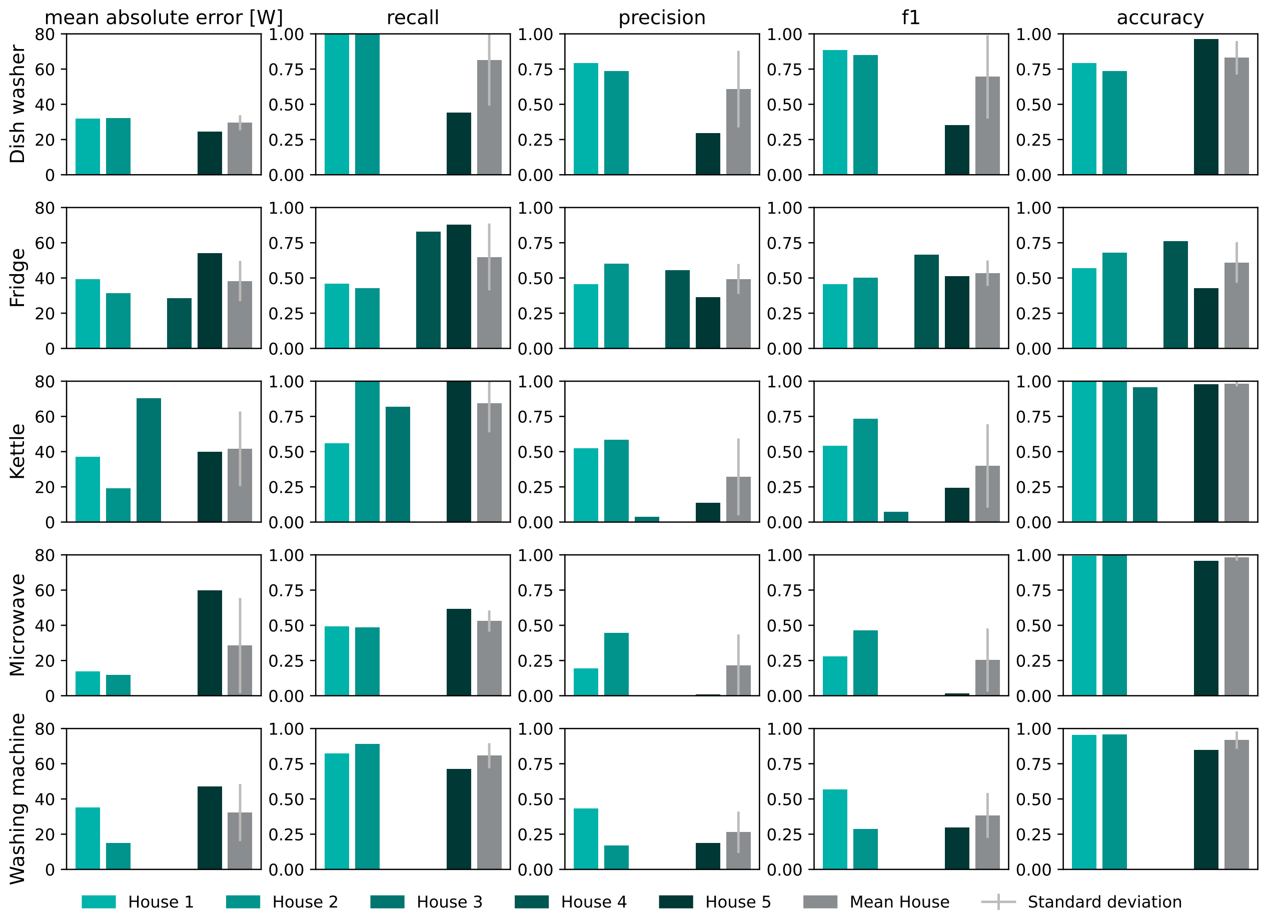

Figure 6: Testing results of the Denoising Autoencoder on the different test households when trained on data of all other households where the device is available. The green bars represent the results of the different folds of the cross-validation. The grey bar depicts the overall result of the leave-one-out cross-validation, i.e., the mean value of each metric over the different test households, as well as its standard deviation.

