Summarizing electricity usage with a neural network

Christopher Sipola

Master of Science
Artificial Intelligence
School of Informatics
University of Edinburgh
2017

Abstract

This project explores whether a neural network is capable of predicting summary statistics of electricity usage for five common household appliances given only the aggregate signal of a smart meter. These appliance-level statistics are needed for many kinds of feedback and analytics provided to energy consumers so they can reduce electricity consumption and save on energy bills. An example of such a statistic is the total energy used by a washing machine in a day. Currently the state-of-the-art approach is to use non-intrusive load monitoring (NILM) to predict appliance-level signals timepoint-by-timepoint, and then compute the statistics using these predictions. However, this is indirect, computationally expensive and generally a harder learning problem.

The statistics can also be used as input into one of the most successful NILM models, the additive factorial hidden Markov model (AFHMM) with latent Bayesian melding (LBM). This model uses these appliance-level statistics as priors to significantly improve timepoint-by-timepoint predictions. However, the model is currently limited to using national averages, since so far there have been no methods for inferring day- and house-specific statistics. Improved statistics can therefore lead to more effective NILM models.

Since this type of learning problem is unexplored, we use a dynamic architecture generation process to find networks that perform well. We also implement a new process for generating more realistic synthetic data that preserves some cross-appliance usage information. Results show that a neural network is in fact capable of predicting appliance-level summary statistics. More importantly, most models generalize successfully to houses that were not used in training and validation, with the best-performing models having an error that is less than half the baseline.

Acknowledgements

I would like to thank my supervisor Nigel Goddard, as well as Mingjun Zhong and Chaoyun Zhang, for their guidance throughout this project.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Christopher Sipola*)

Table of Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Motivation

Providing personalized feedback of energy usage to homeowners can decrease energy consumption, saving homeowners 5–12% on their total energy bills. The most effective feedback is frequent (e.g., given daily), appliance-specific and actionable [7]. Example feedback might be: "Your washing machine ran four times in the last week, using 6 kWh and costing 90p. Over a year, this translates into about 300 kWh and £45." These sort of energy savings are the goal of the IDEAL research project at the University of Edinburgh, which explores how smart technology can be used to encourage homeowners to use less electricity and gas in their homes [8].

A core technology used in this effort is the smart meter, which records the aggregate power level of the house at a high sampling rate. The use of smart meters worldwide is predicted to increase in coming years. For example, researchers predict that 72% of energy customers in EU countries will have smart meters by 2020 in order to comply with EU energy market legislation. The goal of the worldwide roll-outs is "to better manage residential energy demand, conserve energy, improve billing accuracy, and help users understand the energy implications of their appliance usage habits" [3].

While smart meters provide granular house-level electricity usage data, they do not provide appliance-level information, making personalized feedback much less effective. Unfortunately, recording appliance-level data directly using individual appliance monitors (IAMs) can be expensive, costing $300–600 per house [9].[1] A more cost-effective approach is non-intrusive load monitoring (NILM),[2] or energy disaggrega-

[1] These estimates assume the use of devices from companies like Kill-A-Watt or EnergyHub.

[2] Non-intrusive load monitoring (NILM) is also referred to as non-intrusive appliance load monitor-

tion, which is the use of software to disaggregate the signal from a smart meter into the signals of the appliances.

NILM is often framed as a supervised machine learning problem, so training the disaggregation models requires appliance-level data. There are a number of studies that have collected such data using IAMs. Arguably the best dataset for aggregate- and appliance-level data is REFIT, which has a combination of a long study duration, a large number of participating houses and a high sampling rate [10].

While disaggregation is a powerful tool for demystifying energy usage, some challenges remain. One challenge pertains to a notable state-of-the-art model, the additive factorial hidden Markov model (AFHMM), which uses latent Bayesian melding (LBM) to incorporate appliance-level statistics for more accurate predictions. Currently this model is limited to using national averages, and it would likely see improved performance if the statistics were inferred specifically for each house and day [11]. A second challenge is that computing summary statistics by first disaggregating into the appliance series timepoint-by-timepoint is computationally expensive. It is likely that a more straightforward approach would be more efficient. A third challenge is that computing certain summary statistics on noisy timepoint-by-timepoint predictions may be difficult. For example, simple methods that count appliance cycles are meant to be used on clean IAM data, so using them on noisy predictions can introduce another source of error.[3] And a fourth challenge is that using NILM for summarization is inherently a difficult learning problem. This is because NILM models are incapable of, for example, using cross-appliance information across the day to make predictions, or trading off a positive error early in the day against a negative error later in the day.

The goal of this dissertation project is to support IDEAL by exploring whether a model can summarize appliance-level statistics *directly*, given the aggregate signal of a smart meter. The two statistics that will be predicted are the number of activations—that is, the number of times an appliance is used—and energy used by the appliance. The five appliances that will be modeled are the fridge, kettle, washing machine, dishwasher, and microwave. In Figure 1.1, this would mean a model takes the aggregate power series (blue) as input, and returns, for example, the energy used by the washing machine series (pink) as output.

The project uses neural networks since they are generally good at modeling data with complex structure between input features and have already proved successful

ing, abbreviated NALM or NAILM.

[3] An example of such a method is by using the `Electric.get_activations()` function. See Section 3.3.1.

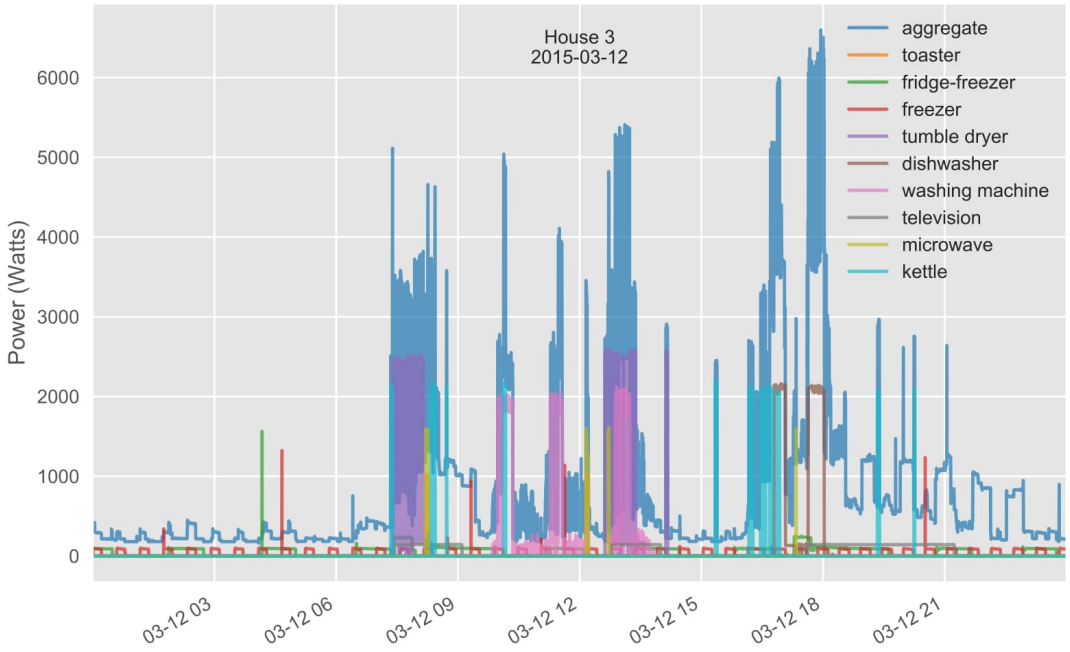

Figure 1.1: Example of a day of data.

at identifying appliance signatures in NILM. Since neural networks require a large amount of data for training, the use of the REFIT dataset is crucial due to its diversity of appliance models and cross-appliance usage behavior when compared to similar datasets.

Since the learning problem in this project is unexplored, large aspects of the project—for example, the construction of the modeling data, the neural network architecture designs, and the methods for evaluation—are different from NILM and other prior research. Therefore, this project explores how well a reasonable set of models perform on this learning problem rather than squeezing out performance gains through optimization.

1.2 Objectives

There will be two **primary objectives** for this project, which have direct relevance to the application domain:

1. Determine whether neural networks can, in principle, predict energy and the number of activations given the aggregate signal of a day of data. This is achieved by evaluating the models on houses that *were* seen during the training and validation process, but for days that were *not* seen.

2. Assess how well the models generalize. This is achieved by evaluating the models on houses that *were not* seen during training and validation. This is a more difficult problem, but also a more important one since it explores how well the models perform when used within the context of the application domain.

Additionally, there will be three **secondary objectives**:

1. Explore whether models that predict a target variable for *all* target appliances (as opposed to just one) perform comparably to models that are trained specifically for certain appliances.[4]

2. Determine to what degree the data augmentation process, which is designed to increase the size of the training dataset for the neural networks, improves generalization results.

3. Explore the results in a way that demystifies the neural network performance beyond the headline evaluation metrics. This means exploring differences in performance between houses that are seen and unseen during training and validation, performing diagnostics of the predictions, and visualizing the inner workings of the neural networks to determine what the models are "looking at" when making predictions.

1.3 Achieved results

We find that neural networks are in principle capable of predicting summarized electricity usage information. More importantly, nine of the twelve appliance models generalize to houses not seen during training, performing better than the baseline. The data augmentation process can be credited in part for the generalization results.

The models perform best on appliances with complex signatures, like the washing machine and the dishwasher. These results are arguably strong enough to provide directly to consumers. The fridge model, on the other hand, generalizes poorly. The multi-target model also does not generalize as well as the single-target models.

[4]The idea exploited by these models, called multi-task learning, is discussed in Section 2.5.

1.4 Dissertation outline

Chapter 1 (this chapter) provided an overview of the motivation, objectives and results of the project. Chapter 2 discusses prior research that is relevant to this project, which is mostly research related to neural networks and NILM. Chapter 3 describes the methods used in the project, including a description and exploration of the REFIT data set, the data cleaning process, the creation of the input data and targets, the selection process for the network architectures, and the results of the architecture selection process. Chapter 4 evaluates the models chosen in the selection process to determine whether the objectives of the project have been met. Chapter 5 summarizes the findings of the paper, describes the limitations of the methods, and suggests paths for further research.

Chapter 2

Background

2.1 Neural networks

In recent years, the revival of the neural network has revolutionized fields such as image recognition, speech recognition, natural language processing, and other areas of machine learning. A neural network—sometimes referred to as an artificial neural network (ANN) or a multilayer perceptron (MLP)—is a powerful supervised machine learning model that has the ability to represent complex nonlinear relationships in the input data. The model was originally developed to model the biological brain, but has since diverged to be optimized for classification and regression tasks.

The basic unit of the neural network is the neuron, although it is often simply called it a "unit." A unit takes a weighted average of input values, applies a nonlinear "activation function," and returns a scalar value (Figure 2.1). The classic activation function is the sigmoid, defined as $f(x) = 1/(1+e^{-t})$, which compresses values to be between zero and one [1].

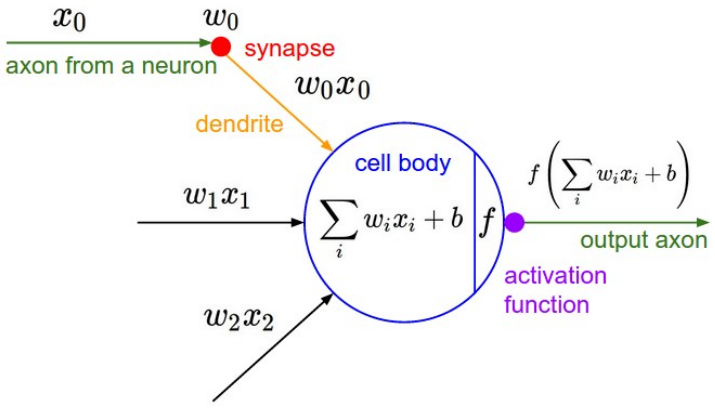

Figure 2.1: A unit of a neural network. Image from [1].

Neural networks are often comprised of layers, which are themselves comprised of units (Figure 2.2). Layers between the input layer and the output layer are called hidden layers. When all the units of a layer are connected to all units of the previous layer, the layer is called "dense" or "fully connected." Neural networks with at least one hidden layer with a nonlinear activation function are universal approximators, meaning that they can, in principle, approximate any continuous function of arbitrary complexity. In practice, more layers often result in more flexibility and representational power, allowing the network to represent increasingly abstract relationships in the data [1].

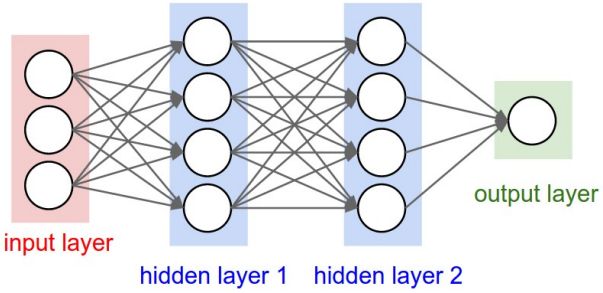

Figure 2.2: Basic neural network architecture with two hidden layers, from [1].

For a neural network to learn, the practitioner must define a loss function to evaluate the performance of the network's predictions. During training, the neural network iteratively searches for a set of weights that minimize this loss function through a gradient-based optimization algorithm such as gradient descent. A process called backpropagation is used to determine the error contribution of each weight and therefore how much each weight should change in each iteration of the optimization.

2.2 Convolutional neural networks

The field of image recognition has seen particular gains from a type of neural network called a convolutional neural network (CNN). A CNN is different from the usual neural network in that it assumes local correlation in input features such as the pixel values in image data. A CNN works by sliding small feature detectors—called "kernels" or "filters"—over the image, identifying pixel patterns such as edges and curves. The outputs of these kernels create "feature maps," which are usually used as input values in the next hidden layer (Figure 2.3). A different set of kernels can then use these feature maps to identify more abstract patterns, such as textures. With enough hidden layers, a CNN can represent complex abstract patterns such as human faces. Notably,

the features identified by the kernels are learned through training, not hand-crafted. Since each kernel has a single set of weights, CNNs have a much smaller number of trainable parameters than would otherwise be the case [12].

