Universiteit Utrecht

Appliance Identification using Long Short-Term Memory Recurrent Neural Networks

Master's Thesis

Jasper Makkinje

April 2018

Academic Supervisor:
Dr. G.A.W. Vreeswijk

Company Supervisor:
A.C. van Rossum

Abstract

This thesis discusses the application of Long Short-Term Memory Recurrent Neural Networks to identify electrical appliances based on the current they draw. An LSTM-model, implemented using Tensorflow, is trained and validated using the PLAID-dataset. This model achieves an average F_1-score of 92% on a testing subset of the data, thereby improving on the state of the art. The resulting model is robust to noise, and generalizes well to previously unseen examples, provided the data are pre-processed to the correct format. In conclusion, LSTMs are well-suited for the appliance identification problem, but the amount of data and computing power required restrict their practical applications.

Keywords: Appliance Identification, Long Short-Term Memory, Artificial Neural Network, Recurrent Neural Network, Crownstone, Power, Current, Voltage *

Acknowledgements

I would like to use this page to thank all the people who helped me get through the sometimes difficult times I encountered while working on this Thesis, whether it be during the internship-phase, while working on the research, or while writing the actual thesis report you see in front of you.

Thank you Anne, for allowing me to research all the things I am most passionate about in one of the most relaxed working environments I have ever had the pleasure of working in. Discussions with you, whether they were about practical thesis-related matters or just regular talks about mathematics, vegetarianism, or the future of humanity always left me with a sense of renewed enthusiasm and excitement. Without you, none of this would have been possible.

Thank you Gerard, for agreeing to be my supervisor, for arranging this wonderful internship, and for helping me by providing valuable feedback as the deadline drew nearer.

Thank you Crownstone team, for being patient with the gaps in my knowledge of sometimes very basic things, explaining these things to me where necessary, and for the gaming sessions and VrijMiBo's.

Thank you Riccardo, for being right by my side every step of the way, exchanging research papers, ideas for analysis, and memes. I'm very glad we were in this together.

Thank you Chloé, for making every day at the office enjoyable with your laughter, listening ear, and music. The Fun Side would have been nowhere near as fun if it wasn't for you.

Thank you Yana, for making sure I actually wrote something down from time to time, and checking in on me when I felt like I would never finish writing.

Thank you Sandrijn, for making me dinner when I needed it most.

Thank you UTKA, for allowing me take my mind off of everything for a few hours every week.

Thank you to my parents and stepparents, for always allowing me to complain about my struggles with 'the 9-to-5 life', and with writing this thesis.

Thank you to my brother and sister, for keeping me motivated to do the very best I can, and reassuring me when I felt like even my best wasn't good enough.

Thank you to all my colleagues at Xomnia and KLM, for providing me with the kick in the butt I needed.

And finally, thank you to everyone else I haven't explicitly mentioned here who contributed in some way, be it by proofreading my text, by listening to me complain, or just by having fun and taking my mind off of my Master Thesis for a minute in the past year.

Thank you all.

Utrecht, April 19, 2018
Jasper Makkinje

Table of Contents

List of Tables

List of Figures

List of Abbreviations

AC .... Alternating Current

ANN .... Artificial Neural Network

BT .... Bluetooth

BTLE .... Bluetooth Low-Energy

CNN .... Convolutional Neural Network

DC .... Direct Current

DL .... Deep Learning

IoT .... Internet of Things

LED .... Light-Emitting Diode

LSTM .... Long Short-Term Memory

ML .... Machine Learning

MLP .... Multi-Layer Perceptron

NALM .... Non-intrusive Appliance Load Monitoring

NILM .... Non-Intrusive Load Monitoring

PLAID .... Plug Load Appliance Identification Dataset

RNN .... Recurrent Neural Network

TSC .... Time Series Classification

TV .... Television

WHITED .... Worldwide Household and Industry Transient Energy Dataset

Chapter 1

Introduction

This chapter serves as a general introduction to the thesis, providing the reader with the context in which it should be considered. First, Section 1.1 describes the broader research context this thesis should be considered in, after which Section 1.2 gives some examples of possible applications. Then, Section 1.3 discusses why automatic appliance identification is necessary, and Section 1.4 outlines the solution proposed in this thesis. Next, Section 1.5 lists the main research question and subquestions. Finally, Section 1.6 closes the chapter by outlining the structure of the rest of the document.

1.1 Context

In recent years, as the potential dangers of global warming became more evident, many people have started looking for simple but effective ways to save energy: by installing LED-lighting, insulated glass, and smart thermostats, just to name a few examples. Simultaneously, the hype surrounding the so-called Internet of Things (IoT) has resulted in a rapidly increasing number of devices capable of sharing their data via the internet. Smart meters, for instance, are devices that measure the electricity consumption of a household and can report energy consumption directly to the power supplier. The data measured by such a device can also be leveraged to provide residents with some insight into what they can do to reduce their energy consumption. Such a reduction would not only directly reduce the monetary cost, which is a good incentive for the users, but it would reduce the user's environmental footprint as well. At the time of writing, smart meters do not yet provide such insight. Separate hardware and software are required for such functionality.

This can be done in two ways: either at plug-level, by installing hardware at the wall-socket where a device is plugged in to measure the usage of each individual appliance directly, or at meter-level, by measuring the aggregated signal at the meter, and using algorithms to disentangle this signal into the individual appliance. Plug-level measurements require relatively simple software, but, depending on the size of the building, a lot of dedicated hardware, whereas meter-level measurements require very complicated software to identify individual appliances from a noisy, aggregated signal, a process called Non-

Intrusive Load Monitoring (NILM), while the hardware used for measuring only needs to be installed in a single location. A great deal of research has been done on NILM. For an extensive list of early research in this field, refer to Heart (1995). A possible reason behind the popularity of NILM over plug-level appliance identification is likely the cost of the hardware required to perform plug-level measurements: these costs quickly becomes too high to be feasible, especially for larger (e.g office) buildings. But with Crownstone, the company for which the research surrounding this thesis was done, these cost and complexity arguments may soon be a thing of the past.

Crownstones offer an affordable, easy-to-maintain, future-proof, and standardized way of measuring plug-level data, but they do much more than this. They add the ability of turning any device on and off using one's smartphone. They allow dimming of any dimmable light bulb, again using just one's phone. They add room-level indoor positional tracking, which can be used, for instance, to automatically turn off the lights when a room is empty, or turn off the TV and other specified devices when a person leaves the house. Such functionality allows people to reduce their energy consumption in a simple and effective way, without them having to take any conscious action once everything is set-up properly. The next section describes some examples of scenarios where automatic device identification can be useful. The list is by no means exhaustive, and serves merely to provide the reader with some illustrations of use cases.

1.2 Example Applications

1.2.1 Enhanced device safety

As Crownstones are able to not only identify users, but their location (accurate up to approximately room-scale level) within the building as well, Crownstones can figure out who is in a certain room at any given time. For instance, consider a scenario where an adult is the garage, operating a dangerous power tool. When a child then enters the room, we may assume the adult is aware of this, and will make sure the child does not get hurt. But if the power tool is plugged in and turned on and a child enters the room *without* an adult being present, we may want to switch off the power tool to prevent the child from hurting themselves.