4.3 Results for the Convolutional Neural Network

Fig. 7 shows that the Convolutional Neural Network offers a superior performance than the Denoising Autoencoder in almost all respects. However, it shares the performance problems of the Denoising Autoencoder regarding the kettle in households 3 and 5, although these are less pronounced. The detection of the microwave also remains problematic, probably because of its changing power consumption. The correct prediction of the washing machine also remains challenging. While the Denoising Autoencoder offered better precision values, the Convolutional Neural Network yields a higher mean recall value, which shows a high standard deviation across the different houses. It is also noticeable that the accuracy values are often close to 1, but precision and recall are more mixed. This can be attributed to the fact that, unlike precision and recall, which only refer to the correct prediction of the positive class (device activated), for the calculation of the accuracy value, the correct detection of the negative class (device switched off) is also relevant. Since the devices are switched off more often than switched on, the number of true negatives can be orders of magnitudes higher than the number of true positives.

5 Integration into Home Assistant and Future Work

Our code is available in a GitLab repository [19] under the GPL-3 license. The repository also contains pre-built Docker images under the same license, and model weights under the Commons Attribution 4.0 license (CC BY 4.0). At the moment, the weights must be downloaded manually, but it is planned to enable an automatic download in the future.

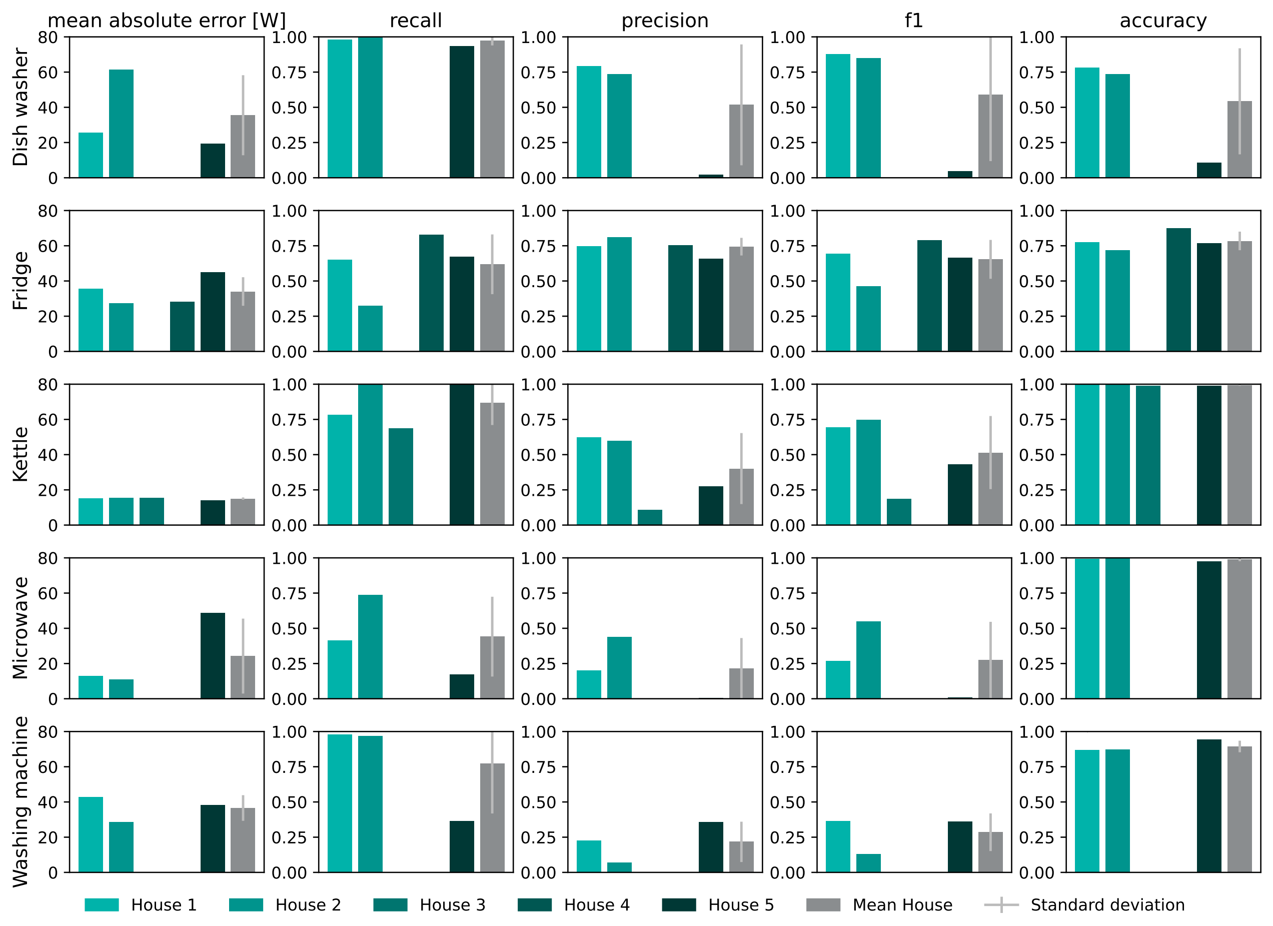

Figure 7: Testing results of the Convolutional Neural Network on the different test households when trained on data of all other households where the device is available. The grey bar depicts the result of the leave-one-out cross-validation, i.e., the mean value of each metric over the different test households, as well as its standard deviation.

For integration into Home Assistant, currently, the end user first generates a permanent access token by simply clicking a button in Home Assistant. Afterwards, the user has to run a single Docker command with this access token and the desired power meter ids as variables. If no power meter id is specified, all available meter ids will be considered.