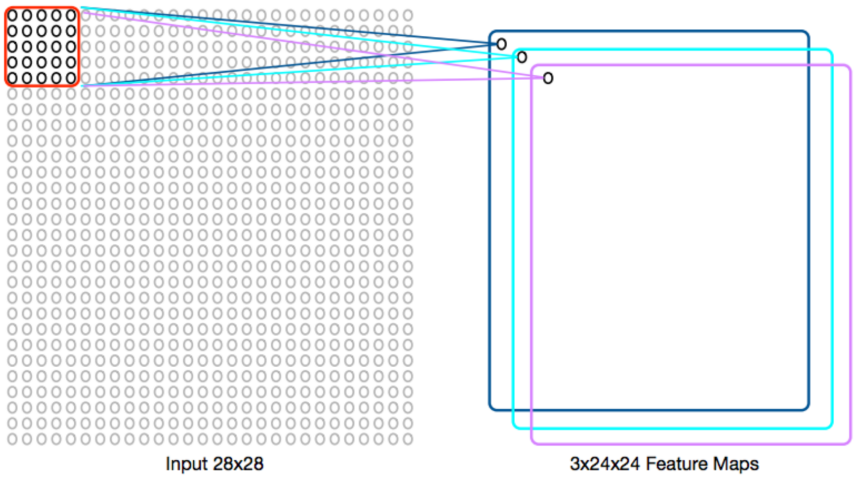

Figure 2.3: Three 5×5 kernels (red) start at the top left of the 28×28 set of pixels, calculating the top-left value in three feature maps. These red kernels then slide over columns to define the first row in the feature maps. Then the process is repeated for the remaining rows. Image from [2].

In order to reduce the dimensionality of the network and to reduce overfitting, pooling layers are commonly placed between convolutional layers every so often (Figure 2.4). The most common type of pooling layer is max pooling, which takes the maximum activation value over some area in a feature map (Figure 2.5). At the end of the network, the final convolutional layer is often "flattened"—that is, rolled out into a 1D vector—so that the final feature maps can be processed by dense layers.

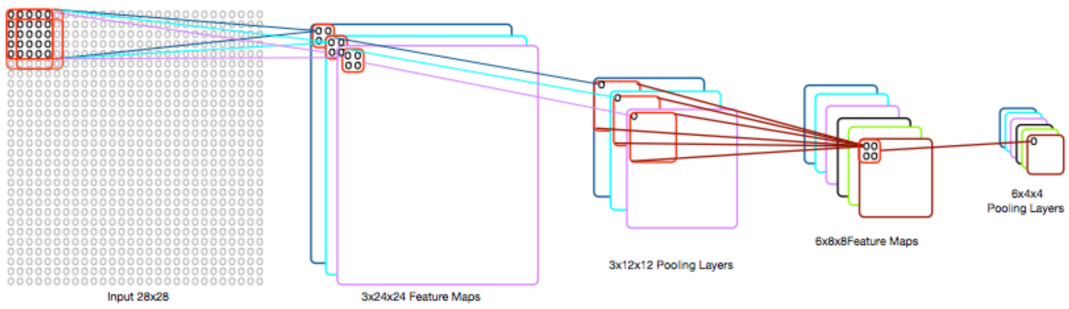

Figure 2.4: Stacked convolutional and pooling layers. Image from [2].

When designing a new CNN architecture, practitioners often take inspiration from state-of-the-art neural networks. One popular state-of-the-art CNN is called VGGNet [13],

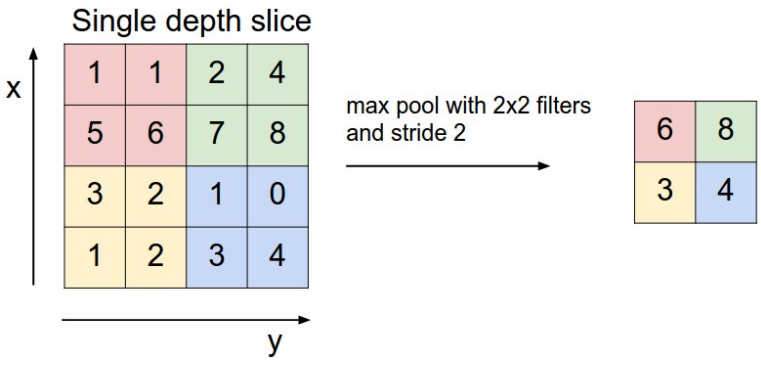

Figure 2.5: Effect of max pooling on a feature map. Image from [1].

which was the runner-up in ILSVRC 2014, a well-known image recognition competition. This network is popular because of its relatively straightforward architecture.[1]

Time series often exhibit autocorrelation, where each timestep is correlated with the previous timesteps in the series. Because autocorrelation is a type of local correlation, time series can be modeled as if they were 1D images, allowing researchers to adapt CNNs for time series classification problems. Instead of edges, the kernels might learn to identify spikes or dips. And instead of human faces, the network might learn to identify heartbeat patterns or household appliance signatures [14, 15, 16, 5, 6].[2]

2.3 Counting with neural networks

The prediction of the number of activations is, at its core, a counting problem.[3] Counting is most explored in computer vision, where the methods often require: (1) handcrafted object detectors; and (2) images with many instances of the desired objects (often hundreds), allowing false positives and false negatives to cancel for the estimated statistics [19]. Unfortunately, most of our appliances do not have anything close to hundreds of activations per day, and we generally want to avoid hand-crafting features.

Research has shown that CNNs can predict the number of objects in an image using

[1] Other state-of-the-art networks often use exotic architecture elements, such as ResNet's heavy use of skip connections [1].

[2] This neural network approach is preferred to using k-nearest neighbor classification (k-NN) with dynamic time warping (DTW) to identify appliance signatures. k-NN with DTW works by stretching or compressing multiple time series so that they align, and then performing k-NN (usually 1-NN) to find the labeled time series that has the most similar pattern [17, 18]. It is unclear how to implement this method for this project given that the input data is aggregated, making matching to labels ineffective.

[3] The prediction of energy is more akin to integration. However, energy and the number of activations are highly correlated, suggesting that neural networks may learn similar features to predict both. See Section 3.4.2.

supervised learning [21]. More generally, neuroscience research has shown that neural networks can learn "visual numerosity"—or the capacity to estimate the number of objects seen. Networks from this research are trained using unsupervised learning (specifically, generative models), but the hidden layer activations can be used to classify images as having more or fewer target objects than some specified number. These models are robust to differences in cumulative area, density and features between the target objects [20].

2.4 Non-intrusive load monitoring

As mentioned earlier, non-intrusive load monitoring (NILM) is the use of software to disaggregate the aggregate signal from a smart meter into the signals of the appliances.[4] In NILM, five appliances are typically modeled: the fridge, kettle, washing machine, dishwasher, and microwave. These appliances are chosen for two reasons: (1) they consume a high amount of energy, meaning energy savings through NILM can be more effective; and (2) they provide more data relative to other appliances since they are common and used relatively frequently.

The additive factorial hidden Markov model (AFHMM) with latent Bayesian melding (LBM) is a state-of-the-art NILM model. At its core, it is a type of hidden Markov model (HMM), which is a model that assumes probabilistic transitions between states that only depend on the current state. The "additive factorial" part of the model means that multiple HMMs are trained in parallel, and the observed output of the model is the sum of the outputs of these HMMs [22]. This is effective for NILM since the aggregate signal is the sum of other signals. The LBM part of the model introduces priors as constraints into the standard AFHMM model, significantly improving results, reducing error by 50% in some cases [11]. Currently national averages are used as priors, but more accurate priors could lead to improved results.

The most effective NILM neural network model is PointNet, which is a CNN that predicts the power value of one appliance at one point in time given a window of aggregate power data [5]. Although NILM is a different problem than summarization, we can find inspiration in PointNet when designing our networks.

[4] Mathematically, the problem can be defined as $p_{\text{agg}}(t) = \sum_{i=1}^{N} p_i(t)$, where $p_{\text{agg}}(t)$ is the aggregate signal at time t and $p_i(t)$ is the signal of an appliance. The goal of NILM would be to find one or more $p_i(t)$ given only $p_{\text{agg}}(t)$, for all values of t.

2.5 Multi-task learning

Multi-task learning is training on multiple related tasks simultaneously using a shared model representation (e.g., the same hidden layers of a neural network), sometimes when only one task is of interest. The error signal of one task serves as a "hint" for the other tasks, sometimes boosting generalization performance [23]. Multi-task learning is common in image recognition, where neural networks often have multiple output classes and each class is a target of interest. This is the extreme case of "hard" parameter sharing since the entire internal representation of the model is shared.

Chapter 3

Methods

3.1 Resources and tools

This project is implemented in Python 2.7. The numpy and pandas Python libraries are used heavily throughout the project for their convenient scientific computing operations and data structures. Scikit-learn, the most popular Python machine learning library, is used for preprocessing. Keras, a high-level neural networks API [24], is used for all neural network modeling, with TensorFlow as the back-end.[1] Matplotlib and seaborn are used for data visualization. Jupyter notebooks are used for prototyping, data exploration and executing code for data visualization. Git is used for version control.

Although convolutional neural networks are computationally expensive to train, the models in this project are considerably less expensive than timepoint-by-timepoint disaggregation. Therefore it is feasible to do all training on a personal laptop using CPUs.

NILMTK, an open-source toolkit for training and evaluating NILM models, is not used since the data for this project has to be preprocessed and modeled in a way that is unlike NILM applications. However, this project does use the NILMTK `Electric.get_activations` function, which was simply copied from GitHub and adapted to fit within the project framework.

[1] Keras is chosen over using TensorFlow directly due to its simplified syntax that better allowed for dynamic architecture creation.

3.2 Overview of the dataset

There are many datasets with smart meter and appliance-level power data that are popular within the NILM community, including UK-DALE [25], HES [26], BLUED [27], and REDD [28]. However, REFIT (Personalised Retrofit Decision Support Tools for UK Homes Using Smart Home Technology) stands out from the others in that it has the combination of a long study duration (21 months; see Figure 3.1), more than just a few houses (20), a large number of appliances recorded with IAMs (9), and a high sampling rate (6–8 seconds).[2] In comparison, BLUED and UK-DALE have just 1 and 5 houses (respectively), HES a 10 minute sampling rate, and REDD a study duration of only 3–19 days [28].

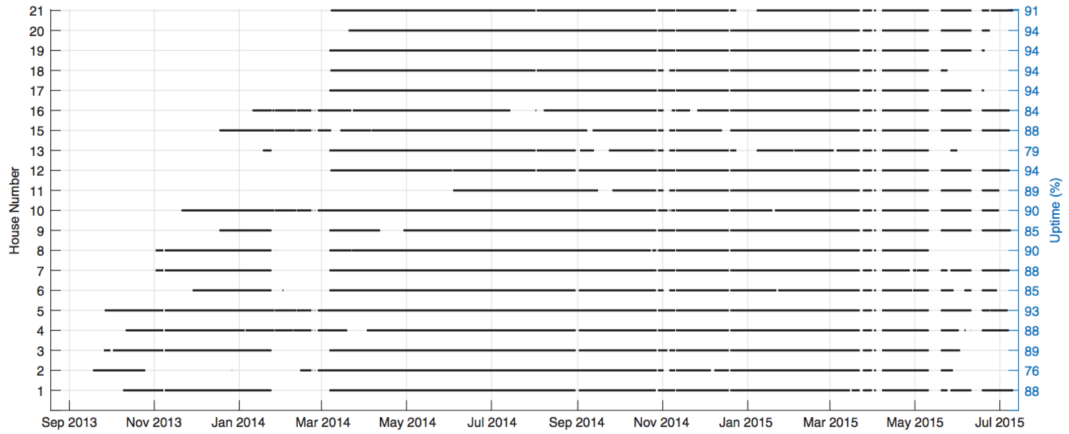

Figure 3.1: Data availability for each house. Figure from [3].

There are two versions of the REFIT dataset: raw and cleaned. This project uses the cleaned data, which corrects some appliance labeling errors in the raw data,[3] forward-fills null values, and removes spikes of over 4000 watts [29]. This cleaned dataset is stored as a set of CSVs, one for each house. Each file has 7–8 million rows and is roughly 300–400 MB in size. There are two datetime columns (one for a datetime string and the other for a Unix datetime), one column for the aggregate power signal, and nine columns for the power signals of the appliances. There is an additional column called "Issues" which has a binary indicator for whether the sum of the recorded appliances exceeds the aggregate signal.

[2] Although the mean and median sampling rates are around 6–8 seconds, there is great variability in the difference between timestamps. The 10th percentile in the distribution of this difference is 2 seconds and the 90th percentile is 13 seconds. This is because the IAMs only record data when there is a change in load [3], which is discussed in Section 3.3.2.

[3] For example, if the appliance plugged into an IAM changed at some point in the study, then this data would account it for by changing the labels.

Of the 20 houses, 18 have at least one fridge (or fridge-freezer), 16 have a kettle, 19 have at least one washing machine, 15 have a dishwasher, and 16 have a microwave (Table 3.1).

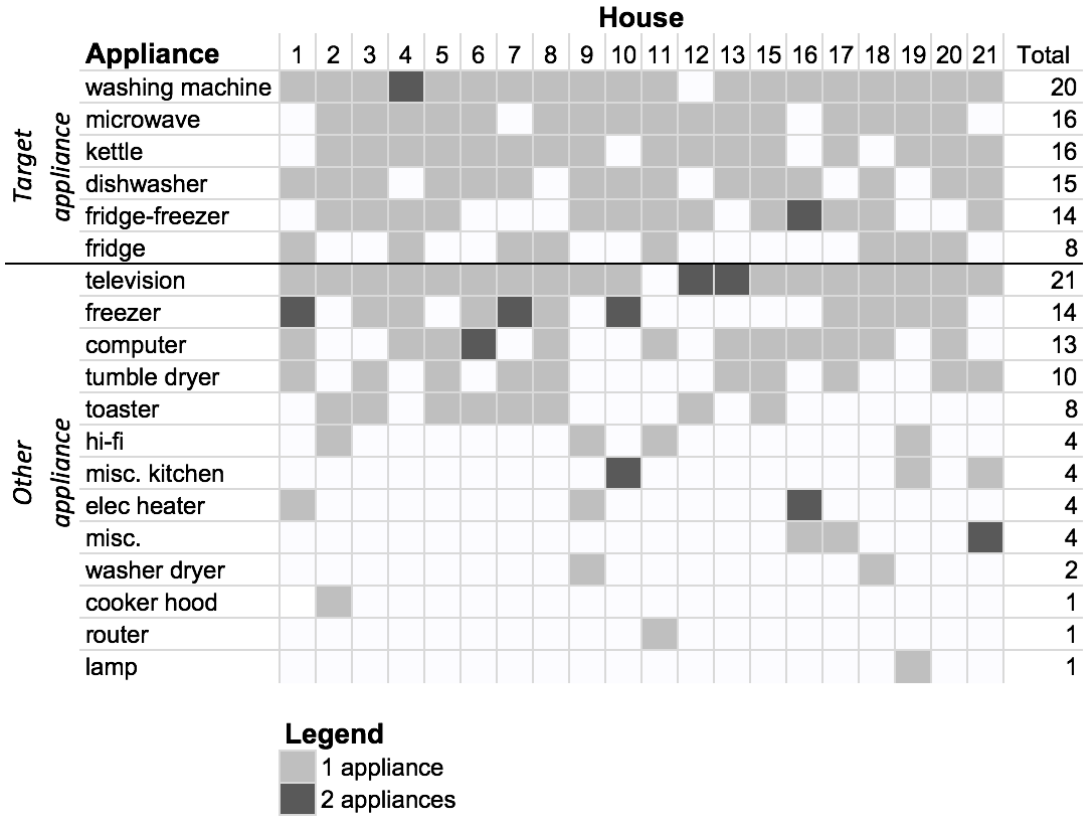

Table 3.1: Monitored appliances in each house. There is no House 14. Table data comes from [3].