This scenario can be envisioned in a wide variety of different settings, and with a wide variety of different appliances, such as the kitchen (e.g. blender), garage (e.g. circular saw), or even the garden (e.g. lawn mower). In these scenarios, fast, accurate, and automatic appliance identification is crucial, due to the vast differences between all these appliances, and the fact that they may not be plugged-in constantly, or plugged into the same wall socket consistently.

1.2.2 Power monitoring and energy saving

Crownstones are able to monitor the current and voltage passing through them, and can therefore calculate the power usage of an appliance over time. By monitoring this power, and reacting to any irregularities, Crownstones may, for instance, be able to prevent fires resulting from short circuit, provided the Crownstone knows which device is currently plugged in, and what constitutes normal behavior for this type of device.

Similarly, by knowing exactly what device is plugged in, and what pattern of power consumption is normal, a Crownstone can power down a device when it is not in active use. Consider, for instance, a smartphone that is left to charge overnight, which reaches 100% charge within an hour of being plugged in. A Crownstone may notice this, and cut off the power to the charger, only to resume it for a short while (e.g. fifteen minutes) before the user wakes up. This saves power, without any sort of user interaction, provided the Crownstone knows which device is plugged in, and what pattern of power consumption is normal for this type of device. The same idea can be applied to a TV that has been in stand-by mode for a long time, a microwave that is only being used sporadically, or a laptop that is left permanently plugged-in, but only needs to charge occasionally. And while the aforementioned devices allow for power to be saved without any direct impact on how they are used, recognizing the device that is plugged in means this power-saving behavior can be avoided for devices that *always* need to be ready, such as alarms, doorbells, or refrigerators.

1.3 Why Automatic Appliance Identification?

Currently, users can manually enter the type of device plugged into a particular Crownstone. This allows the software to take this information into account, and, assuming it is done correctly for all Crownstones installed, would suffice to perform each of the example tasks outlined below. Then why go through the effort of automating this process? What can be gained from automatic appliance identification, when manual labeling is already in place?

Some arguments for this automation are obvious. For instance, it is more customer-friendly, as it saves the end user a step they would otherwise have to perform manually. Moreover, research into the power behavior of various devices can be relevant in and of itself, especially as smart wall sockets become more commonplace.

The key advantage of smart plugs, as opposed to dedicated smart appliances such as smart lights, is their generality: the need to buy new smart appliances disappears, as consumers can simply plug their existing devices into one of these smart plugs, thereby adding much of the smart functionality of a brand-new, expensive smart product. This added level of abstraction allows for a more consistent, general, and future-proof smart home, but this generality does come at the cost of having to deal with the enormous variety of devices out in the world: while a smart light bulb can know its own specifications, a smart plug will have to deal with all sorts of devices, without a priori knowledge of the devices' properties.

This challenge calls for an algorithm that is general (i.e. able to extrapolate to new, unseen appliances that belong to the same device category), and is able to identify quickly, as people may plug different devices into wall sockets regularly. For instance, while the refrigerator may stay plugged in to the same socket for years, a wall outlet above the kitchen counter may see a blender, panini press, and microwave plugged in, all within less than an hour.

1.4 Proposed Solution

To allow Crownstones to perform automatic appliance identification, the author proposes a Long Short-Term Memory (LSTM) Recurrent Neural Network (RNN) model (see Chapter 2), trained using current data of various appliances. This type of model was selected because it is particularly well-suited for tasks involving time-series. The final model trained for this thesis achieves state-of-the-art performance (see Chapter 5) on the PLAID dataset (see Chapter 4), indicating that this approach is highly feasible.

1.5 Research Questions

Main research question

The main question this thesis answers is the following:

"Are LSTM-models well-suited for automatic appliance identification?"

To answer this question, an LSTM-network is trained using the PLAID Dataset (for more on this dataset, see Subsection 4.2), and this model's characteristics, such as its accuracy, precision, recall, and F_1-score are compared to existing solutions.

Subquestion 1

"What level of performance can be achieved using LSTM-models?"

The ideal result from this thesis is a classifier that performs better than existing methods, across the board. To answer this question, the LSTM-model is tuned and trained to perform as well as possible on pre-existing datasets, and these results are compared to those of existing classification methods.

Subquestion 2

"How does performance change when the data are modified in various ways?"

For instance, what happens to the model's performance when stochastic noise is added to the data?

Subquestion 3

"How well does the algorithm generalize to unseen examples?"

This is a standard question in the field of machine learning, but it has two interesting aspects for this particular problem: 1. How well does the model generalize to new appliances *within the same appliance class*, and 2. How well does the model generalize to known devices in different states?

1.6 Document Structure

The rest of this document is structured as follows. First, Chapter 2 explains the principles underlying LSTM-networks. Next, Chapter 3 goes into the methods used for training, testing, and evaluation of the model, after which Chapter 4 describes the data used for this thesis. Chapter 5 then discusses the results obtained. After that, Chapter 6 talks about related literature, potential shortcomings, and possible future work. Finally, Chapter 7 rounds off the thesis with a brief conclusion.

Chapter 2

Long Short-Term Memory Recurrent Neural Networks

This chapter explains the fundamental concepts behind LSTM-networks. First, Section 2.1 discusses their relation to RNNs, and the broader scope in which they should be considered. Then, Section 2.2 briefly describes how LSTMs work, after which Section 2.3 describes some of the earlier applications. Finally, Section 2.4 explains why the choice was made to apply LSTMs to the problem of appliance identification.

2.1 RNNs

LSTMs, as briefly discussed in 1.4, are a type of Artificial Neural Network (ANN). They are a specific type of RNN, in which information passing through the network has a certain level of *persistence* that is lacking in regular ANNs. This persistence makes RNNs particularly well-suited for time-series data, as the network is able to remember input from previous timesteps when looking at the current input, and change its decision accordingly.

Mathematically, this memory-like persistence can be thought of as follows:

$$\boldsymbol{h}_t = \Phi\left(W\boldsymbol{x}_t + U\boldsymbol{h}_{t-1}\right),$$

where $\boldsymbol{h}_t$ is the network's output (also hidden state) at timestep t, Φ is a non-linear activation function (such as a sigmoid, ReLu, or hyperbolic tangent function), W is the weight matrix, similar to the one from a regular, feed-forward ANN, $\boldsymbol{x}_t$ is the input at timestep t, U is a weight matrix dictating how strongly the history from the previous timestep is factored in, and $\boldsymbol{h}_{t-1}$ is the output (hidden state) of the previous timestep. Both W and U are learned during the training process. Intuitively, this means the network can remember the previous input (e.g. the letter *q*, in a model predicting text on character-level), and use that information when predicting the next timestep (e.g. by making the letter *u* more likely).

As the number of timesteps between inputs grows, the persistence tends to grow weaker. This is caused by the fact that new input is more important than the previous input, mean-

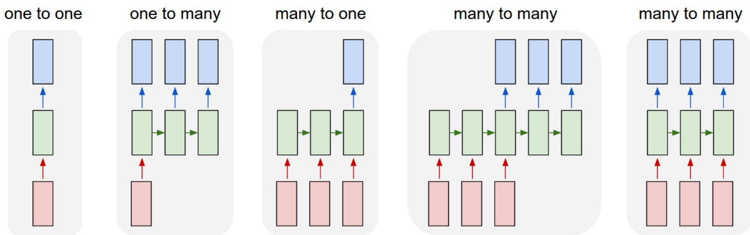

Figure 2.1: This figure shows some examples of tasks for which RNN-networks are particularly suitable. Real-world examples of these tasks include image captioning (one to many), text classification (many to one), text-to-speech (many to many), and many more. (Image courtesy of Karpathy (2015))

ing U will cause long-past input to slowly disappear over time. This phenomenon was studied extensively by Hochreiter (1991) and Bengio et al. (1994), after which LSTMs were invented to solve just this problem.