The seamless integration into Home Assistant is not fully finished. While downloading the power meter history from Home Assistant and disaggregating this data works fine, we still want to implement an automatism to do this in regular intervals, as well as a convenient way for the user to view the disaggregated data. Unfortunately there is no possibility to add dashboard entries to historic data via the Home Assistant API. Via the API, it is only possible to set a current state. By using a custom dashboard and time series database connected through OAuth 2.0 with its own backend, this problem can be circumvented. As the Home Assistant dashboard is limited to a representation of only 10 historic days, with this approach, it is even possible to analyze a longer time frame.

Furthermore, in the future, disaggregation results on unknown homes could be improved by training the Neural Networks with additional data in order to make them more general. Several open datasets can be employed [23, 24, 25, 26].

Another interesting idea would be to test the networks with an AI accelerator like a Tensor Processing Unit (TPU) from Google Coral [22]. Such TPUs are available as edge devices, which can be connected via USB and consume only a few Watts of power. Thus, they can enable even low-powered devices (e.g. a Raspberry Pi) or an already existing NAS (Network Attached Storage) to efficiently compute the disaggregation. Unfortunately, at the time of writing no TPUs were available due to the chip shortage.

6 References

[1] K. D. Lee, "Electric load information system based on non-intrusive power monitoring," Ph.D. dissertation, Massachusetts Institute of Technology, 2003.

[2] J. Kelly and W. Knottenbelt, "Neural nilm: Deep neural networks applied to energy disaggregation," in *Proceedings of the 2nd ACM International Conference on Embedded Systems for Energy-Efficient Built Environments*, ser. BuildSys '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 55–64. [Online]. Available: https://doi.org/10.1145/2821650.2821672

[3] G. Hart, "Nonintrusive appliance load monitoring," *Proceedings of the IEEE*, vol. 80, no. 12, pp. 1870–1891, 1992.

[4] Home Assistant, "Home assistant open source home automation," https://www.home-assistant.io, 2022, last accessed 20 December 2022.