3.3 Calculation of target variables

3.3.1 Number of activations

The activations—that is, the periods when an appliance is being used—can be extracted from an appliance's power series using code adapted from the NILMTK Python package [6]. The process is described below:

> The arguments we passed to `get_activations()` for each appliance are shown in Table 4 [included below as Table 3.2]. On simple appliances such as toasters, we extract activations by finding strictly consecutive samples above some threshold power. We then throw away any activations shorter

than some threshold duration (to ignore spurious spikes). For more complex appliances such as washing machines whose power demand can drop below threshold for short periods during a cycle, NILMTK ignores short periods of sub-threshold power demand.

Appliance	Max power (watts)	On power threshold (watts)	Min. on duration (secs)	Min. off duration (secs)
Kettle	3100	2000	12	0
Fridge	300	50	60	12
Washing m.	2500	20	1800	160
Microwave	3000	200	12	30
Dish washer	2500	10	1800	1800

Table 3.2: Table from [6] showing arguments passed to NILMTK's `Electric.get_activations()` function.

The function arguments in this project are kept the same as in the original paper (Table 3.2).[4]

3.3.2 Energy

The basic formula for energy is $E = Pt$ where E is energy, P is power and t is time. Since the IAMs only record data when there is a change in load [3], we can assume that the power level at timestamp t_n remains constant between timestamps t_n and t_{n+1}, where n is the index of the recording such that $t_n < t_{n+1}$.[5] We can therefore calculate total energy in our data as

$$E = \sum_{n=0}^{N-1} P_n(t_{n+1} - t_n) \tag{3.1}$$

where E is the total energy used in the day, P_n is the power level at recording n, and N is the total number of recordings in a day. Energy will be reported in kilotwatt-hours (kWhs), which is equivalent to 1,000 watts of power for a period of one hour.

[4]The one exception is the on-power threshold of the kettle, which is reduced from 2,000 to 1,500 watts to capture the kettle activations in House 3.

[5]Here, t is not elapsed time but rather the amount of time elapsed since some fixed datetime in the past. This is a common way of representing timestamps in software, such as with Unix time. Elapsed time can be calculated by taking the difference between timestamps.

3.4 Data exploration

3.4.1 Target appliance signatures

In order to be able to diagnose the performance of our models later in the project, it can be useful to explore the signatures of the target appliances. To start, the fridge's signature is somewhat rectangular and low-power, with a characteristic spike at the start (Figure 3.2). There is also a great deal of variety in both its time duration and energy per activation, with some skewness toward high values (Figure 3.3). The kettle has a rectangular, high-power signature usually lasting for one or two minutes. Prior research has shown that this simple rectangular shape has lent the kettle to easier timepoint-by-timepoint disaggregation than other appliances [5]. The washing machine has a complex, noisy, long-duration signature that alternates between high-power mode and low-power modes. The dishwasher signature is also complex and long-duration, but it has less noisiness than the washing machine and exhibits more switching between high-power and low-power modes. The microwave, like the kettle, is rectangular and also lasts for a few minutes, but it has a duration distribution that is skewed more strongly to the right.

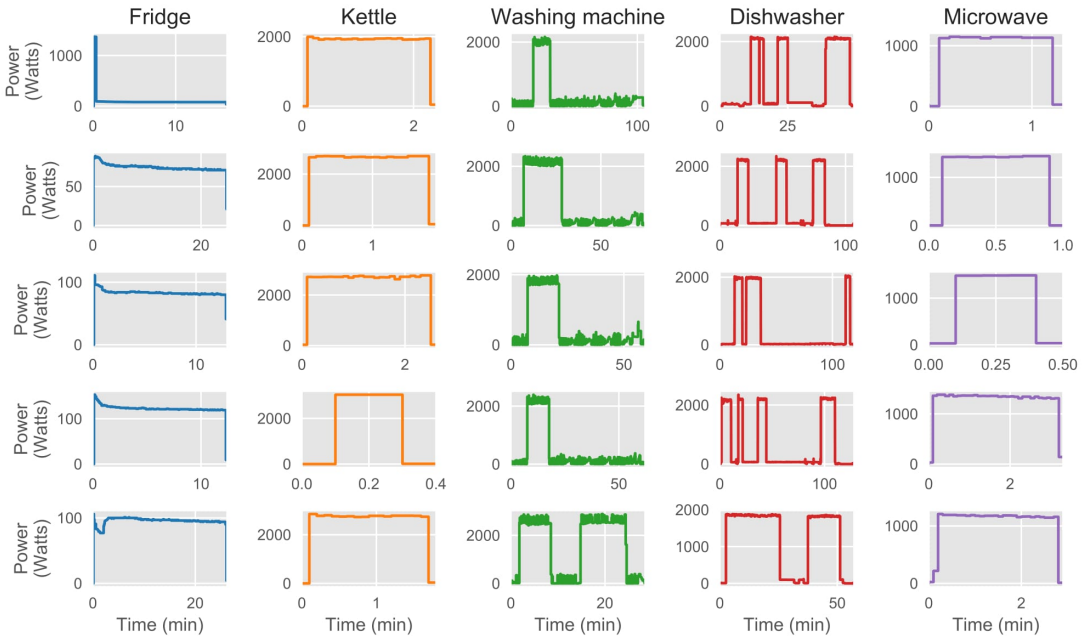

Figure 3.2: Five random signatures for each of the five target appliances for a single activation. Each target appliance is represented as a different color.

For the purpose of this project, it is important to categorize or group appliances

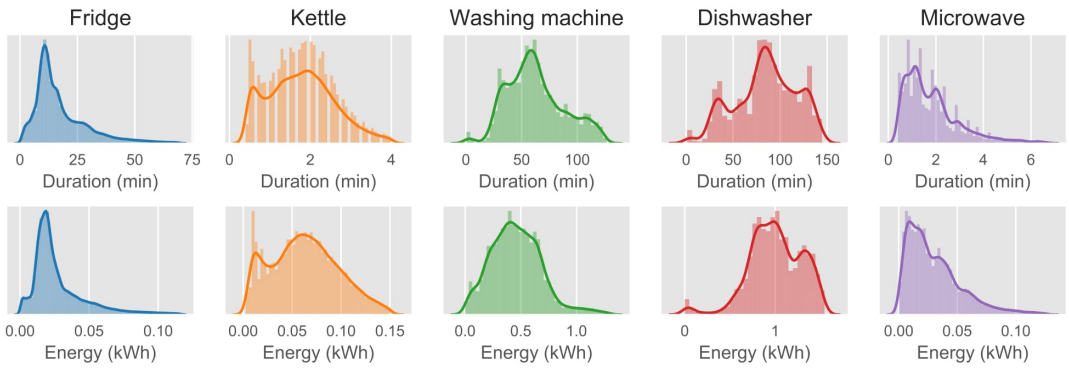

Figure 3.3: Distribution of time duration per activation (first row) and energy per activation (second row) for each target appliance. The largest 2% of values were not plotted.

in a way that is useful for generalization and does not needlessly hinder the model. In particular, if two appliances have similar functions and similar signatures, then we want to consider them to be the same appliance to avoid systematic false positives. Also, requiring a model to distinguish between very similar signatures could force it to over-rely on other house-specific appliance signals, which would hurt generalization.

This grouping problem is hardest for the fridge. There are 7 fridges, 14 fridge-freezers and 1 IAM with both a fridge and a freezer plugged into it.[6] While the details of how and why each of these signatures differ is complicated,[7] inspection of their signals shows that they are similar enough that we can reasonably categorize them as "fridges." A similar decision has to be made for the washer and the washer-dryer, but in this case the two appliances have signatures that are quite different from one another. Therefore we can safely exclude washer-dryers from the category of "washing machine."

Using this appliance classification, we find that the fridge has the largest number of activations across all houses at 204,774, while the kettle has 41,719, the microwave 17,080, the washing machine 6,577, and the dishwasher 4,725.

[6]The house with both a fridge and a freezer plugged into the same IAM is House 19, and the IAM is Appliance 1. The power series has the label "Fridge & Freezer" in the README.

[7]A thorough analysis of the fridge and freezer signatures requires some knowledge of how compressors and fans operate within fridges and fridge-freezers, how many there are in each fridge(-freezer) model, and their power. For example, a fridge-freezer with one compressor often has a simple signature, but a fridge-freezer with two compressors may have a signature comprised of two stacked compressor signatures.

3.4.2 Daily statistics

Since our models will be training and testing on daily data (Section 3.6.2), it is useful to explore energy and activations data summarized at the daily level. In doing so, we see there is a great deal of variety in the amount of energy used by each house (Figure 3.4). For example, House 15 uses a median 5.6 kWh of energy per day, while House 8 uses around three times that much at a median 16.7 kWh per day. Some houses, such as House 11, also exhibit more variance across days than other houses.

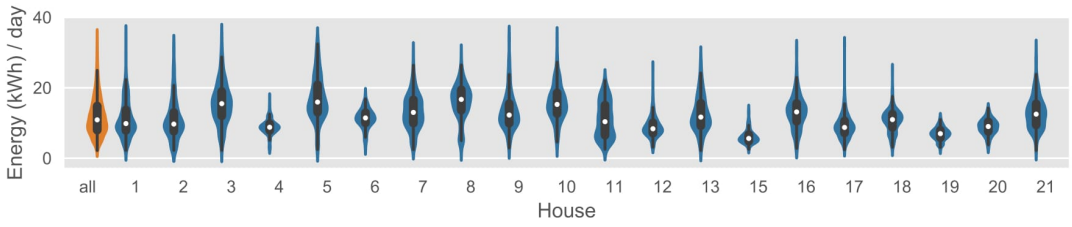

Figure 3.4: Distribution of daily energy used per house as recorded by the aggregate signal. The distribution for all houses combined is highlighted in orange.

Breaking this energy usage down by target appliance (Figure 3.5) lets us compare between-house and within-house variance of the target appliances. We see that the fridge has a great deal of between-house variance, with House 8 using a median 0.2 kWh per day and House 8 using nearly 10 times as much at a median 1.9 kWh per day. The fridge also seems to have the least amount of within-house variance. As a result, the fridge's distribution for *all* houses has much more variance than the distribution for any individual house. The lack of within-house variance makes sense since the fridge is the only target appliance that is activated without the express action of the user, and it is therefore much more invariant to day-to-day changes in behavior.

Figure 3.6 explores the relationship between energy and the number of activations in more detail. As expected, energy is positively correlated with the number of activations for all of the target variables. This is good news: It means that if one target variable is more difficult to predict—likely energy, since it is a more nuanced and complicated calculation than counting—then at the very least our models should be able to get most of the way toward an accurate prediction by predicting the easier target variable times some constant. However, we should hope that our models can learn how to predict each target variable in a way that is not simply a linear transformation of the other.

There are two main sources of variance in energy for a particular number of acti-

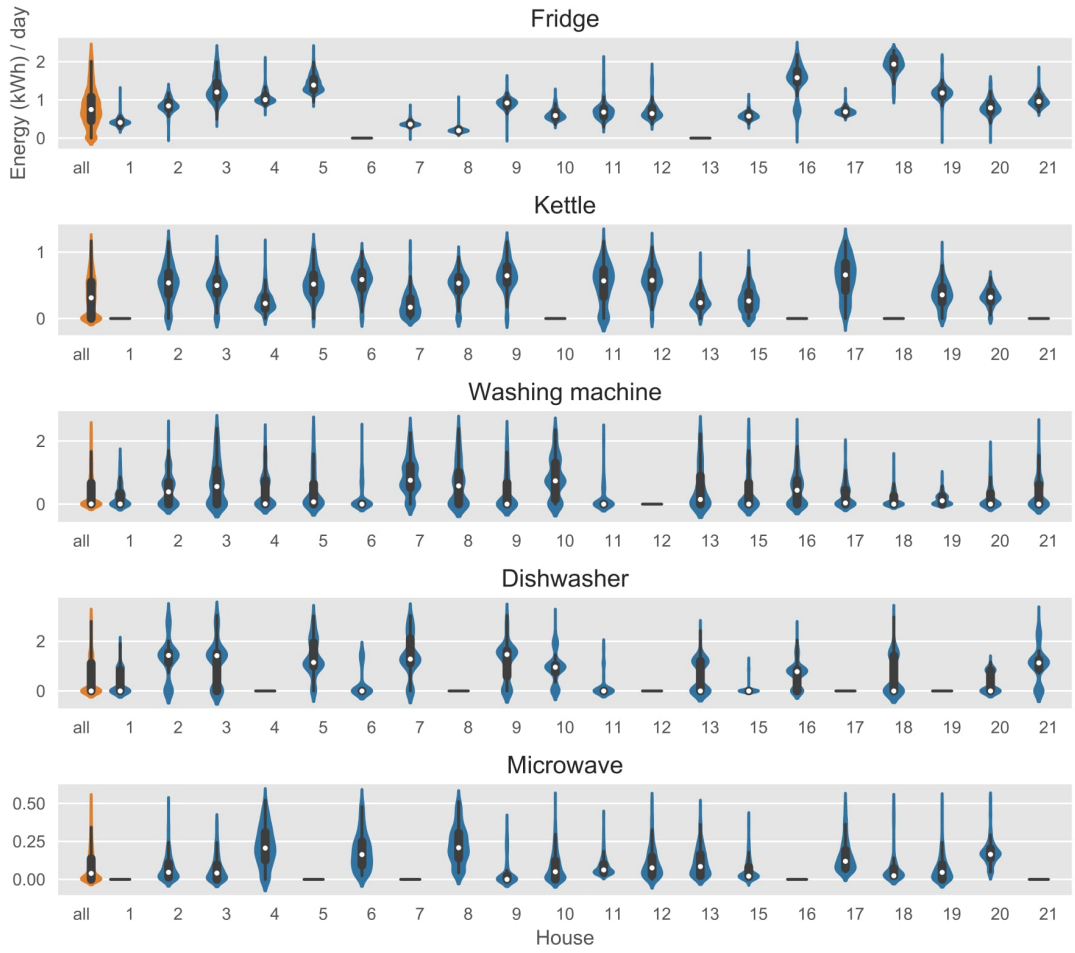

Figure 3.5: Distribution of daily energy used by house, broken down by appliance. The distribution for all houses combined is highlighted in orange. Horizontal lines indicate that the house did not have that appliance hooked up to an IAM (e.g., the fridge for House 6). If a house had more than one of the target appliance hooked up to IAMs (e.g., the two washing machines in House 4), then the daily energies for those appliances were summed.

vations: (1) different appliance models across houses; and (2) different usage behaviors/settings for any one appliance model within a house.[8] To separate these sources of variance, we can calculate the correlations between the two target variables for each house (Figure 3.7).