2.2 LSTMs

LSTMs were first introduced by Hochreiter and Schmidhuber (1997) to solve the problem of long-term dependencies. As mentioned in Section 2.1, they are a type of RNN with some modifications. An in-depth explanation of LSTMs is outside the scope of this thesis. The reader is encouraged to read up on the details, as many excellent articles have already been written on the topic (e.g. Olah (2015)). For completeness' sake, this section provides an overview of the fundamental concepts.

The key modifications between regular RNNs and LSTMs can be found in how the hidden state is handled. Whereas regular RNNs simply multiply the current input $\boldsymbol{x}_t$ and the previous hidden state $\boldsymbol{h}_{t-1}$ by their respective (learned) weight matrices, sum the result, and run that sum through an activation function to produce the next output, LSTMs perform a number of more complicated, interwoven steps.

First, a *forget gate layer* decides which information in the hidden state should be kept, and what should be thrown away, based on the current input and the previous output. This is done by outputting a number between zero and one for each element in the hidden state (using a sigmoid function), where zero means 'throw this element away', and one means 'keep this element in its entirety'.

Then, an *input gate layer* decides which values in the hidden state should be updated (again using a sigmoid function), and then creates a list candidate values (using a hyperbolic tangent function) that could be added to the state. These two can then be combined to figure out which elements of the state should be updated, and by what value exactly.

Next, the old cell state is actually updated using the output of the three steps explained previously. First, the forget gate layer is applied, meaning the old cell state forgets information now deemed irrelevant. Then, by applying the input gate layer, the network finds which values should be updated, and by how much exactly. This results in the new cell state.

Finally, the network decides what the output for this step should be. This is done by the *output gate layer*, which is based the current cell state, and the output of the previous timestep. The network first decides which parts of the new cell state are relevant for the next output. This is done by applying another sigmoid function to the previous output. The new cell state is then passed through a hyperbolic tangent function, the result of which is multiplied by the selection made earlier. This makes sure only sensible outputs are selected (due to the sigmoid function of the previous output), and the appropriate values get selected (due to the hyperbolic tangent function of the new cell state). Both the output and the cell state are then passed on to the next timestep, and the whole cycle repeats itself.

2.3 Applications of LSTMs

This section will briefly discuss three concrete examples of the numerous successes that have been attained using LSTMs. All three of these examples involve language in some way, which is an omnipresent form of sequential data that was historically very hard to work with for computers.

Sutskever et al. (2014) used LSTMs to perform *sequence to sequence learning*, also known as machine translation. They trained one encoder LSTM, mapping input to a fixed-dimensionality vector, and one decoder LSTM, mapping that vector back to a sentence. This setup could then be used to translate English to French, and has served as the basis for much more advanced neural translation methods that followed.

Vinyals et al. (2015) combines a Convolutional Neural Network (CNN), a type of neural network particularly well-suited for image data, with an LSTM to automatically caption images. The output of the CNN is used as the input for the LSTM, which is trained to output sentences based on that input. The resulting end-to-end solution allows for automatic captioning of images, connecting visual information to textual information, which is a crucial step in solving computer vision.

Zen et al. (2016) builds on some prior research using LSTMs for text-to-speech, by further optimizing their speed, size, and overall quality. The resulting network laid the foundations for voice models that can be used on mobile devices, resulting in more accurate voices for the increasingly omnipresent mobile voice assistants.

2.4 Using LSTMs for Appliance Identification

Over the past years, LSTMs have quickly become the go-to technique for anything involving time-series in the deep learning community. Depending on the particular use case, other techniques may give superior performance, but LSTMs tend to perform surprisingly well on many different datasets (Lipton et al. (2015)).

As appliance identification is clearly a time-series problem, but, at the time of writing, no prior research project appears to have applied LSTMs to solve it, exploring the application of LSTMs on the appliance identification problem only seems a logical next step.

Chapter 3

Method

In this section, the method employed to perform model training, evaluation, and testing is described. First, Section 3.1 describes the experimental steps taken to answer the research subquestions. Next, Section 3.2 provides details about the implementation used for the experiments, after which Section 3.3 lists the hyperparameters used during model training, and explains how these were selected..

3.1 Experimental Steps

To answer the three subquestions outlined in Section 1.5, three separate experiments were performed.

To answer subquestion 1, "What level of performance can be achieved using LSTM-models?", models with different hyperparameter settings were trained, until a satisfactory set of hyperparameters was found (for more details on this process, refer to Section 3.3). Note that these need not necessarily be the best settings that could theoretically be found, but they are the best that could be found with the available resources, and substantially better than the default hyperparameters.

To answer subquestion 2, "How does performance change when the data are modified in various ways?", the model trained for subquestion 1 was used to perform inference on data samples with increasing levels of Gaussian noise. The mean value of this noise was always set to 0, and the standard deviation was increased logarithmically. The classification accuracy, precision, recall, and F_1-score were then compared for each level of this added noise.

To answer subquestion 3, "How well does the algorithm generalize to unseen examples?", the training and testing was performed on separate subsets of the data. As a result, the test set scores shown in this thesis are an indication of how well the model generalizes to unseen examples.

3.2 Implementation

Due to their popularity, most modern machine learning toolkits offer a ready-made implementation of LSTM-nodes, so little coding has to be done regarding these fundamentals. This thesis uses the Tensorflow implementation of LSTMs, accessed through Tensorflow's Python API. For details on this API, please refer to the online Tensorflow API documentation. Tensorflow was chosen due to its popularity at the time of writing, and the active community and good documentation resulting from that popularity. The particular implementation used for this thesis is a modified version of Romijnders (2017), which is freely available online. Aside from modifications necessary to import the PLAID-dataset (i.e. changing the expected shape of the input data), addition of output statements for more in-depth performance analysis, and tuning of the hyperparameters to improve performance on this specific problem, this code was left unchanged. For more details on the hardware and software used, please refer Table 7.1 and Table 7.2, respectively, in Appendix A.

3.3 Hyperparameters

Hyperparameters are the tuning knobs of machine learning methods. They can be used to tweak, for instance, the number of layers, the amount of nodes per layer, and the maximum number of iterations of gradient descent that should be performed. The tuning of hyperparameters can be a tedious process, as the search space is incredibly large, can have many local optima, and, depending on the settings chosen, can take a long time. Currently, there is no clear, failsafe, optimal way to tune hyperparameters. In fact, state-of-the-art results that use deep learning are often found because either a lot of computing power was available, allowing for a large number of hyperparameter settings to be tried, or because a lot of time was spent just trying out different settings. Due to the limited computing resources available, the latter most closely resembles the approach used for this thesis: starting with the default hyperparameter settings , individual hyperparameters were adjusted until those with the greatest impact were found, and then a range of values was tried for those particular hyperparameters. Figure 3.1 shows how the model cost decreases with every iteration, for various settings of hyperparameters. The best model found (the purple line near the top, with the highest test set accuracy after 50,000 iterations of stochastic gradient descent), and thus the model used for all subsequent experiments, used the hyperparameters listed in Table 3.1.