[5] GitHub, "The state of open source software," https://octoverse.github.com/2022/state-of-open-source, 2022, last accessed 13 December 2022.

[6] Home Assistant, "Integrations," https://www.home-assistant.io/integrations/, 2022, last accessed 13 December 2022.

[7] ——, "Integrating your electricity grid," https://www.home-assistant.io/docs/energy/electricity-grid/#connect-to-your-meter, 2022, last accessed 13 December 2022.

[8] A. Parecki, "Oauth 2.0," https://oauth.net/2/, 2022, last accessed 13 December 2022.

[9] J. Kelly and W. Knottenbelt, "The UK-DALE dataset, domestic appliance-level electricity demand and whole-house demand from five UK homes," *Scientific Data*, vol. 2, no. 150007, 2015.

[10] J. Kelly, "Uk domestic appliance level electricity (uk-dale) - disaggregated (6s) appliance power and aggregated (1s) whole house power," https://data.ukedc.rl.ac.uk/browse/edc/efficiency/residential/EnergyConsumption/Domestic/UK-DALE-2017/ReadMe_DALE-2017.html#Disaggregated, 2017, last accessed 13 December 2022.

[11] The pandas development team, "pandas-dev/pandas: Pandas," Feb. 2020. [Online]. Available: https://doi.org/10.5281/zenodo.3509134

[12] Y. Pan, K. Liu, Z. Shen, X. Cai, and Z. Jia, "Sequence-to-subsequence learning with conditional gan for power disaggregation," in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020, pp. 3202–3206.

[13] O. Abdel-Hamid, A.-r. Mohamed, H. Jiang, L. Deng, G. Penn, and D. Yu, "Convolutional neural networks for speech recognition," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 22, no. 10, pp. 1533–1545, 2014.

[14] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, p. 84–90, may 2017. [Online]. Available: https://doi.org/10.1145/3065386

[15] Y. Zhang, G. Yang, and S. Ma, "Non-intrusive load monitoring based on convolutional neural network with differential input," *Procedia CIRP*, vol. 83, pp. 670–674, 2019, 11th CIRP Conference on Industrial Product-Service Systems. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2212827119307243

[16] P. V. Bhupathiraju, "Deep neural networks based disaggregation of swedish household energy consumption," Master's thesis, 2020.

[17] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[18] The Tensorflow Team, "Making new layers and models via subclassing," https://www.tensorflow.org/guide/keras/custom_layers_and_models#the_model_class, 2022, last accessed 14 December 2022.

[19] R. M. Grund, "Projects gitlab repository," https://git.fh-aachen.de/energy-disaggregation-tool, 2022, last accessed 20 December 2022.

[20] ——, "Projects gitlab package registry," https://git.fh-aachen.de/energy-disaggregation-tool/disaggregation-worker/-/packages, 2022, last accessed 20 December 2022.

[21] Intel Ark, "Intel® core™ i7-4770k processor," https://ark.intel.com/content/www/us/en/ark/products/75123/intel-core-i74770k-processor-8m-cache-up-to-3-90-ghz.html, 2013, last accessed 8 January 2023.

[22] Google LLC., "Google coral products," https://coral.ai/products/, 2020, last accessed 28 December 2022.

[23] J. Z. Kolter, "Redd : A public data set for energy disaggregation research," 2011.

[24] A. S. Uttama Nambi, A. Reyes Lua, and V. R. Prasad, "Loced: Location-aware energy disaggregation framework," in *Proceedings of the 2nd ACM International Conference on Embedded Systems for Energy-Efficient Built Environments*, ser. BuildSys '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 45–54. [Online]. Available: https://doi.org/10.1145/2821650.2821659

[25] W. Kleiminger, C. Beckel, and S. Santini, "Eco data set (electricity consumption & occupancy)," 2016.

[26] D. Murray, L. Stankovic, and V. Stankovic, "An electrical load measurements dataset of united kingdom households from a two-year longitudinal study," *Scientific Data*, vol. 4, p. 160122, 01 2017.