[8]There are instances where the activation is zero and the energy is positive. In each instance, this could be due to one of two things: (1) `Electric.get_activations` fails to catch the activation; or (2) there is a data error giving positive energy without an accompanying signature that can be caught by `Electric.get_activations`. Given the possibility of these two scenarios, we cannot rely on one metric to correct the other. These errors can also add to the variability of energy in cases where the number of activations is positive.

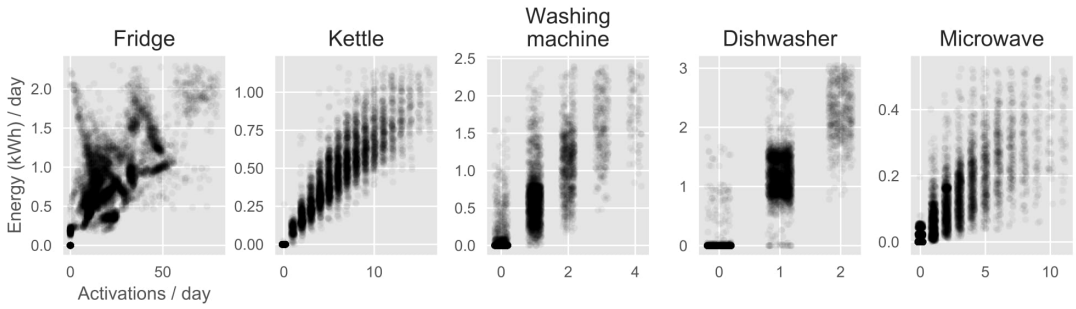

Figure 3.6: Relationship between energy used per day and number of activations per day. The points are jittered along the x-axis to reduce overlap.

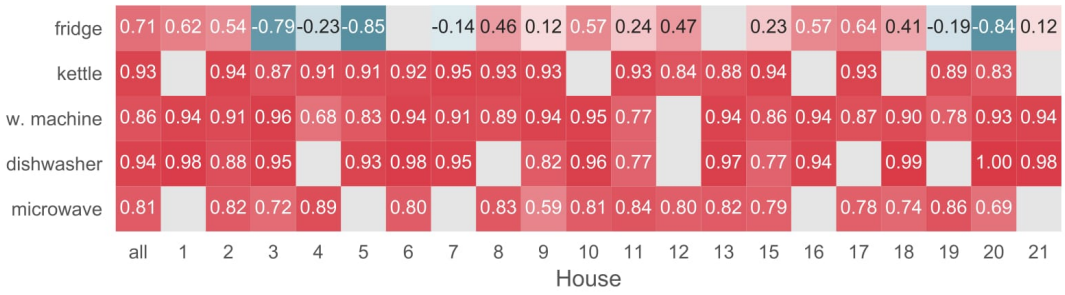

Figure 3.7: Correlation between daily energy and number of activations, by appliance and house. A house without an appliance is represented as a gray square with no correlation number—not to be confused with zero correlation.

We find that, for the most part, the correlations between the target variables are strongly positive within houses, reflecting the positive correlation for all houses together. The exception is the fridge, where the correlation for all houses is higher than the correlation for any individual house. Some of these within-house correlations are actually strongly negative.

To visualize the negative correlation for one house, we can take Figure 3.6 and highlight one of the houses that has a negative correlation—say, House 20. The result is Figure 3.8. As expected, the fridge pattern of House 20 does not reflect the overall pattern. The other appliance patterns, on the other hand, do reflect the overall pattern.

In general, we want our models to use the target appliance signatures to make their predictions. But we also want them to use cross-appliance patterns to the extent that they are generalizable across houses. An example of a potentially useful cross-appliance pattern is the positive correlation between kettle and microwave usage (Figure 3.9). Another example is the correlation between washing machine and dishwasher

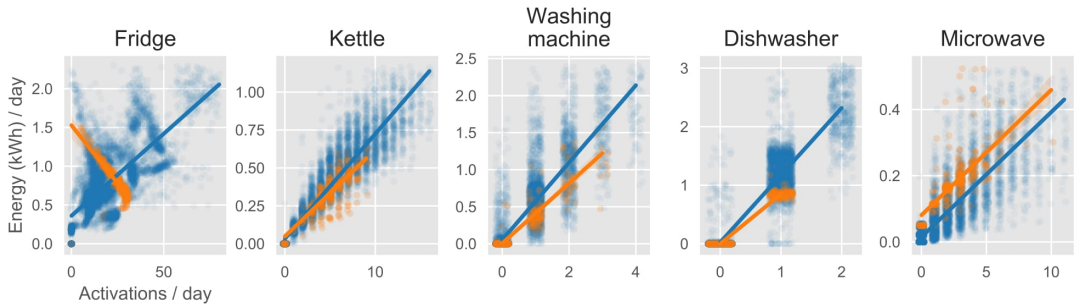

Figure 3.8: Relationship between energy used per day and number of activations per day, with House 20 highlighted in orange. A line of least squares is fitted to emphasize correlation. The points are jittered along the x-axis to reduce overlap.

usage. This agrees with intuition: The first correlation represents days where there is more cooking, and the second represents days where domestic chores are being done.

However, such conclusions must be made with caution, since aggregate patterns may disappear when observing within-house patterns. For example, the fridge activations are strongly negatively correlated with kettle activations. This is clearly a pattern that exists between houses as opposed to within houses. That is, houses with fridges that have many activations per day tend to be houses where the kettle is not used very often—at least in our dataset.

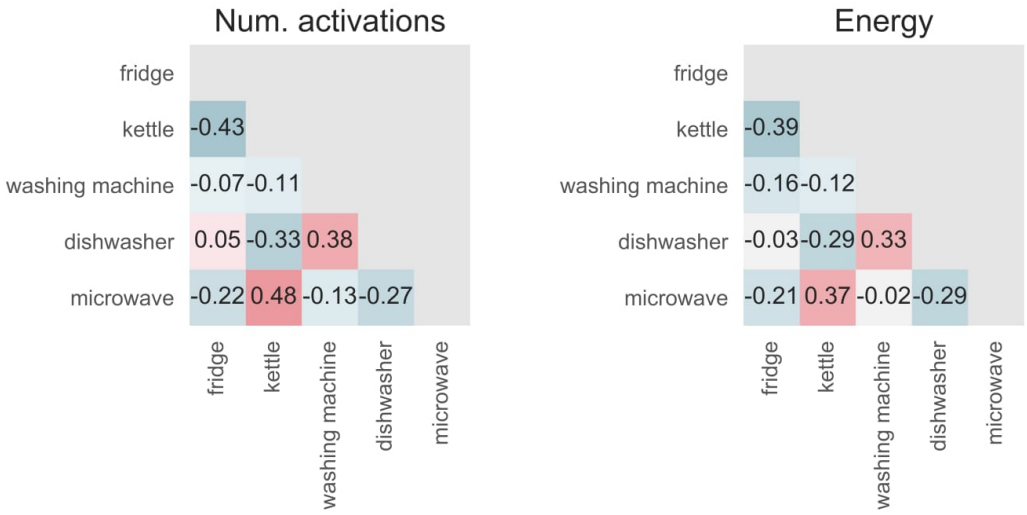

Figure 3.9: Correlation between target appliances for the specified target variable. For example, the correlation between daily number of microwave and kettle activations is 0.48.

Our models may be led astray by such correlations. Appliances with high between-

house variance but low within-house variance—such as the fridge—are more likely to suffer from such issues, since the models will be encouraged simply to predict the average target value for the house using house-specific appliance patterns. We will see how this affects generalization performance in Section 4.

3.5 Data cleaning

3.5.1 Overview of data issues

The data-cleaning process is based on previously known and newly discovered data issues, which include gaps in the data, the effect of solar panels on the aggregate signal, inconsistencies between recordings, and mislabeled data. For a full description of these issues and more, see Appendix A.

3.5.2 Unused data

Some observations are completely discarded. These discarded observations suffer from at least one of the following:

- **Gaps or irregularities in the timestamps**, meaning one or more of the following is true: there are fewer than 500 recordings in a day (which affects 1013 observations, or 9.3% of the total);[9] there is a gap of more than 15 minutes (650, 6.0%); the timestamp range is less than 23.5 hours (422, 3.9%); or the clocks are changed for daylight saving (20, 0.18%).[10]

- **Errors in the appliance series or major errors in the aggregate series**, meaning one or more of the following is true: at least one appliance has a power value that is unchanging and above 25 watts for more than 10% of the day (781, 7.2%); the sum of the appliance series is greater than the aggregate series for more than 10% of the day (246, 2.3%); or there is a negative power values in at least one power series (9, 0.083%).

In total, 2,663 observations (24% of the total) are unused because of these errors. This is an unfortunate amount of data to discard, but pilot tests showed that substantial cleaning is necessary in order to get reasonable modeling results.

[9] The expected number of recordings in a day is around 14,400. See Appendix A.2 and Section 3.6.3.

[10] The daylight saving change is not an error, but removing these observations simplifies the data creation process.

3.5.3 Data only used to create synthetic training data

There are cases where only the aggregate signal has issues but the appliance signals are otherwise clean. It would be wasteful to throw away this data altogether, so they are used for the creation of the synthetic data. This includes the following cases:

- **Solar panel interference.** Three houses (1, 11 and 21) suffer solar panel interference, causing a large amount of noise in the aggregate signal. This represents roughly 15% of the data. See Appendix A.1 for a description of this problem.
- **Low correlation between IAMs and aggregate.** Instances where the correlation is below 0.1 captures 15 observations, or 0.21% of the total.
- **Strings of large, repeating *aggregate* power values.** Instances where the aggregate series has a power value that is unchanging and above 25 watts for more than 10% of the day represents 86 observations, or 1.2% of the total.

3.6 Dataset creation

3.6.1 Data split: training, validation and testing

It is standard in machine learning to split data into three sets: one to train the model (the training set), another to help choose between models (the validation set), and another to report results (the test set). It is common to simply take all available observations and split them into three sets, usually 60% for training, 20% for validation and 20% for testing [30]. However, in this project we want to have two test sets: one containing houses that the model *has* seen in training and validation (to explore the difficulty of the learning problem in principle), and one with houses the model *has not* seen (to test generalization performance). Therefore, a simple split of the dataset would not work since we have two test sets and need to be especially careful about information leakage.

To start, we hold out a set of houses explicitly until test time. Additionally, since we want to validate our models in a way that encourages generalization, we *also* hold out a *different* set of houses that we will use for validation. Three houses in each holdout set should (hopefully) provide enough diversity of appliance models. Any fewer might make the validation process and the test results very sensitive to the holdout houses, while any more might take away too much data from the training set.

We want each house in these two holdout sets to have at least one of each of the five target appliances. The only houses that meet this criterion are Houses 2, 3, 5, 9, 15, and 20, which conveniently gives us exactly the number of houses we need. There is little reason to choose some houses for one holdout set and other houses for another holdout set, so we will randomly assign Houses 3, 9 and 20 to the validation holdout set and 2, 5 and 15 to the test holdout set. These houses will be referred to throughout this paper as "unseen" houses—that is, houses that were not seen by the model during training. The unseen *validation* houses are represented as purple in Figure 3.10, while unseen *test* houses are represented as green.

In addition to testing separately on "seen" houses, we will also include some diversity of appliance signals to the validation set by adding some "seen" house data, which can help reduce sensitivity to holdout houses. This seen house data for the validation and test sets are from *dates* that are unseen during the training process.

To create this "seen" validation and test data, we simply split the data for houses that we are *not* holding out for validation and testing into two groups: 70% for training and 30% for the combined "seen" testing and validation sets (later split in half so that each has 15%). This split between the training data and the seen validation/test data is performed randomly across all dates to minimize the influence of seasonality—which is preferred to simply splitting the dataset into three parts using date ranges. In Figure 3.10, the training data (70%) is represented as blue, and the collective validation and testing data for seen houses (30%) is represented as yellow.[11]. Since the largest source of variation in performance is likely across houses, the split from the combined 30% into the 15% for the seen validation set and 15% for the seen test set is stratified by house, meaning each house is represented roughly evenly in each group.

Although this validation method results in models that generalize well to unseen houses (Section 4), it would have been more effective (and elegant) to validate through k-fold cross-validation (CV), where each fold is the data of one or more houses.[12] This way we would have been able to validate for generalization on *all* houses (except for those held out for testing) and also not have to use seen days to provide diversity of signatures. However, if we were to do 5-fold CV (that is, validation five times with five different holdout sets), then the number of models that need to be trained would

[11] One color is used for the collective 30% both because it creates simplicity in the graph and because inspecting the breakdown between the two is not very informative.

[12] In k-fold CV, the dataset is split into k "folds," or subsets of the original data. In one iteration, the model is trained on $k-1$ folds and validated on the remaining fold. This procedure is repeated until every fold is used as the validation set. The validation error of each fold is then averaged.

increase by a factor of five. This is not feasible since there are already a large number of neural networks that need to be trained (Section 3.7). We can rely on other methods to encourage generalization, discussed later in this paper.[13]

The individual appliance signals of the training data—but *not* the validation or test data—are used to create synthetic data. This prevents information from the validation or test data from leaking into the training process. Additionally, the individual appliance signals of some observations with poor-quality aggregate signals (which includes houses with solar interference) are used to create synthetic training data, since in this case we only care about the quality of the appliance signals (orange in Figure 3.10).

In total, 3,146 examples—or 45% of all cleaned, real data—are in the "real" part of the training set (as opposed to the synthetic part), 686 (10%) are in the validation set for seen houses, 686 (10%) are in the test set for seen houses, 1,159 (17%) are in the validation set for unseen houses, and 1,301 (19%) are in the test set for unseen houses. There are also 39,800 synthetic training examples, increasing the size of the training set by a factor of nearly 14.[14]

When training a model for NILM, care is taken to balance "positive" and "negative" examples (i.e., windows where the target appliance is activated and when it is not) by sampling them explicitly [6]. This is because most target appliances are activated infrequently, and therefore the model would see more negative examples than positive ones if sampling naively over time. However, for our summarization task, balancing is not necessary since there are not so many examples where the target appliances have zero energy or activations that we would have to worry about this issue.