Table 3.1: Optimal hyperparameters found

Hyperparameter	Setting
Batch size	1,000
Number of iterations	50,000
Dropout fraction	0.99
Number of layers	8
Cells per layer	20
Maximum gradient normalization	5
Learning rate	0.001

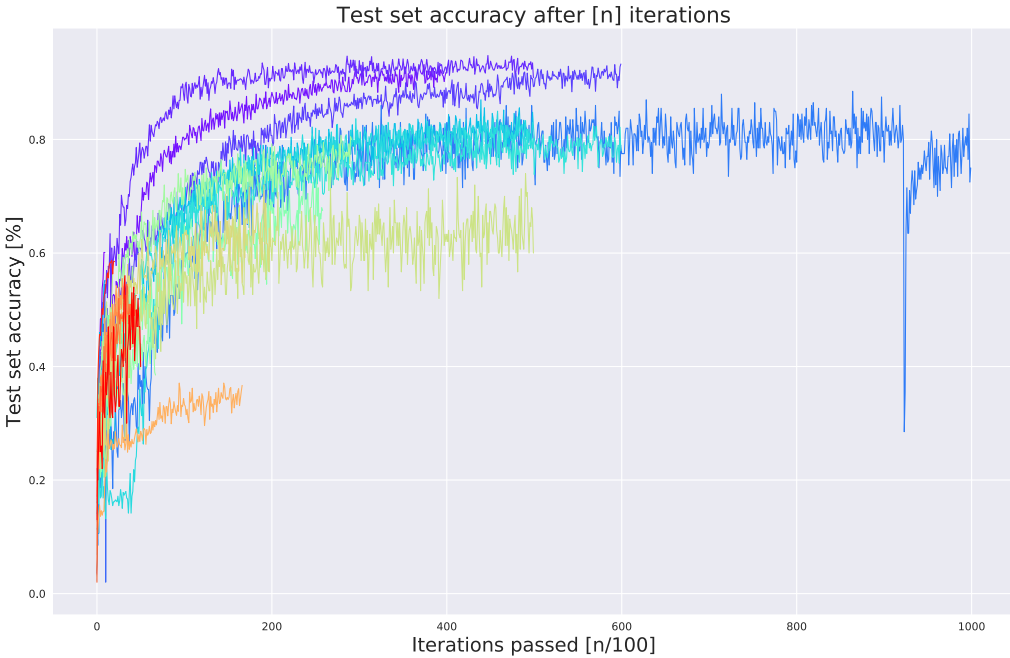

Figure 3.1: This figure shows how the accuracy climbs as the number of iterations grows. Each line represents a distinct training sequence, with a different set of hyperparameters being used for each line (omitted for legibility). Some training sequences were stopped early (e.g. the orange line that ends around 38% accuracy), as the improvements were deemed to be too slow. The jitteriness in the lines is due to the use of stochastic gradient descent, where each iteration is trained only on a subset (batch) of the total training data set. The hyperparameters outlined in table 3.1 resulted in the highest accuracy, which is here represented by the purple line at the top).

Chapter 4

Data

This chapter describes the data used to train an LSTM model for appliance identification. Section 4.1 discusses the data taken directly from Crownstones. Next, Section 4.2 describes the PLAID-dataset, after which Section 4.3 concludes the chapter by explaining how data were preprocessed before use.

4.1 Crownstone

An important source of data, and the original motivation for this project, is the Crownstones themselves. Using a Bluetooth Low-Energy (BTLE) connection, one can retrieve data samples from the current and voltage sensors inside Crownstones, which can then be stored for analysis. Data can be polled at various refresh rates, though the on-board cache memory places a restriction on the amount of data-points that each sample can contain. Due to this memory constraint, each sample can consist of at most 75 individual data points, after which there is a short gap in the measurements while the gathered data are transferred over Bluetooth.

Figure 4.1 shows a single sample of a laptop charger. The sample consists of 75 data points, gathered from a Crownstone with a sampling frequency of 3,000 Hz, meaning the whole sample spans a duration of 25 milliseconds. Note that the values for the current are relative values, and these are *not* measured in ampere. The effects of this fact, and how they can be remedied, are further discussed in Subsection 4.3.

Due to the aforementioned brief gap in measurements, and the difficulties involved in gathering the enormous amounts of data required for most deep learning-algorithms to perform well, other datasets were considered to perform model training and evaluation on. This search led to the PLAID-dataset, which was used for the experiments performed for this thesis.

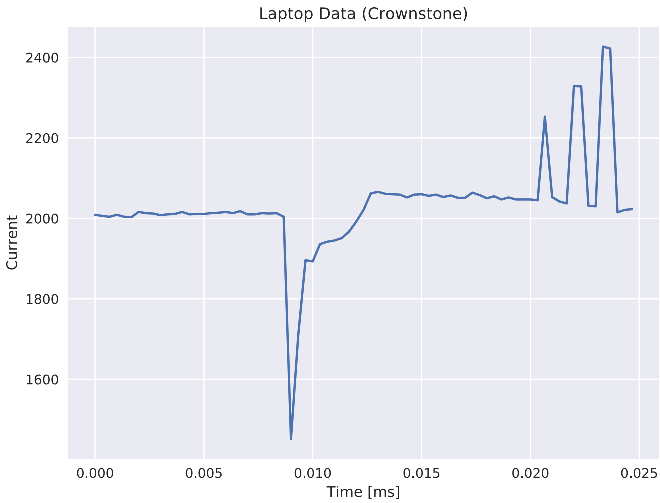

Figure 4.1: This figure shows the current drawn by a laptop charger over the course of 25 milliseconds. These data were gathered from a Crownstone, which explains the large values on the y-axis that cannot be interpreted directly as ampere.

4.2 PLAID

To train deep learning models, such as LSTMs, a large amount of data is required. One key feature of the PLAID-dataset (Gao et al. (2014)) is its size. It consists of data from 1074 distinct appliances across 11 different classes (the labels for which can be found in table 4.1), gathered from 55 individual households in the United States. These data were gathered for several (between 1.00 and 13.43) seconds at a polling frequency of 30,000 Hz. This higher frequency (recall that Crownstones sample at 3,000 Hz) allows for an interesting research subquestion regarding the generality of the algorithms used for this thesis to be answered: can a model that was trained on down-sampled PLAID-data accurately classify data obtained directly from Crownstones?

Figure 4.2 shows one sample (of a laptop charger) from the PLAID-dataset. This particular sample has been down-sampled (more on this in Subsection 4.3) to match the data coming fom Crownstones, meaning this sample consists of 75 data points at a sampling frequency of 3,000 Hz, meaning the whole sample spans a duration of 25 milliseconds. Note that the negative value is a result of Alternating Current (AC), which is converted to Direct Current (DC) by the laptop's AC-DC adapter.

Figure 4.3, from Gao et al. (2015), shows some randomly selected samples of the current data for all 11 device classes. An elaborate descriptive analysis (Gao et al. (2015)) of the PLAID-dataset can be found on the PLAID initiative's website, so to avoid duplication of information, this analysis is not included again here. The reader is encouraged to visit their website for more in-depth information on PLAID.