[13] However, these methods will still rely on a set of holdout houses for validation as opposed to using CV.

[14] This is still not many training examples for a neural network, but one consolation is that an observation often has multiple activations—so any model often updates its weights using information from multiple signatures per example. This is assuming the model learns, at least in part, by using the signatures of the target appliance.

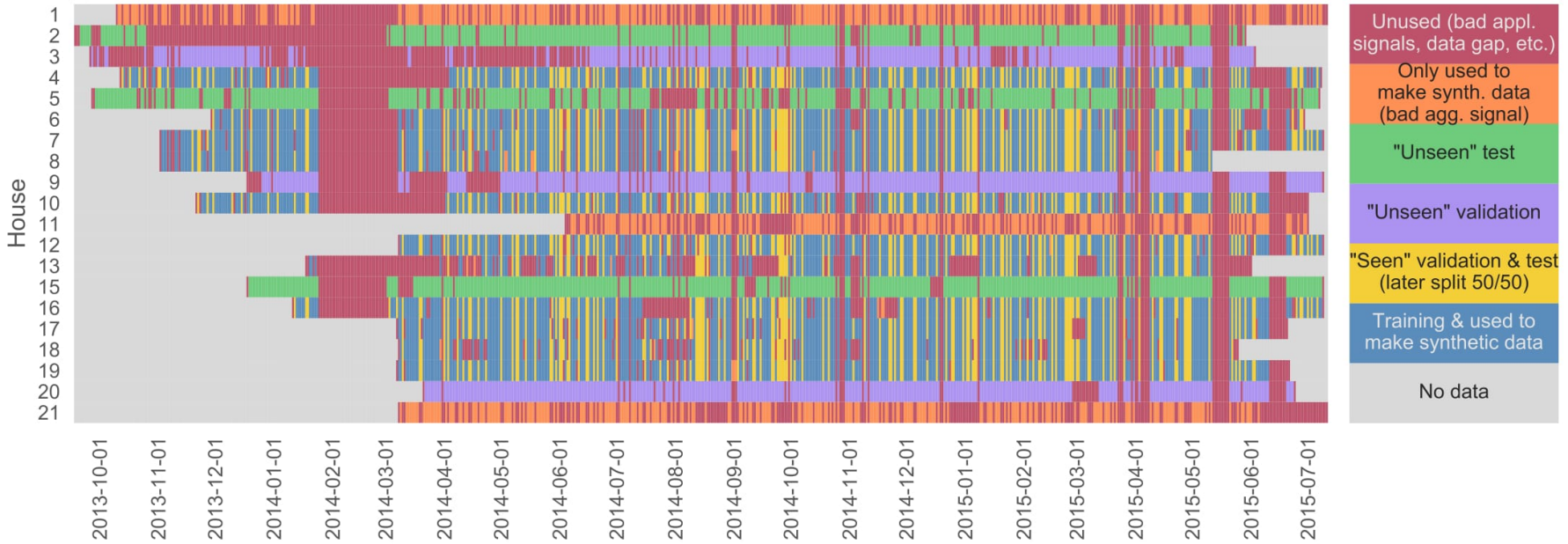

Figure 3.10: Split between training, validation and test datasets. Real training data is in blue, while the appliance signals from the blue and orange data are used to make the synthetic data. The horizontal orange stripes represent the houses whose aggregate signals are made unusable due to solar panel interference (1, 11 and 21). The horizontal purple stripes are for the houses that are held out for validation (3, 9 and 20), while the green stripes are held out for testing (2, 5 and 15) (both “unseen”). The thin vertical yellow stripes indicate the dates that are held out for validation and testing (“seen”). The red data is not used at all due to errors in the appliance signal or gaps in the data, such as the outages in February 2014.

3.6.2 Structure of the datasets

The goal is to create a feature matrix $X_{N\times D}$ and a target matrix $Y_{N\times K}$ where N is the number of aggregate signals, D the number of features (which is equivalent to the number of timesteps in a day), and K the number of targets for a particular target variable. K is equal to 1 for the single-target models and 5 for the multi-target models.

The window is set to be a 24-hour day, midnight-to-midnight, since it is a natural unit of time for generating summary statistics.[15] Another option for the window length is a full week, but there are several downsides to this: (1) more recordings per example when there are already quite a lot; (2) fewer independent observations (by a factor of seven); and (3) potentially poorer performance for short-duration appliances [6]. Using a day of data also does not result in a loss of granularity: If need be, the daily summary statistics can be aggregated into weekly statistics.[16]

3.6.3 Timestep standardization

Since the number of recorded measurements varies by day (Appendix A.2), the number of timesteps in each aggregate series will need to be standardized so that they can be used as model input. To do so, we create a standardized series that will assume consistent 6-second sampling, and then align the aggregate power series to this standardized timestamp series. For a day of interest, the standardized series would have the times 00:00:06, 00:00:12, 00:00:18, ..., 23:59:56, and 00:00:00 (following day) in `hh:mm:ss` format.

In order to align an unaligned power series $\boldsymbol{p}$ with the standardized timestamp series $\tilde{\boldsymbol{t}}$, we take $\boldsymbol{p}$'s unaligned timestamp series $\boldsymbol{t}$ and find indices $\boldsymbol{i}$ where each $\boldsymbol{i}_j$ gives the maximum value of $\boldsymbol{t}_{\boldsymbol{i}_j}$ such that $\boldsymbol{t}_{\boldsymbol{i}_j} \leq \tilde{\boldsymbol{t}}_j$ for all j.[17] Indices $\boldsymbol{i}$ can then be used to find the power value $\boldsymbol{p}_{\boldsymbol{i}_j}$ that is associated with standardized timestamp $\tilde{\boldsymbol{t}}_j$. The

[15] Some NILM researchers find it useful to train on randomly selected time windows [6]. While this may increase robustness for NILM modeling by serving as a form of data augmentation, it would mean at least some loss of time-of-day information for our problem. This may hurt model performance because some time-based usage patterns may be generalizable across households. For example, if many households use washing machines in the afternoon, then it could be beneficial to allow our model to use this information for learning. This project introduces robustness through other means, namely through other forms of data augmentation and the use of regularization in the architecture of the neural network models.

[16] One potential downside to using a day instead of a week as the input window is that appliance signatures are more likely to be split at the dayline. However, an inspection of the appliance data shows that long-duration appliances tend not to be used around midnight. It is also conceivable that the models would be able to learn using partial signatures, but there are likely too few samples for this to happen.

[17] The size of $\boldsymbol{i}$ is equal to the size of $\tilde{\boldsymbol{t}}$, but it is generally not equal to the size of $\boldsymbol{t}$.

procedure to find $\boldsymbol{i}$ is described in Algorithm 2 in Appendix B. We take the maximum power value *before* the standardized timestamp because IAMs only record data when there is a change in load, which is a fact was also used in the calculation of energy in Equation 3.1.[18]

After a daily power series is standardized, it has 14,400 data points, which is the number of seconds in a day divided by 6. This represents the number of timesteps D of our feature matrix X from Section 3.6.2.

Given this alignment process, functions for calculating target variables, a system for loading power series, and cleaned data, it is straightforward to create the real data that will be used in the project (as opposed to the synthetic training data discussed in Section 3.6.4). The process is simply a double `for` loop over all houses and dates for which we have clean data, and then a third loop over the appliances to calculate the target variables (Algorithm 3 in Appendix B).

3.6.4 Data augmentation

Neural networks require a large amount of training data since there are so many trainable parameters. To increase the amount of training data, researchers commonly use a technique called data augmentation, which is the expansion of the training set by applying realistic distortions to the input data [31, 32]. In image recognition, this is often done by blurring, rotating or cropping the images. Data augmentation also has the benefit of adding robustness to the model by preventing it from memorizing specific observations.

In NILM, it is possible to perform data augmentation by adding appliance signals to synthesize aggregate signals. This is the approach in [6], where each target appliance is included in the synthetic aggregate with 50% probability and each non-target appliance—called a "distractor" appliance—is included with 25% probability.[19] The models are then trained on data that is half real, half synthetic. The authors concede that this method "ignores a lot of structure that appears in real aggregate data," and

[18] Another option would have simply extended or reduced the number of points in the power series by evenly duplicating or deleting observations throughout the series. This would also result in less information loss. However, this does not standardize the differences between consecutive timestamps, which is important in principle since energy is the product of power and time. Arbitrarily stretching time would, in turn, arbitrarily stretch energy values. Zero-padding is yet another option which would also result in less information loss, but it also does not standardize timestamp differences—and it is unclear how the neural network model would respond to so many repeated zeros. Zero-padding is routinely used in convolutional layers in neural networks in order to preserve dimensionality between certain layers, but the networks in these cases does not observe block after block of zero matrices.

[19] The appliance signals are chosen randomly from a bank and could be from any house or date.

that "a more realistic simulator might increase the performance of deep neural nets on energy disaggregation" [6].

This project aims to build a more realistic simulator for three main reasons:

1. We have fewer training observations given that we are summarizing at the daily level, so the quality of the augmented data is important.

2. Since we are looking at a full day of data, there are more cross-appliance usage patterns that the models can learn when compared to NILM.

3. While naive simulator may improve generalization and robustness to unseen data points, a careful implementation can also improve the model's ability to generalize in a way that is specific to our application domain.

This simulator (Algorithm 4 in Appendix B) is different from the naive method in one major way, which is that it uses the appliance signals of specific house and date combinations as its starting point as opposed to drawing randomly from a bank of appliance signals that can come from any house and date. In doing so, it preserves some of the cross-appliance usage patterns while also adding enough cross-house randomness to encourage generalization to unseen houses.

For each house and date combination, there are two major steps:

1. A loop over the **target appliances** in that house, where there is a 50% probability of either adding the appliance signal to the synthetic aggregate or swapping it with the same-appliance signal of a random house and date. This is loop affects the first five appliances of the example house in Figure 3.11 and is represented by Line 9 of Algorithm 4. If there are multiple power series of the same target appliance type, then each is addressed separately.[20]

2. A loop over the **distractor appliances** of that house, where there is a 50% probability of either adding the appliance to the synthetic aggregate or excluding it. This affects the last four appliances in Figure 3.11 and is represented by Line 26 of Algorithm 4.

During this process, the appropriate target metrics are computed based on the chosen series and added to the matrix of targets (Line 18 of Algorithm 4).[21]

[20] This is represented by Line 12 of Algorithm 4, which loops once for each washing machine in Figure 3.11 and zero times for the microwave.

[21] One alternative approach to data augmentation would be to start with the real aggregate signal of

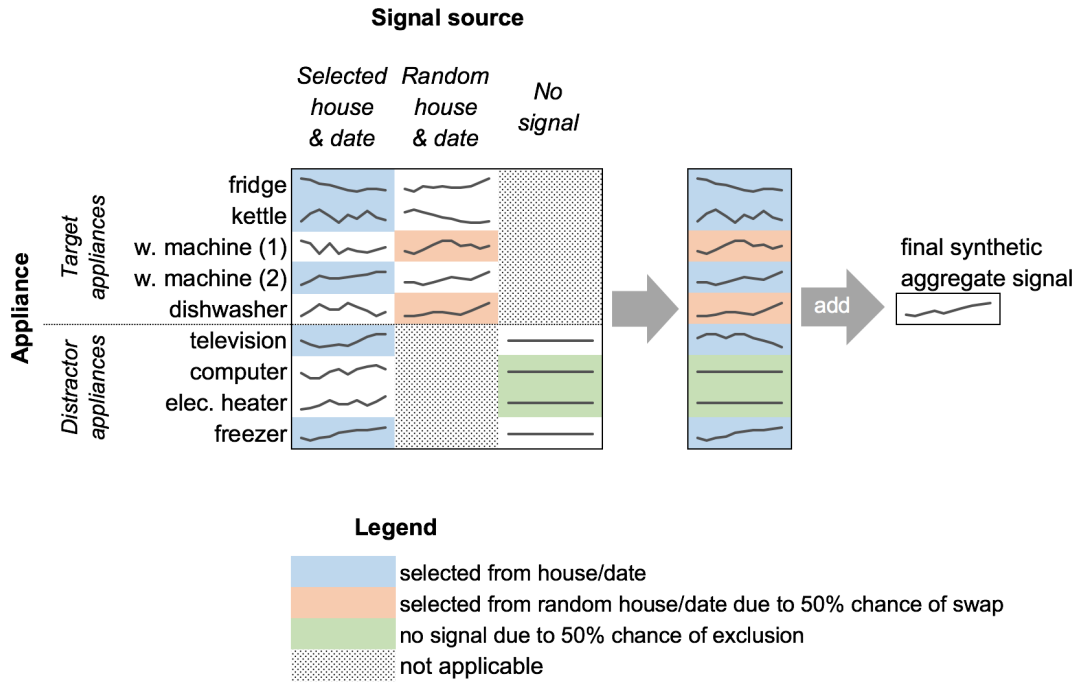

Figure 3.11: Process for creating synthetic data for an example house that has two washing machines but no microwave.

3.6.5 Data preprocessing: inputs

Data preprocessing is an important step to the training of neural networks, encouraging better performance and lower computational costs during training [33]. It is common to standardize the data by subtracting the mean and dividing by the standard deviation for each feature, or to scale features to be within the range $(0, 1)$ or $(-1, 1)$. For this project, we do something akin to the first, since we do not want the range of the data to be completely defined by power spikes.

However, our approach will be a bit more nuanced for three reasons:

1. The synthetic data has a systematically lower mean than the real data. The reason for this is that the synthetic data only consists of data from the IAMs, which do not account for all of the electricity usage (Appendix A.1).

2. We are dealing with time series data where each "feature" is a timestep. Standardizing each feature separately would simply not make sense because each

each house and date combination, and then subtract off appliance signals in order to either swap them with other signals or to remove them altogether. Yet another approach would be to subtract off all IAMs to create a "residual" series that could be used to create even more realistic data. However, these approaches are not possible due to the misalignment of timestamps between power series, which would have created spiky artifacts (Appendix A.1).

value would then become relative to the cross-series timestep values instead of the other values in the series.

3. Dividing the synthetic data and real data by their respective standard deviations would destroy relative level change information between the datasets, when this is an important factor to distinguishing otherwise similar signatures. This matters in particular when distinguishing between the kettle and the microwave, whose signatures have very similar shapes but different power levels.

To address (1) and (2), we first demean the real and synthetic training data separately by subtracting off the sample means of *all* values of the respective matrices as

$$\ddot{X}^{(r)} = X^{(r)} - \hat{\mu}^{(r)}$$

and

$$\ddot{X}^{(s)} = X^{(s)} - \hat{\mu}^{(s)}$$

where $\ddot{X}$ is the demeaned dataset, X is the dataset with structure described in Section 3.6.2, $\hat{\mu}$ is the scalar matrix-wide mean,[22] and (r) and (s) signify real and synthetic respectively.

Now we do not have to worry about (3) when we vertically concatenate the real and synthetic matrices

$$\ddot{X} = \begin{bmatrix} \ddot{X}^{(r)} \\ \ddot{X}^{(s)} \end{bmatrix}$$

and shrink the scale

$$\tilde{X} = \frac{1}{\hat{\hat{\sigma}}} \ddot{X}$$

where $\tilde{X}$ is the standardized input dataset and $\hat{\hat{\sigma}}$ is the scalar sample standard deviation computed for *all* values of $\ddot{X}$.[23]

It was tested whether first-differencing the input series improved learning.[24] A neural network is in principle capable of learning first-differencing, but it is best to exploit this information beforehand if it is known to be a better representation of the

[22] We compute the matrix-wide mean for the given matrix X as $\hat{\mu} = \frac{1}{NK} \sum_{i=1}^{N} \sum_{d=1}^{D} X_{i,d}$ where N is the number of observations and D is the number of features or timesteps.

[23] In practice, the synthetic dataset has more observations than the real dataset. Since training is performed with data that was half synthetic and half real, we calculate $\hat{\hat{\sigma}}$ to account for each dataset equally instead of naturally giving more weight to the synthetic dataset.

[24] To first-difference a series $\mathbf{x} = \{x_1, x_2, ..., x_D\}$ is to subtract off the previous value (the "lag") for every element of the series, creating the series $\{x_1 - x_0, x_2 - x_1, ..., x_D - x_{D-1}\}$. Note that the first-differenced series has $D-1$ rather than D elements.

data. The motivation for this is that the signatures would no longer stack like they do in the usual aggregate series, potentially relieving the network of learning power levels and allowing it to focus on power differences. This also naturally standardizes the data so it has a mean of zero. However, it was not found to be helpful in pilot tests, resulting in higher error.

3.6.6 Data preprocessing: targets

In this project we will standardize the outputs, which is uncommon in machine learning. We do so because we do not want to favor the error of one appliance over another when training the multi-target models. Otherwise these multi-target models, which compute loss as a mean across N observations *and* across K targets, would have a strong bias for minimizing loss for high-energy appliances (like the washing machine) at the expense of minimizing the loss of low-energy appliances (like the kettle) when possible. While this may be desirable in some domain applications, it is most interesting to treat all appliances equally so that we could compare the outputs of the multi-target model with those of the single-target models.

Each target variable for each appliance is therefore standardized separately by simply dividing by its standard deviation. We do not subtract off the mean because we will find it useful to have the minimum possible value equal to zero.[25] The standard deviations of the target variables are saved so that it is possible to recover the original targets later.

3.7 Hyperparameter and architecture selection

3.7.1 Architecture design overview

Our learning problem can be framed as a multi-class, multi-label or regression problem. A problem is best framed as multi-class or multi-label when there are only a few possible target values [21]. Since this is not the case for all of our appliances, we use regression to keep the learning problem and the loss function the same for all models. This makes comparison between appliance models easier and simplifies implementation.

It is common to use proven, state-of-the-art neural network architectures as the

[25] See Section 3.7.5 for how this affects the activation function of the final dense layer in the networks.

basis for related learning problems, since building architectures from scratch is challenging [1]. However, the learning problem for this project is unexplored, so we will have to work harder to build the architectures.

We can take some design inspiration from two areas:

- **Image recognition.** One major benefit of using the insights from image recognition is that the field is well-explored. However, the downside is that it does not apply directly to our problem, namely because images are 2D (our series are 1D), the signatures are quite rich (ours are comparatively simple) and image recognition tends to be about classification (ours is regression). Much of our inspiration for image recognition will come from VGGNet [13].

- **NILM.** Since the summarization task is different from energy disaggregation, NILM neural network architectures cannot be used directly. However, it is reasonable to assume that NILM models may learn some of the same low-level features as our summarization models, and therefore some architecture elements that are successful for NILM may also be successful for our problem. Some examples include the number of filters and the kernel size in the convolutional layers. Our architecture inspiration will come from PointNet (Figure 3.12, architecture (A)).

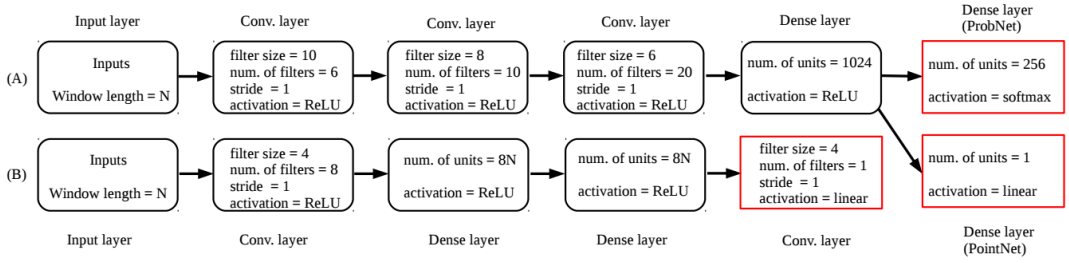

Figure 3.12: NILM network architectures as depicted in [5]. PointNet is the first row (A), following the arrows to the PointNet dense layer. The second row (B) describes a competing network that that predicts an entire sequence of power levels of a appliance instead of just the midpoint.

The typical architecture for a CNN is

```
INPUT -> [[CONV w/ RELU]*N -> POOL?]*M -> [FC -> RELU]*K -> FC
```

where * denotes repetition, INPUT is the input layer, CONV is a convolutional layer, RELU is a ReLU activation layer[26] (often simply defined in the previous layer as the activation function), POOL is an optional max pooling layer, and FC is a fully connected ("dense") layer. Typically, 0 <= N <= 3 and K < 3 [1]. In this project we will use N = 1 to include more pooling layers to handle the high dimensionality of our input space. We will stick to the usual K < 3.

As demonstrated by state-of-the-art architectures—including VGGNet—it is effective to increase (usually double) the number of filters with each convolutional layer or every few convolutional layers, and to decrease the size of consecutive dense layers. We will therefore follow this pattern in our architectures.

Algorithm 1 gives a high-level overview of the process to dynamically create the architectures used in this project. In particular, it describes the relationship between the number of convolutional layers, the number of filters in the convolutional layers, the number of dense layers, and the number of units in the dense layers. Other major elements of the network that not discussed in Algorithm 1, such as the kernel size, are described in Sections 3.7.3 and 3.7.4.

Data: (1) number of convolutional layers; (2) number of filters in the first convolutional layer; (3) number of hidden dense layers; (4) size of last dense layer
Result: model
Initialize empty model
for *i in 0 to number of convolutional layers* **do**
    Add convolutional layer with ($2^i \times$ num. first conv. layer filters) filters
    Add pooling layer
end
Add dropout layer with dropout rate $= 0.5$
Add flattening layer
for *i in 0 to number of dense layers* **do**
    Add dense layer with ($2^{(\text{num. dense layers})-i-1)} \times$ last dense layer size) hidden units
    Add dropout layer with dropout rate $= 0.25$
end
Add dense layer with K output units
return model

Algorithm 1: Bare-bones description of how the number of filters and the size of the dense layers are determined.

In addition to the single-target appliance-specific models, we will train two multi-target models—one for each target variable. As discussed in Section 3.6.2, these models will have one output for each appliance ($K = 5$). The entire internal representation—

[26] The ReLU activation function is explained in Section 3.7.4.

that is, all hidden layers—will be shared, which is the extreme case of "hard" parameter sharing. This is unlike the usual case of hard parameter sharing, where at least one layer is reserved for each task (Figure 3.13) [4]. This is done for simplicity in the architecture selection process.

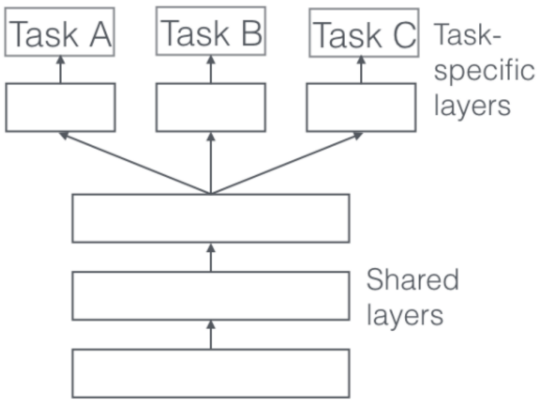

Figure 3.13: Typical representation of hard parameter sharing with multi-task learning. Image from [4].

3.7.2 Hyperparameter selection procedure

The primary difficulty of designing neural network models for a new learning problem is selecting the appropriate hyperparameters. (For the sake of brevity, the term "hyperparamters" in the rest of this paper will also include aspects of the architecture, such as the number of convolutional layers.) Among the most used and most straightforward strategies to find appropriate hyperparameters are grid search and manual search.[27]

Given that the learning problem in this project is unexplored, and that each model is quite computationally expensive to train, we will instead use random search. Random search is a simple process where each hyperparameter is chosen randomly from some distribution, the model is trained, and the validation error calculated. The process is repeated as many times as the practitioner would like, and typically the model with the best validation error is selected. Empirically and theoretically, "randomly chosen trials are more efficient for hyper-parameter optimization than trials on a grid," often finding

[27]Manual search is a time-intensive, unstructured process when the practitioner tests sets of hyperparameters one by one, relying on the performance results of each run (and a good deal of intuition) to get increasingly close to a set of hyperparameters that perform well. Grid search is computationally intensive, structured process that tries each possible combination of hyperparameter values. In both cases, validation error is usually used to select among models.

a better set of hyperparameters given a fixed computational budget [34].[28]

3.7.3 Random hyperparameter choices

Before defining how hyperparameter values are randomly selected in each iteration of random search, it is useful to first define some (simplified) notation. Let $\text{Uniform}(a,b)$ be a uniform distribution between a and b, $\text{Geom}(a,b)$ be geometric sampling between a and b,[29] $\text{Choice}(S)$ be the uniform choice between discrete values in set S, $\lfloor x \rfloor$ be the greatest integer that is less than or equal to x (often referred to as the "floor" of x), and $x \sim y$ be short for "x is distributed as y." Note that when we sample from, say, $\lfloor \text{Geom}(1,10) \rfloor$, the highest possible value that can be chosen is 9 since the probability of selecting exactly 10 is zero according to the uniform distribution.

Given these definitions, the bullet points below describe the hyperparameters, their variable names in the project (for reference later in this paper), their sampling distributions, and the reasoning behind the distribution choices.

- **Number of convolutional layers** (variable name `num_conv_layers`) $\sim \text{Choice}(\{4,5,6,7\})$. Fewer convolutional layers were not found to have good performance in pilot tests. More convolutional layers, when paired with positive strides and pooling, often lead to the network collapsing to a single timestep in at least one of the convolutional layers.[30]

- **Number of dense hidden layers** (`num_dense_layers`) $\sim \text{Choice}(\{1,2\})$. This defines the number of dense hidden layers, which are before the final dense layer. More or fewer dense hidden layers were not found to be helpful.

- **Number of filters in the first convolutional layer** (`start_filters`) $\sim \lfloor \text{Geom}(4,9) \rfloor$. VGG-19 starts with 64 filters in the first layer, and this number doubles in the 3rd, 5th, 9th, and 13th convolutional layers, with the final convolutional layer having 512 filters [13]. Taking into account that we have one dimension instead of two,

[28] Random search performs well because in many applications, performance results are not sensitive to many of the hyperparameters, so random search does not waste trials by iterating through these uninfluential hyperparameters while holding all other hyperparameter values constant. Random search performs particularly well on problems for which there is not a strong prior for the appropriate hyperparameter values.

[29] $\text{Geom}(a,b)$ is equivalent to $\exp(\text{Uniform}(\log(a),\log(b)))$, favoring values closer to a than to b. This is good for sampling for values for which the range of possible values is effectively all positive numbers, or when we want to give a preference to numbers at the lower end of the distribution.

[30] A check was put in place to select a different set of hyperparameters when this occurred, but choosing a reasonable value for the number of convolutional layers was still worth doing.

then it is reasonable to believe that $\sqrt{64} = 8$ filters might be a good number of filters for our first convolutional layer. However, given that our data is sparser and contains signals that are less rich when compared to image data, then half of this number may also be reasonable. Sampling geometrically between 4 and 8, we have a bias for values closer to 4 for the sake of model simplicity. PointNet has a starting filter size of 6, supporting our range of choices [5].

- **Convolutional layer kernel size** (`kernel_size`) $\sim \lfloor \text{Geom}(3,7) \rfloor$. VGGNet found 3×3 to be most effective kernel size [13]. In fact, this currently seems to be the most popular kernel size for image recognition problems [1]. The one-dimension version is a length-3 filter. However, PointNet found that larger filter sizes work well, from 10 in the first convolutional layer to 6 in the last one. Giving some preference to VGGNet in this case, we make 6 the upper bound in our sampling. This variable affects all convolutional layers in the network.[31]

- **Stride length of the convolutional layer filter** (`strides`) $\sim \lfloor \text{Geom}(1,3) \rfloor$ A stride length of 1 in image recognition problems is considered most effective (since it destroys the least amount of information), while a stride length of 2 is more computationally efficient [1]. We will sample geometrically between the two, choosing 1 more often since this is what PointNet uses. This variable affects all convolutional layers in the network.

- **Max pooling size** (`pool_size`) $\sim \lfloor \text{Geom}(2,5) \rfloor$. A max pooling size of 2×2 with a stride of 2 in both spacial dimensions is most popular for image recognition problems [1]. The 1D equivalent is a window of size 2 with a stride of length 2. We will also test pooling sizes up to 4 (with the stride equal to the pooling size) since our inputs are especially large and sparse. The use of pooling follows the precedent of VGGNet. This variable affects all pooling layers in the network.[32]

- **Size of the last dense hidden layer** (`last_dense_layer_size`) $\sim \lfloor \text{Geom}(8,33) \rfloor$. Each dense hidden layer has half the number of units as the dense layer before it (except for the first one). Larger dense layers were not helpful in pilot tests.

[31] "Zero-padding" is also used, meaning that each side of an input layer is padded with zero-valued nodes so that when `strides=1`, the dimensionality of the data does not change. This is used simply to make the layer sizes more intuitive on inspection.

[32] "Zero-padding" is used here, as it was with the convolutional filter.

- **Learning rate** (`learning_rate`) $\sim$ Geom$(0.0003, 0.003)$. Values outside of this range did not work well in pilot tests.