Table 4.1: Device labels used for PLAID-dataset

Label number	Device class
1	Air Conditioner
2	Compact Fluorescent Lamp
3	Fan
4	Refrigerator
5	Hair Dryer
6	Heater
7	Incandescent Light Bulb
8	Laptop
9	Microwave
10	Vacuum Cleaner
11	Washing Machine

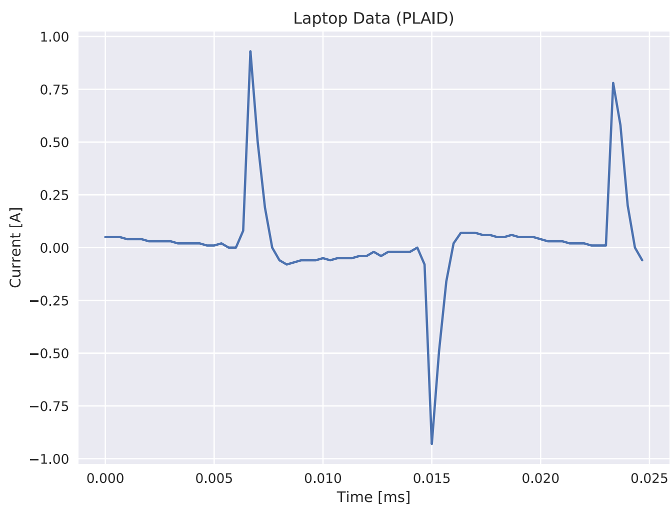

Figure 4.2: This figure shows the current drawn by a laptop charger over the course of 25 milliseconds. These data were gathered from the PLAID-dataset, meaning that unlike the Crownstone data shown earlier, the values on the y-axis for this figure are in ampere. Note that the negative values are a result of the alternating current drawn by the laptop's adapter, which uses an internal AC-DC converter to ensure that DC power be delivered to the laptop.

Because the PLAID dataset is freely available online, it has been used by other research projects in the past. Examples include Zhao et al. (2016), Adabi et al. (2015), and Barsim et al. (2016). The results from these other projects can serve as a performance benchmark to which the models trained for this thesis can be compared. This allows for straightforward comparison between the different, more classical methods used in earlier work, and the recent deep learning models that are considered in this thesis. Such a comparison can be found in Section 6.1.

4.3 Preprocessing

Training an LSTM-network is a compute-intensive process. As this intensity grows with the size of the input, the original dataset was split up before being fed into the network in order to reduce the training time required.

The original PLAID-dataset consists of an array of 1074 rows (one for each unique appliance), each of which has between 30,000 and 402,900 columns, depending on how long the device's current was measured for. These data were split up into smaller sections of 200 data points (i.e. 6.667 ms of current data) per sample, and each of these sections were individually labeled according to the sample they were taken from. As a result, the number of data samples increased significantly, from 1,074 with the original dataset, to 485,251 after this split, but each individual entry consisted of far fewer (viz. 200) data points.

To perform down-sampling by a factor 10, bringing the frequency of the PLAID-dataset of 30,000 Hz down to the sampling frequency of Crownstones of 3,000 Hz, the first data point of a sample was kept, after which the next nine were discarded before the next data point was selected, and so on. Specific values aside (as mentioned previously, the data taken from Crownstones are not measured in ampere), the down-sampled PLAID data and the data taken from directly from Crownstones look roughly similar (compare Figure 4.1 and Figure 4.2). As a result, it seems reasonable to assume that classification of Crownstone data using a model trained on PLAID data is feasible, once Crownstone data can be properly converted to units of ampere.

For comparison to the Crownstone data, the PLAID-dataset were preprocessed to mimic the data taken from Crownstones as closely as possible. For this resample, the data were down-sampled to 3,000 Hz using the method described above, and each sample was reshaped to consist of 75 data points (or 25 ms of current data), resulting in a total of 129,401 labeled examples.

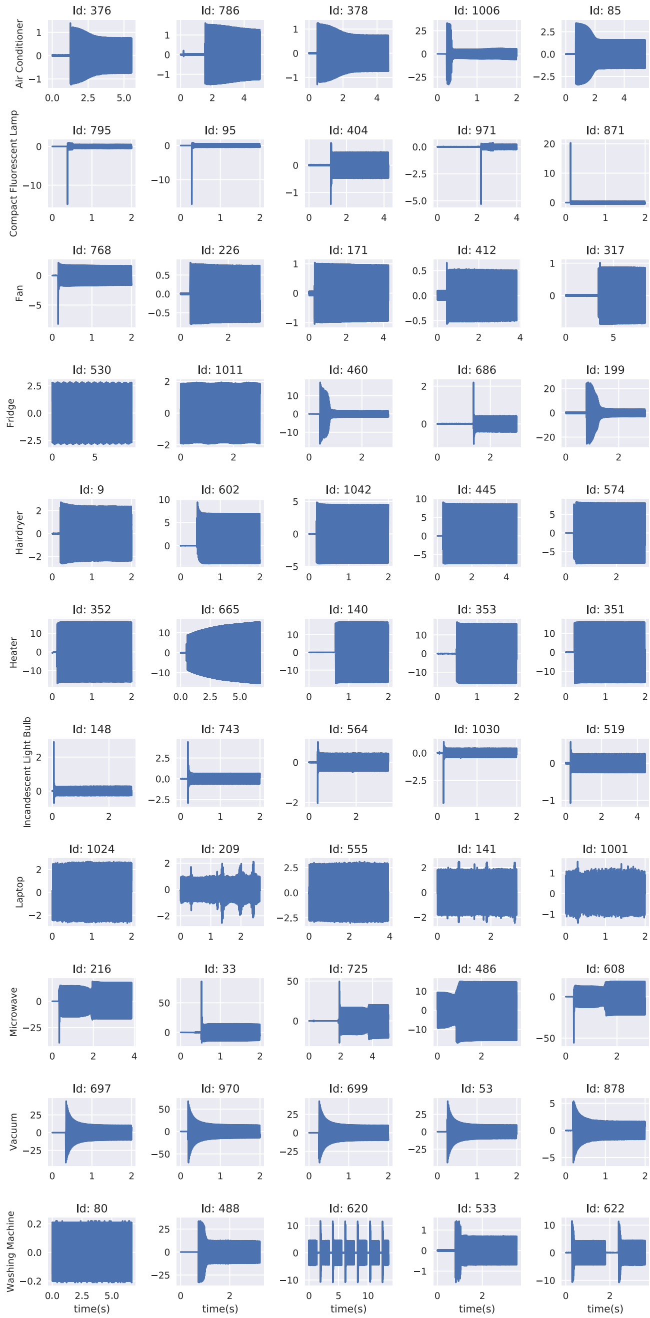

Figure 4.3: This figure shows five samples, gathered from the PLAID-dataset, of the current drawn during the transient (boot-up) phase, for all 11 appliance types. These data clearly show that even within appliance class, there can be a large difference in transient behavior, including the duration of the transient phase.

Chapter 5

Results

This section discusses the experimental results obtained with the LSTM-model. First, Section 5.1 discusses the training time required, and how this relates to changes in the hyperparameters. Next, Section 5.2 discusses some performance metrics for the best model found, after which Section 5.3 concludes by describing the model's robustness to changes in the testing data.

5.1 Required Training Time

As discussed in Subsection 3.3, the hyperparameters have a large impact on the amount of time and computing power required for training. Fully training a model for 50,000 iterations took approximately five days using the setup outlined in Appendix A. Of course, increasing the learning rate, reducing the number of iterations, decreasing the batch size, or reducing the number of layers or nodes per layer can reduce this time, possibly at the cost of worse performance. Training time could also be reduced by training this model on an NVIDIA GPU, but as such hardware was not available for this project, training had to be performed on a CPU instead.

5.2 Model Performance

With the right hyperparameters, this model performs well. The mean F_1-score, i.e. the harmonic average of the precision and recall across all 11 appliance classes, on the Crownstone-split (see Subsection 4.3 for more details on how this split was made) of the PLAID-dataset is 92%, which is higher than other state-of-the-art results using more classical approaches. More details on the model's performance can be found in the classification report in Table 5.1, and the comparison with the literature in Section 6.1.

Table 5.1: PLAID test set classification report

Device Class	Precision	Recall	F_1-score	Support
Air Conditioner	0.96	0.85	0.90	921
Compact Fluorescent Lamp	0.69	0.99	0.82	1699
Fan	0.97	0.89	0.93	1498
Refrigerator	0.93	0.86	0.89	587
Hair Dryer	0.99	0.86	0.92	1590
Heater	0.96	0.96	0.96	442
Incandescent Light Bulb	0.97	0.87	0.91	1266
Laptop	0.99	0.97	0.98	1874
Microwave	0.95	0.95	0.95	2247
Vacuum Cleaner	0.98	0.88	0.93	325
Washing Machine	0.98	0.86	0.92	491
Average / Total	0.93	0.92	0.92	12940

5.3 Model Robustness

To investigate how robust the model is when noise is introduced, Gaussian noise with an increasing standard deviation was added before classifying the test set using the trained model. The figures below show how the accuracy, precision, recall, and F_1-score (Figure 5.1), cost (Figure 5.2), and classifications (Figure 5.3) change when exponentially increasing the standard deviation of added the Gaussian noise, while keeping the noise's mean fixed at 0.

The scores and cost behave predictably: small amounts of noise hardly affect these values, but as the noise becomes larger, the model is finding it increasingly difficult to perform accurate classification. Interestingly, as the noise becomes very large ($\sigma > 10$), a small *increase* in performance can be observed. From the confusion matrices, it becomes clear that samples are more likely to be classified as larger appliances (i.e. appliances that tend to draw more current, such as air conditioners, microwaves, and washing machines) as the amount of noise is increased. This is explained by the fact that the PLAID-data are not normalized across classes, meaning the actual values of the current matter. Intuitively, adding large-valued noise to the original data will cause the data to be more extreme, meaning classification of the larger appliances becomes more likely.

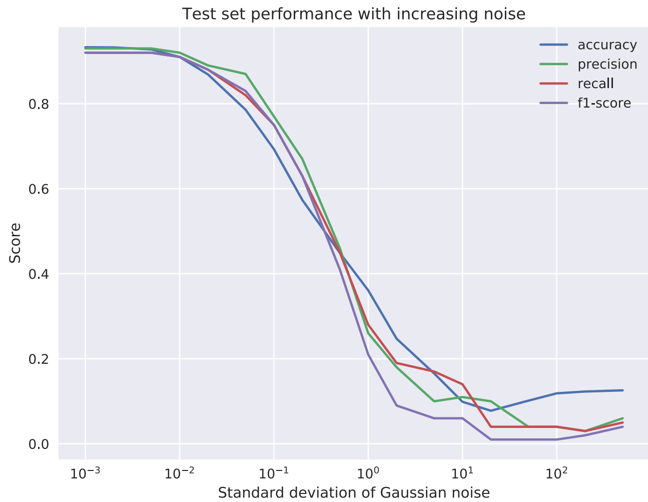

Figure 5.1: This figure shows how the accuracy, precision, recall, and F_1-score on the test set decrease as Gaussian noise (with zero mean and increasing standard deviation) is added to the data. Note that the x-axis has a logarithmic scale.

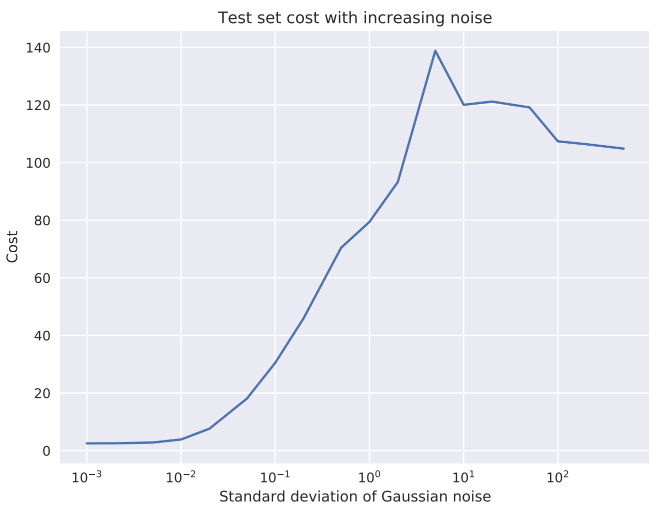

Figure 5.2: This figure shows how the cost on the test set classification increases as Gaussian noise (with zero mean and increasing standard deviation) is added to the data. Note that the x-axis has a logarithmic scale.

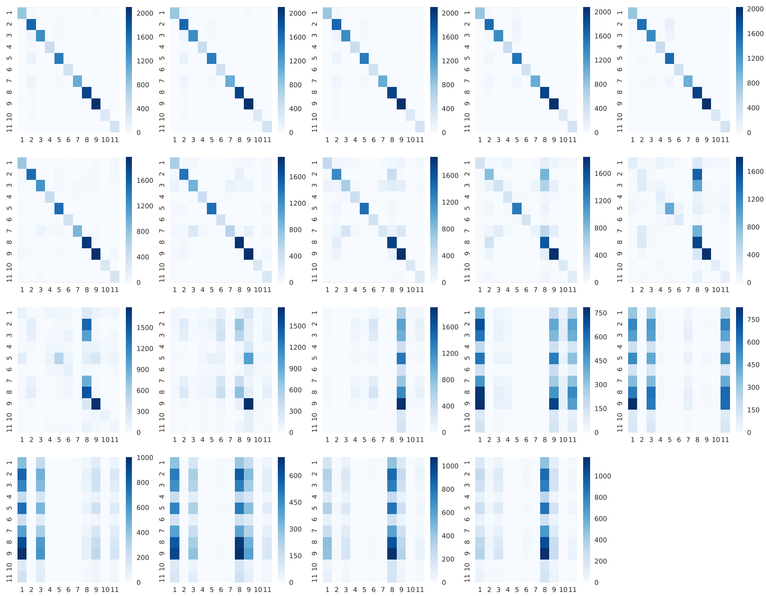

Figure 5.3: This figure shows how the classification confusion matrix changes as more Gaussian noise is added. As the noise increases and causing the data to take on increasingly large values, each sample gets classified as a large appliance.

Chapter 6

Discussion

This chapter goes into some of the context surrounding this thesis. First, Section 6.1 discusses related literature. Then, Section 6.2 mentions some complicating factors encountered in the process of writing this thesis, and finally Section 6.3 discusses some possible examples of future work.