- **L2 regularization penalty** (`l2_penalty`) $\sim$ Choice$(0,$ Geom$(10^{-8}, 10^{-6}))$. Only very small L2 penalties (or none at all) worked well in pilot tests. We apply this penalty to weights in all convolutional *and* dense layers.

3.7.4 Static hyperparameters

While some hyperparameters were chosen dynamically, some were set to a static value either because (1) pilot tests showed a clear preference for one value over another; and/or (2) to simplify the random search process. These static hyperparameters are:

- **Hidden layer activation function** (`hidden_layer_activation`). This variable affects all hidden layer activation functions in the neural network. We use ReLU, defined as $f(x) = \max(0, x)$, since it is one of the most popular activation functions (if not *the* most popular). The sigmoid is not tested because it has fallen out of favor due to a tendency to cause the weights to get trapped—or "saturated"—at either end of the function, slowing down learning. Pilot tests also tried ELU ("exponential linear unit")[33] since there is some evidence that it "speeds up learning in deep neural networks and leads to higher classification accuracies" when compared to ReLU [35]. However, it did not give perceptibly better results than ReLU.

- **Dilation** (`dilation_rate`). Dilated convolutions[34] were found not to help learning, so we set `dilation_rate` to 1, meaning no dilation.

- **Deepening of the filters** (`deepen_filters`). This is a binary variable that determines whether to double the number of filters in each progressive convolutional layer. As discussed in Section 3.7.1, we set this to `True` because state-of-the-art models like VGGNet have shown that it is an effective technique. It also resulted in better performance in pilot tests.

- **Use pooling** (`do_pool`). This is a binary variable for whether max pooling layers should be used. As mentioned in Section 3.7.1, we set this to `True` to handle the

[33] ELU is defined as $f(\alpha, x) = \begin{cases} \alpha(e^x - 1) & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases}$.

[34] Dilated convolutions use large but sparse feature detectors that "increase the receptive field by orders of magnitude, without greatly increasing computational cost" [36].

high dimensionality of the input space. There is precedence for this in VGGNet. It also resulted in better performance in pilot tests.

- **Dropout probabilities** (`dropout_rate_after_conv` and `dropout_rate_after_dense`). This refers to the dropout rates in the dropout layer[35] after the convolutional layers, and in the dropout layer after each dense layer. We set these rates to 0.5 and 0.25 respectively, since these are what work well in an instructional example created by the original author of Keras [38].

- **Batch normalization** (`use_batch_norm`).[36] This is a binary indicator for whether a batch normalization layer should be used after each dense layer. These layers seem to result in higher error rates, so we set this to `False`.

- **Optimizer** (`optimizer`).[37] We choose the Adam[38] optimizer simply because it is one of a few standard, high-performing optimizers. We do not vary this because the learning rate often has to be tweaked separately for each optimizer, so varying the two simultaneously would have led to too much complexity when inspecting the results of the random search process. Outside of the learning rate, we leave the Adam parameters to the Keras default values [24], inspired by the original Adam paper [40]: $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 10^{-8}$, and decay $= 0$.

3.7.5 Other network and training elements

- **Activation of the final dense layer**. This is set to be linear in most regression problems, but we know beforehand that energy or activation predictions below zero are incorrect. Therefore we use a ReLU activation function to turn negative predictions into zeros. Given the number of target values that are exactly zero, ReLU is preferred to smooth approximations of ReLU—such as softplus[39]—

[35] Dropout is a technique where some proportion of hidden units of a layer are set to zero during the training process. This makes the network behave as if it were an average of multiple networks, usually leading to better generalization [37].

[36] Batch normalization is a technique that addresses the problem of "internal covariate shift," which is the shifting of the distribution of layer activations during training [39]. It does so through specialized layers that normalize activations and then sometimes apply further adjustments to the normalized values. Reported benefits include regularization and improved speed of convergence.

[37] The optimizer is the algorithm that handles how the weights are updated every iteration of the parameter optimization.

[38] Adam is an adaptive learning rate schedule. With standard gradient descent, the gradient of the loss function is used to update the weights. But with Adam, the gradient is "smoothed" using the momentum of previous weight updates, usually leading to faster convergence [40, 2].

[39] The softplus function is defined as $f(x) = \ln[1 + \exp(x)]$.

which make predictions of exactly zero impossible.

- **Loss function**. This is chosen to be mean square error (MSE), since it is the most straightforward loss function for regression problems, and there is not a strong reason to use something else. We will also use this metric to evaluate our models (Section 4).

- **Weight and bias initialization**. The weights are initialized using the Glorot uniform initialization.[40] The biases are initialized with zeros.

- **Epochs, samples per epoch, batch size and early stopping**. The models are run for a maximum of 100 epochs, with 8192 examples per epoch.[41] Half of the samples are real and half are synthetic. Each batch has 32 examples.[42] The training of the model is terminated if the validation error does not improve in the previous few epochs, and the weights at the epoch with the best validation error are saved.[43]

3.7.6 Example architecture

To give an example of how the random hyperparameter choices affect the architecture, we can select some hyperparameters that might be sampled in an iteration of the random search process.

- `num_conv_layers = 5`
- `num_dense_layers = 2`
- `start_filters = 4`
- `kernel_size = 3`
- `strides = 1`

[40] The Glorot uniform initialization draws samples from Uniform$(-a, a)$ where $a = \sqrt{6/(x_{in} + x_{out})}$, x_{in} is the size of the input layer and x_{out} is the size of the output layer.

[41] An epoch usually means a pass through the training data. However, this is not a strict rule, and the number of examples per epoch can vary based on the data, the learning problem, the volatility of the validation error, and other factors.

[42] A batch is a group of training examples. The size of the batch refers to the number of observations that are used to compute the gradient, which is then used to update the weights during an iteration of the optimization algorithm.

[43] This is commonly called "early stopping." In the first 15 iterations of the random search process, the "patience" was set to 5, where patience is the number of epochs the model would wait without seeing an improvement in validation error before stopping the training process. However, after observing the volatility of the validation error curves at around 15 iterations of the random search process, patience was increased to 10 for the remaining 10 iterations. Longer patience did not seem to result in markedly lower validation errors.

- `pool_size = 2`
- `last_dense_layer_size = 16`
- `learning_rate = 0.001`
- `l2_penalty = 1e-7`

The resulting architecture for a single-target model using the hyperparameters above would produce a network of 18 layers and 930,473 parameters.[44] That architecture is depicted below (in the style of [6]) with its elements colored by the hyperparameters that determine their value. Note that the L2 regularization penalty is not included (for concision), and neither is the learning rate. The "output shape" is in the format "(timestep dimension, number of filters)" for layers with filters, and "timstep dimension" for all other layers. Also specified in *italics* are the number of parameters in the associated layer.

1. Input (output shape = 14,400)
2. **1D conv** (kernel size = 3, stride = 1, number of filters = 4, activation function = ReLU, output shape = (14,400, 4)) (*16 parameters*)
3. **1D max pool** (pool size = 2, stride = 1, output shape = (7,200, 4)) (*0 parameters*)
4. **1D conv** (kernel size = 3, stride = 1, number of filters = 8, activation function = ReLU, output shape = (7,200, 8)) (*104 parameters*)
5. **1D max pool** (pool size = 2, stride = 1, output shape = (3,600, 8)) (*0 parameters*)
6. **1D conv** (kernel size = 3, stride = 1, number of filters = 16, activation function = ReLU, output shape = (3,600, 16)) (*400 parameters*)
7. **1D max pool** (pool size = 2, stride = 1, output shape = (1,800, 16)) (*0 parameters*)
8. **1D conv** (kernel size = 3, stride = 1, number of filters = 32, activation function = ReLU, output shape = (1,800, 32)) (*1,568 parameters*)
9. **1D max pool** (pool size = 2, stride = 1, output shape = (900, 32)) (*0 parameters*)
10. **1D conv** (kernel size = 3, stride = 1, number of filters = 64, activation function = ReLU, output shape = (900, 64)) (*6,208 parameters*)
11. **1D max pool** (pool size = 2, stride = 1, output shape = (450, 64)) (*0 parameters*)
12. **Dropout** (dropout rate = 0.5, output shape = (450, 64)) (*0 parameters*)
13. **Flatten** (output shape = 28,800) (*0 parameters*)
14. **Fully connected** (activation function = ReLU, output shape = 32)) (*921,632 parameters*)
15. **Dropout** (dropout rate = 0.25, output shape = 32) (*0 parameters*)

[44] Parameters in the context of neural networks means trainable weights, not to be confused with hyperparameters.

16. **Fully connected** (activation function = ReLU, output shape = 16) (*528 parameters*)
17. **Dropout** (dropout rate = 0.25, output shape = 16) (*0 parameters*)
18. **Fully connected** (activation function = ReLU, output shape = 1) (*17 parameters*)

3.7.7 Hyperparameter selection results

In total, the random search procedure trains 300 networks, or 25 random search iterations for each of the 12 models. To compare models within each appliance and target variable combination, we first use a moving average with a window size of 3 to smooth the validation error over epochs for each model. The model with the lowest smoothed validation error is considered the "best" model.[45] When a new best model is found in the random search process, it is represented as a step change in Figure 3.14. The best models at the end of the 25 random search iterations are the models we choose to use for the remainder of this paper.

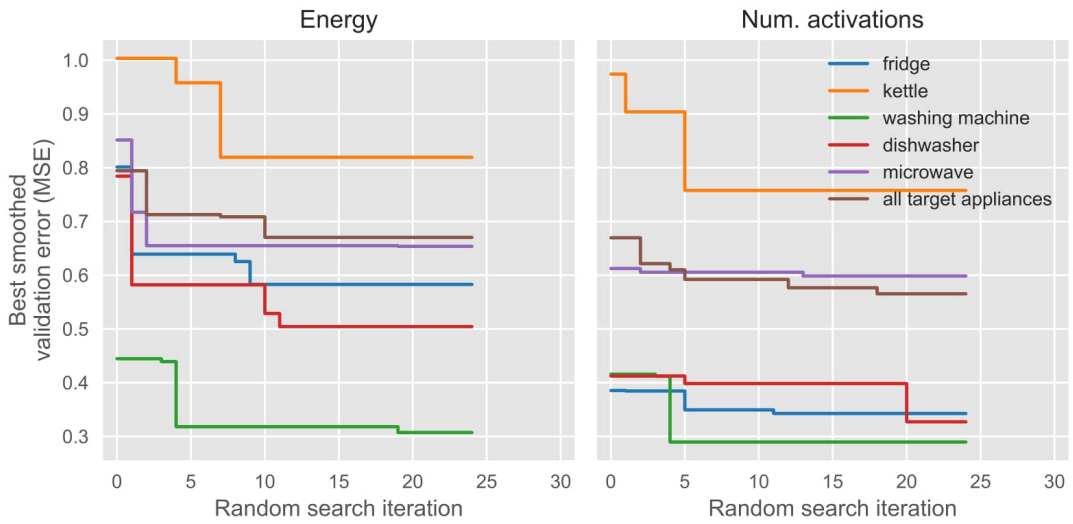

Figure 3.14: Results of the random search process. The curves represent the cumulative minimum of the minimum smoothed validation error of the models for the specified appliance and target variable combination. The validation error is computed on the preprocessed targets.

We see that validation error curves for the chosen models are quite erratic (Figure 3.15). The reason for this volatility is the choice of the validation set, which is mostly comprised of unseen houses. Because there is limited variety in the appliance

[45] The smoothing prevents us from choosing models with dramatic downward spikes.

signatures, a change in the weights of the model sometimes leads to very different predictions.[46] The multi-target models have the smoothest validation curve, which is likely because of two related reasons: (1) their errors are an average across target variables, which makes them more stable; and (2) their error signals are richer in that they come from five target variables.

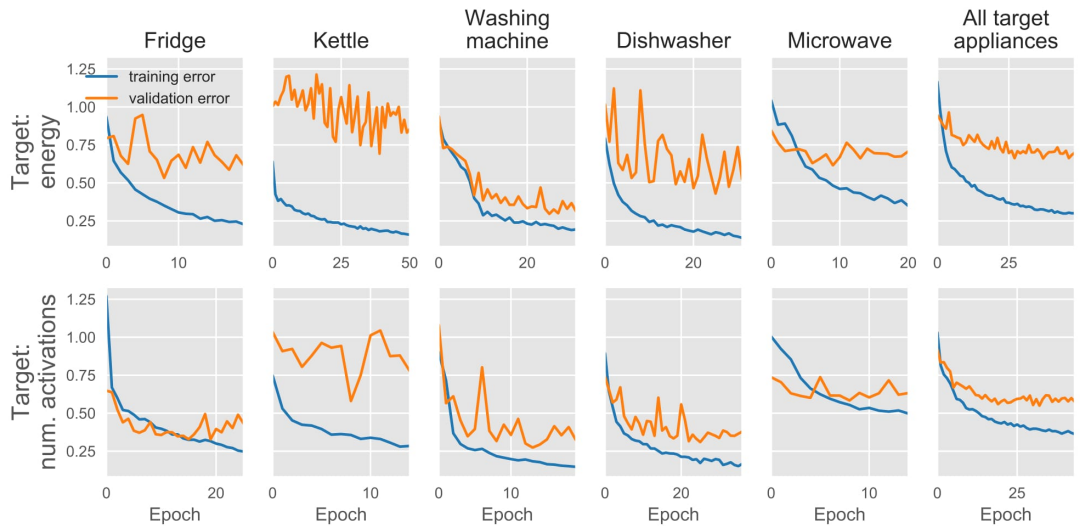

Figure 3.15: Training and validation error for the chosen models, where the error is computed on the preprocessed targets. The depicted validation error curves have not been smoothed, although the choice of the models was based on smoothed validation error. The number of epochs differ due to early stopping.

The architectures and hyperparameters of the chosen models are described in Tables 3.3 and 3.4. It is difficult to say why there is such variety in the architectures across networks. For example, we can say that the learning problem is different for each model, and that this is reflected in the difference in architectures. Or we could say that the models are insensitive to the hyperparameters within the ranges they were chosen from—which would imply that the choices of the distributions were good. Therefore, while we might be tempted to say that (for example) the washing machine models ended up having more filters so that they can learn to identify more complex signatures, we cannot rule out that this is the result of chance. However, two patterns that do seem robust across models are that larger pooling sizes and smaller strides perform best.