6.1 Comparison to the Literature

6.1.1 Datasets

Two recent, public databases for appliance identification based on plug-level power data are available: PLAID (Gao et al. (2014)) and WHITED (Kahl et al. (2016)). PLAID contains high-resolution measurements of appliances in 11 device classes, gathered from 55 households across the US. WHITED contains few (for many device classes, only 1) high-resolution measurements for 46 device classes,

Due to the data-hungry nature of deep learning-methods examined in this thesis, the PLAID-dataset was select for model training and evaluation. The low number of unique appliances per class in the WHITED-dataset would cause the resulting model to generalize less easily, which would hurt performance on Crownstone data. For more detailed information about the PLAID-dataset, see Section 4.2.

6.1.2 NILM and appliance identification

Appliance identification can, as discussed in Section 1.1, be performed either on meter-level data, to try and disentangle an aggregated signal into individual appliances, or directly on plug-level data. Given the context of this thesis, i.e. the Crownstone product, the latter is more relevant. Unfortunately, it was also studied far less, as meter-level NILM is easier to perform on a large scale, due to the relatively small amount of dedicated hardware required. For the sake of brevity and relevance, this section will discuss research into plug-level appliance identification only. For an elaborate summary of early research into (meter-level) NILM, refer to Heart (1995).

6.1.3 Deep learning techniques in appliance identification

The general context of this thesis, i.e. the application of modern deep learning-techniques to the problem of appliance identicialification, has not been studied much up to this point. The most relevant study at the time of writing is De Baets et al. (2018). For this paper, researchers plotted the VI-trajectories for the various appliances, converted these plots to images, and then used CNNs to classify these images. This approach led to an average F_1-score of 77.6% on the PLAID-dataset (compared to 92% obtained for this thesis), and an average F_1-score of 75.46% on the WHITED-dataset, which, while not state-of-the-art, is a interesting result given the indirect approach used. The key appliances holding back the performance, the authors state, are appliances that have similar electrical components: fans and the air conditioners are confused (as both contain fans), and both washing machines and refrigerators contain a motor.

A study done by Rizky Pratama et al. (2018) seems similar to this thesis, but the approach used there is quite different. They use LSTM-networks to classify, on an aggregate, meter-level signal, which appliances are active in a given room. They do this by creating 14 classes, each of which comprises a combination of appliances being used, and assign the aggregate signal to one of these classes. As a result, they are able to identify which appliances are being used in a given room, which can be considered a form of indirect appliance identification. Their approach would not scale well in the context of Crownstone, though, as it requires prior knowledge of which appliances are present in a room, and what the aggregated power usage of various combinations of these appliances looks like.

Another initiative worth mentioning is TimeNet Malhotra et al. (2017). This is a recently published pre-trained LSTM-network designed for time-series classification purposes. Sadly, the authors have not published the code for this network online, so it could not be tested.

The particular contribution of this thesis, i.e. the application of LSTM-networks to classify appliances directly using their current drawn over time, appears not to have been studied before. This makes immediate comparison to the literature difficult, and has as a side-effect that the hyperparameters, as discussed in 3.3, had to be experimentally determined for this thesis, and could not be based on any prior research.

6.2 Complicating Factors

6.2.1 Random initialization

As with all deep learning and machine learning algorithms, the initial state from which one starts training the network can have a tremendous effect on the performance of the final model. Weight initialization is done at random (though different algorithms may employ somewhat different techniques), and this randomness can happen to work in your favor, or happen to work against you.

Training an identical model several times with different random initializations can show the impact of this factor, as well as allow researchers to cherry-pick the best resulting model at the end. Due to time and computation restrictions, this analysis will be left for future research.

6.2.2 Hyperparameters

As discussed in Chapter 5, the hyperparameters have an enormous amount of influence on the model performance. They determine how quickly the model learns, to what level of performance it converges, and how well the model will generalize to unseen examples.

Getting the code to run is the easy part of deep learning; finding a set of hyperparameters that results in good model performance is where the work starts. For this thesis, many different sets of hyperparameters were experimented with, until the set resulting in the final model used for the experiments was found. This model was the best found, but it should by no means be considered optimal. Training a model for long enough to be able to judge whether or not its performance is promising (i.e. possibly better than previous attempts) takes several hours up to an entire day on the hardware used for this thesis. As a result, only a limited number of models were trained, using a limited set of hyperparameters. Further research might optimize these further, and thus produce models that perform better using the exact same algorithms.

6.2.3 Steady-state versus transient behavior

One thing to note for the splits made for this thesis (as described in 4.3) is that the training dataset now consists of mostly steady-state behavior. As a result, the high accuracy of the model likely reflects the fact that the model has learned to recognize the devices' steady-state behavior well. As this is the state devices generally spend most time in, one could argue that recognizing this particular state is most important. Still, an ideal model should be able recognize both the transient and the steady-state behavior, so the present work could be extended by gathering a large amount of data on the transient states as well, and training a model on those data.

6.2.4 Generality of the data

The PLAID-dataset used to train the models for this Thesis was, as indicated in Section 4.2, gathered in the United States. While the appliances themselves likely originated from all over the world, the potential effects of the US power grid were not studied in this thesis. Having a dataset with data from all over the globe would remedy this problem, but is substantially harder to obtain.

Furthermore, a downside of the PLAID-dataset is that, since it was gathered by a third party, the 11 different device classes for which data were gathered were not selected specifically for this research project, nor to be of the most use to Crownstone. As a result, the device classes do not align very well with those expected to be used most by Crownstone's customers. Therefore, while its size makes it a very suitable dataset to perform model training and analysis on, the resulting model is not of much use for Crownstone directly, as many of the devices customers are expected to plug in often were not part of the PLAID-dataset. Still, this thesis does prove that such models can be trained using deep learning, provided that enough data are available.

6.2.5 Computing requirements

The performance achieved using the methods discussed in this paper is better than that of other state-of-the-art methods, such as De Baets et al. (2018). Still, the computational requirements of these methods should also be taken into account. Clearly, if a model has vastly superior performance and requires marginally more time and/or computing power to run, the newer model is still at an advantage. But if two models share similar performance, but one is far more computationally efficient, then the more efficient model will be preferred in real-world applications.

Deep learning has been widely criticized for its enormous requirements regarding available training data, computing power, and computing time. While many methods have become more efficient in recent years, more traditional methods are still superior for many applications. But knowing that well-performing models can be trained, the question remains whether deep learning is a *sensible* approach for appliance identification.

Only time will tell what the answer to this question will be. But even with the limited amount of time and computing power available for this thesis, state-of-the-art accuracy was achieved using the methods described. With further tuning of the hyperparameters, the use of more computing power, and the availability of more training data, performance is likely to increase further, meaning the state of the art would thereby forward. And while it cannot be denied that the performance per unit of computing power, especially during the training phase, is dramatically lower than that of simpler, more traditional methods, computing power may not be as much of an issue in the future. Which method is preferable ultimately depends on the specific application for which it is used.

Finally, the field of deep learning as a whole is, of course, by no means stationary. Existing methods will become more efficient, entirely new methods will be thought of, the amount of available data will continue to grow, and computing power will keep increasing and becoming cheaper. Over time, as these improvements add up, traditional methods may lose their preferential status, as the (potential) simplicity of applying deep learning to problems for which enough data are available starts outweighing computational efficiency.

6.3 Future Work

6.3.1 More efficient algorithms

While the results obtained for this thesis are good enough to be useful in a real-world application, the memory and computing power required for inference are still too high to be run locally on the Crownstones themselves. While inference can be performed quickly, the model is in the order of 100 MB in size, and therefore requires the storage of at least a smartphone, or even the cloud. More efficient and/or compression of models should solve this problem, and allow inference to run directly on the Crownstones themselves, which would result in an even quicker and more seamless end-user experience.

6.3.2 Device state identification

Many devices, such as simple lights, paper shredders, or toasters, only have a single state, or maybe an *on* and an *off* state, while others, such as televisions, washing machines, and hair dryers, might have many different states. Even if we disregard the boot-up behavior of all these devices, this still results in a variety of different current patterns that can occur, even for a single device. If we then also factor in the amount of distinct devices that fall into a certain device class, all of which can have their own distinct set of unique states, it becomes clear how difficult it is to obtain a database that covers enough of these scenarios to be truly general.

Obtaining labels for the different states these devices can be in, perhaps by automatically recognizing that a previously-labeled device is now in an unseen state, and adding a new label accordingly, would significantly increase the generality of these methods, and enable long-term pattern recognition, as discussed in Subsection 6.3.3.

6.3.3 Long-term pattern recognition

If the same device remains plugged into the same Crownstone for a longer period of time, and that Crownstone is able to identify the various states this device can be in (as discussed in Section 6.3.2), this information can be used to identify common usage patterns in these data. For instance, if a certain television is never used during working hours, the Crownstone it is plugged into might switch it off entirely (as opposed to keeping it in stand-by mode) in order to save power. Alternatively, if an elderly person turns on the bathroom light for approximately 20 minutes every morning, and the Crownstone notices that one morning the light is suddenly on for much longer (e.g. more than two standard deviations longer than normal), the Crownstone might notify a family member, as the user may have fallen and become unconscious.

While many such scenarios can be imagined, user-friendliness and privacy always need to be taken into account. While recognizing some of these high-level patterns may be relatively straightforward given the state-information of certain devices, the ethical concern remains whether these data can actually be used, whom they belong to, and whether it is truly more user-friendly.

Chapter 7

Conclusion

This thesis discusses how an LSTM-model was trained to perform appliance identification, how this resulting model was tested, and what results were obtained. The resulting LSTM-model performs substantially better than recent related work, obtaining a macro-average F_1-score of 92%, compared to $77,6\%$ obtained by De Baets et al. (2018) and 89% obtained by Barsim et al. (2016). Adding noise to the data predictably reduces performance, but the models are robust to small levels of noise. The resulting model generalizes well to unseen examples, provided that new data be in a format comparable to the data used during training.

To answer the main research question: Yes, LSTM-models can perform well on the appliance identification task, given the availability of enough training data and computing power. This is hardly a surprising result. LSTMs had been applied successfully to other time series classification-tasks before (meaning the model can work, given appropriate training data), and appliance identification had been previously performed using other techniques (meaning the data are classifiable). A key advantage of LSTMs over more traditional methods is the lack of in-depth knowledge required regarding electronics: simply providing the model with enough data suffices to train an accurate, robust, and general model, that can be easily updated and deployed to production using Tensorflow. And provided the data are stored in a secure fashion, the model can continue to improve over time, learning to recognize more devices, device states, and long-term usage patterns.

In conclusion, the two main take-home messages from this thesis are the following: LSTM-models are very well-suited for appliance identification, but obtaining the large amount of data and computing power necessary to train them properly is costly.

Bibliography

A. Adabi, P. Mantey, E. Holmegaard, and M. B. Kjaergaard. Status and challenges of residential and industrial non-intrusive load monitoring. In *Technologies for Sustainability (SusTech), 2015 IEEE Conference on*, pages 181–188. IEEE, 2015.

K. S. Barsim, L. Mauch, and B. Yang. Neural network ensembles to real-time identification of plug-level appliance measurements. *Signature*, 2(11), 2016.

Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

L. De Baets, J. Ruyssinck, C. Develder, T. Dhaene, and D. Deschrijver. Appliance classification using vi trajectories and convolutional neural networks. *Energy and Buildings*, 158:32–36, 2018.

J. Gao, S. Giri, E. C. Kara, and M. Bergés. Plaid: A public dataset of high-resolution electrical appliance measurements for load identification research: Demo abstract. In *Proceedings of the 1st ACM Conference on Embedded Systems for Energy-Efficient Buildings*, BuildSys '14, pages 198–199, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-3144-9. doi: 10.1145/2674061.2675032. URL `http://doi.acm.org/10.1145/2674061.2675032`.

J. Gao, E. C. Kara, S. Giri, and M. Bergés. A feasibility study of automated plug-load identification from high-frequency measurements. In *Signal and Information Processing (GlobalSIP), 2015 IEEE Global Conference on*, pages 220–224. IEEE, 2015.

G. W. Heart. Nonintrusive appliance load monitor published references. `http://www.georgehart.com/research/nalmrefs.html`, 1995.

S. Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*, 91:1, 1991.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

M. Kahl, A. U. Haq, T. Kriechbaumer, and H.-A. Jacobsen. Whited-a worldwide household and industry transient energy data set. In *3rd International Workshop on Non-Intrusive Load Monitoring*, 2016.

A. Karpathy. The unreasonable effectiveness of recurrent neural networks. `https://karpathy.github.io/2015/05/21/rnn-effectiveness/`, 2015.

Z. C. Lipton, J. Berkowitz, and C. Elkan. A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019*, 2015.

P. Malhotra, V. TV, L. Vig, P. Agarwal, and G. Shroff. Timenet: Pre-trained deep recurrent neural network for time series classification. *arXiv preprint arXiv:1706.08838*, 2017.

C. Olah. Understanding lstm networks. `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`, 2015.

A. Rizky Pratama, F. J. Simanjuntak, A. Lazovik, and M. Aiello. *Low-power Appliance Recognition using Recurrent Neural Networks*. 2018.

R. Romijnders. Lstm tsc. `https://github.com/RobRomijnders/LSTM_tsc.git`, 2017.

I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

O. Vinyals, A. Toshev, S. Bengio, and D. Erhan. Show and tell: A neural image caption generator. In *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*, pages 3156–3164. IEEE, 2015.

H. Zen, Y. Agiomyrgiannakis, N. Egberts, F. Henderson, and P. Szczepaniak. Fast, compact, and high quality lstm-rnn based statistical parametric speech synthesizers for mobile devices. *arXiv preprint arXiv:1606.06061*, 2016.

B. Zhao, L. Stankovic, and V. Stankovic. On a training-less solution for non-intrusive appliance load monitoring using graph signal processing. *IEEE Access*, 4:1784–1799, 2016.

Appendix A

Experimental Setup

All models discussed in this thesis were trained and evaluated on a desktop computer with the specifications found in table 7.1 (hardware), and the software specified in table 7.2.

Table 7.1: Specification of the hardware used

Component	Specification
CPU	Intel Core i5-4590
GPU (not used)	Sapphire R9 290 OC Tri-X
GPU memory (not used)	4 GB GDDR5
RAM	16 GB (4x4GB) Crucial Ballistix Tactical
Main hard drive	Crucial MX100 256GB
Secondary hard drive	Seagate Barracuda 7200.14
Motherboard	Gigabyte GA-Z97-D3H

Table 7.2: Specification of the software used

Software	Version
Windows 10	Build 17025.1000
Linux Subsystem	Ubuntu 16.04.3 LTS
Tensorflow	1.1.0
Python	3.5.2
Numpy	1.13.3
MatPlotLib	2.1.0
Seaborn	0.8.1
Pandas	0.21.0