[46] To ensure that this was the reason for the volatility, a test was performed that validated the models on only the "seen" portion of the validation set, which resulted in much smoother validation curves (results not shown here).

	num_conv_layers	num_dense_layers	start_filters	kernel_size	strides	pool_size	last_dense_layer_size	learning_rate	l2_penalty	number of parameters
fridge	7	1	4	5	1	3	13	1.6e-3	1.0e-8	2.4e+05
kettle	6	2	4	6	1	4	9	2.4e-3	2.7e-7	7.5e+04
washing machine	7	1	7	4	1	3	23	1.2e-3	0.0e+1	6.1e+05
dishwasher	7	2	6	3	1	3	17	4.4e-4	6.2e-7	3.9e+05
microwave	5	2	6	3	1	4	26	2.8e-3	7.2e-8	9.5e+04
all target appliances	5	2	5	3	1	4	26	2.1e-3	3.7e-8	7.7e+04

Table 3.3: Final architectures for models predicting energy.

	num_conv_layers	num_dense_layers	start_filters	kernel_size	strides	pool_size	last_dense_layer_size	learning_rate	l2_penalty	number of parameters
fridge	5	1	5	6	1	3	25	1.4e-3	1.4e-8	1.5e+05
kettle	5	1	5	5	1	4	14	7.1e-4	0.0e+1	3.8e+04
washing machine	5	2	7	4	1	3	10	5.8e-4	3.6e-7	1.7e+05
dishwasher	6	1	4	6	1	3	15	4.1e-4	2.5e-8	1e+05
microwave	5	2	4	6	1	4	29	1.8e-3	0.0e+1	7.4e+04
all target appliances	6	2	5	6	1	3	22	8.1e-4	1.2e-8	2.4e+05

Table 3.4: Final architectures for models predicting the number of activations.

Chapter 4

Evaluation

4.1 Evaluation metrics

To evaluate our models, we define one or more loss functions $L^{(i)} = L(y^{(i)}, f(\boldsymbol{x}^{(i)}))$, where $L^{(i)}$ is the loss for test example i, $y^{(i)}$ is the target for test example i, $\boldsymbol{x}^{(i)}$ is the vector of input data for test example i, and f is the neural network model that takes an input vector and outputs a prediction. The two loss functions used as evaluation metrics in this paper are:

1. **Mean square error (MSE)**: $L^{(i)}_{MSE} = (y^{(i)} - f(\boldsymbol{x}^{(i)}))^2$

2. **Mean absolute error (MAE)**: $L^{(i)}_{MAE} = |y^{(i)} - f(\boldsymbol{x}^{(i)})|$

MSE will be the primary evaluation metric reported.[1] We use MSE simply because it is a standard metric for regression problems. We also used it as the loss function in the training of our networks. To make interpretation easier, we divide MSE by the MSE of the "naive baseline," which is the average of the target values from the real part of the test set. So if a model's MSE relative to the baseline is 0.7, then the model has a MSE that is 30% lower than the baseline. This baseline was chosen because average statistics for energy and number of activations are currently used as priors in

[1] It may seem that a more intuitive performance metric would be mean absolute percentage error (MAPE), $L^{(i)}_{MAPE} = 100 \times \left|\frac{y^{(i)} - f(\boldsymbol{x}^{(i)})}{y^{(i)}}\right|$ which is easily interpretable and naturally standardized for each metric. For example, one could say that our predictions for some target appliance are, on average, within $X\%$ of the actual daily value. Furthermore, it does seem as though a prediction error for a small target value should incur a more severe penalty than a same-sized error for a large target value. However, since our target data has many energy and activation values that are exactly zero ($y^{(i)} = 0$), MAPE is undefined for many values and is generally unstable when $y^{(i)}$ is close to zero. Therefore this metric is not used for this project, and neither are other metrics that rely on ratios or percentages.

the AFHMM with LBM model.[2] MAE was chosen because it has the benefit of being in the units of the target variable.

Total loss over the test set when using loss function L is computed as $L_{\text{test}} = \frac{1}{N}\sum_{i=1}^{N} L^{(i)}$ where N is the number of examples in the test set. Since L_{test} is itself a random variable, we can compute the standard error as $\sigma_{L_{\text{test}}} = \sqrt{\sigma_L^2/N}$ where σ_L is the standard deviation of L. Metrics are reported with a range of two standard errors, which represents a confidence interval of roughly 95%.

4.2 Performance results

4.2.1 Single-target models

The results for MSE, presented in Table 4.1 and visualized in Figure 4.1, show that the models are successful at predicting both the daily energy and the daily number of activations on unseen days for houses the model saw during training ("seen"). Most target appliance models perform at around half error of the baseline or better.

The washing machine and the dishwasher perform particularly well on seen houses, with errors roughly 60–80% lower than the baseline for the prediction of both energy and number of activations. It is not surprising that these appliances perform well, since the richness and distinctness of the signatures make the models better able to pick them out from the noise of the aggregate signal and less likely to confuse them with the signatures of other appliances. This contrasts with NILM, where neural network models find the rectangular signature of the kettle easiest to predict [5]. (The fridge also performs well on seen houses, but we will explore why this is the case later.)

Most models also generalize quite well to unseen houses. The models that perform best on unseen houses are again the washing machine and dishwasher models. For example, the model that predicts washing machine activations has an error that is 57–71% lower than the baseline, while the dishwasher activations model has an error that is 54–66% lower. In the original units of the appliances, the washing machine activations model is off by 0.34–0.40 activations on average, while the dishwasher is off by 0.31–0.36 activations (Appendix C).

The model for fridge activations generalize very poorly. In fact, they perform much worse than the baseline. This is likely because there was so much more variation

[2] There is also not a simple model that would perform well on this sort of complex problem outside of other neural networks.

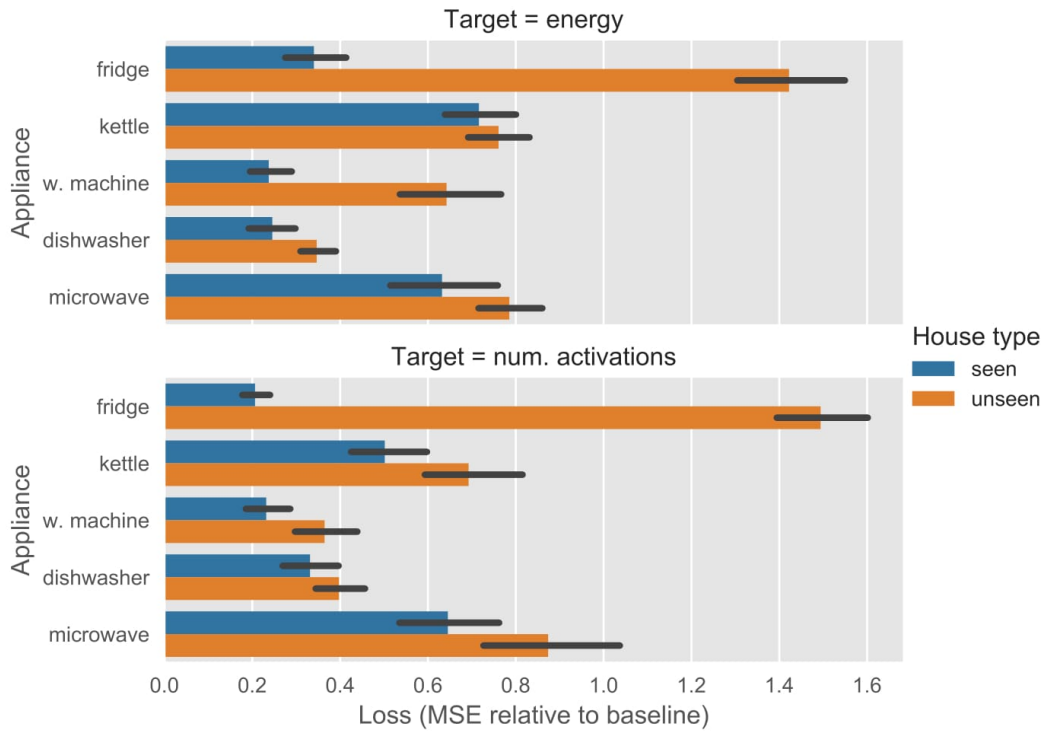

Figure 4.1: Test error relative to the baseline for single-target models, with 95% confidence intervals.

		Loss (MSE relative to baseline)	
Appliance	House type	Energy	Num. activations
fridge	seen	0.34 ± 0.07	0.21 ± 0.03
	unseen	1.42 ± 0.13	1.49 ± 0.11
kettle	seen	0.72 ± 0.08	0.50 ± 0.09
	unseen	0.76 ± 0.07	0.69 ± 0.11
w. machine	seen	0.24 ± 0.05	0.23 ± 0.05
	unseen	0.64 ± 0.12	0.36 ± 0.07
dishwasher	seen	0.25 ± 0.05	0.33 ± 0.07
	unseen	0.35 ± 0.04	0.40 ± 0.06
microwave	seen	0.63 ± 0.12	0.65 ± 0.12
	unseen	0.79 ± 0.08	0.87 ± 0.17

Table 4.1: Test error relative to the baseline for single-target models, with 95% confidence intervals.

between houses than within houses (reminder: Section 3.4.2), encouraging the model to focus on house-specific signals instead the fridge signatures to make the predictions.

For both seen and unseen houses, it does not seem that predicting the number of activations is markedly easier than predicting energy—or vice versa. This is not surprising given the high correlation between energy and the number of activations.

4.2.2 Multi-target models

The multi-target model does not generalize as well as the single-target models (Figure 4.2). In fact, all multi-target models have a generalization error that is either equal to or greater than that of the single-target models. It is unclear whether this is because the models need to be more flexible than the single-target models (e.g., by making the neural networks larger or including task-specific hidden layers), or because multi-task learning is simply not beneficial for this type of learning problem.

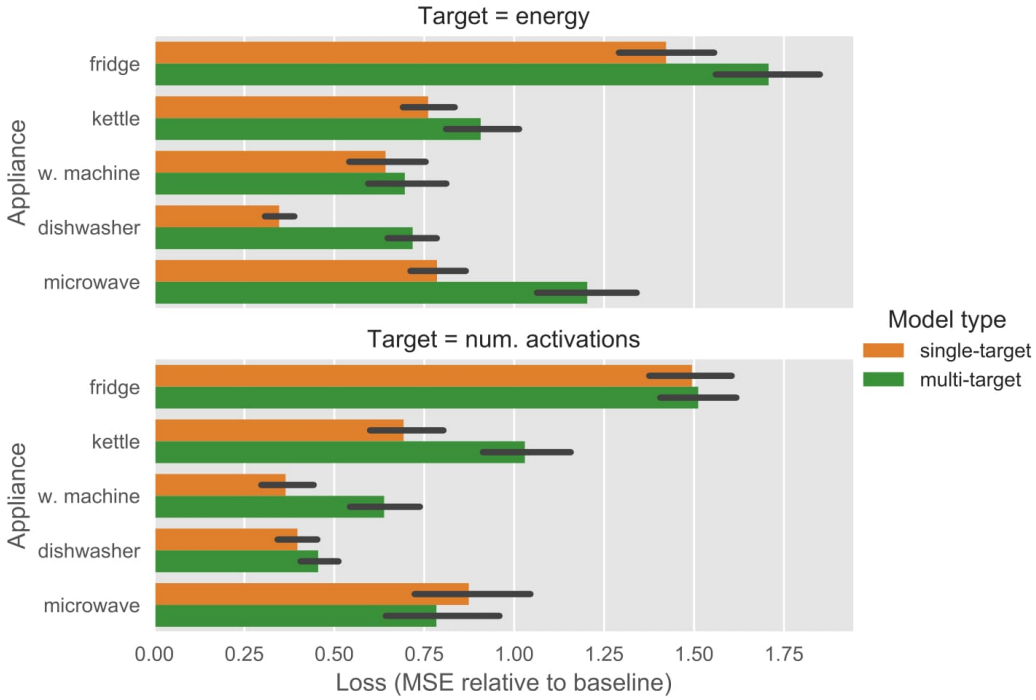

Figure 4.2: Test error relative to the baseline for unseen houses, with 95% confidence intervals.

4.2.3 Accuracy of activations feedback

If the statistics are to be reported directly to consumers, then it is helpful to explore the accuracy of these statistics. Perhaps the most useful feedback would be for the washing machine and the dishwasher, since they are high-energy appliances where feedback is actionable.

We see that when there are zero washing machine activations a day, the model gives the correct prediction 89% of the time (Figure 4.3). When there is one activation, the model predicts this with 78% accuracy. And with two or more, 61% accuracy.[3] The dishwasher has a tighter range for the number of activations per day, so the model predictions are even more accurate (Figure 4.4).

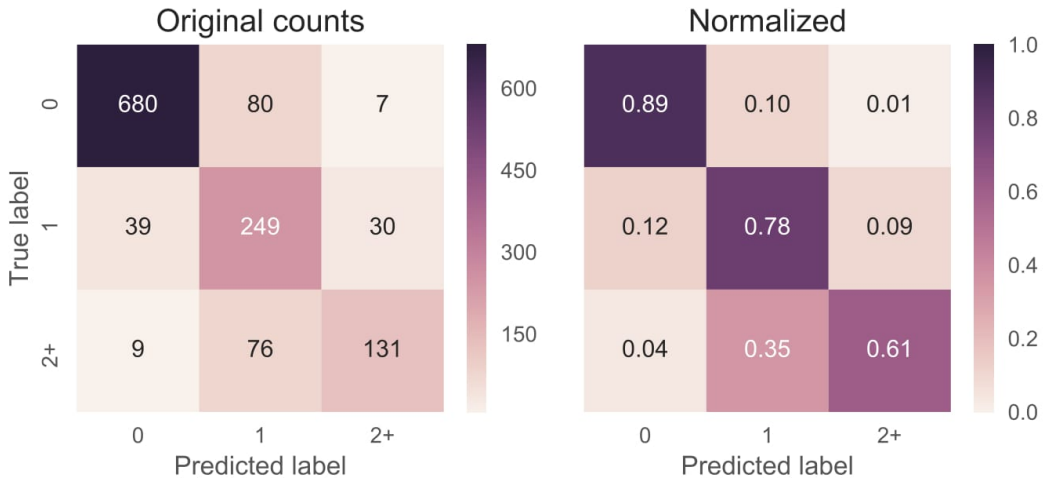

Figure 4.3: Confusion matrix for washing machine activations on unseen houses. Model predictions are rounded to the nearest integer.

It is a bit difficult to predict the exact number of kettle activations since the kettle can have quite a few activations per day (Figure 4.5). To improve accuracy, it could be helpful to provide feedback as ranges of activations, such as: "You used your kettle 3 to 5 times yesterday." The kettle predictions tend to have a downward bias that gets more pronounced as the number of target activations increases.

A description of the microwave activations can be found in Appendix C. The fridge was not plotted since feedback would not be actionable, and there were too many activations for a sensible plot.

[3]To determine the cutoff for the number of activations in the confusion matrices, the 95th percentiles of the targets and predictions are calculated. Then the maximum of these two values are used as the upper limit for both axes in the confusion matrix.

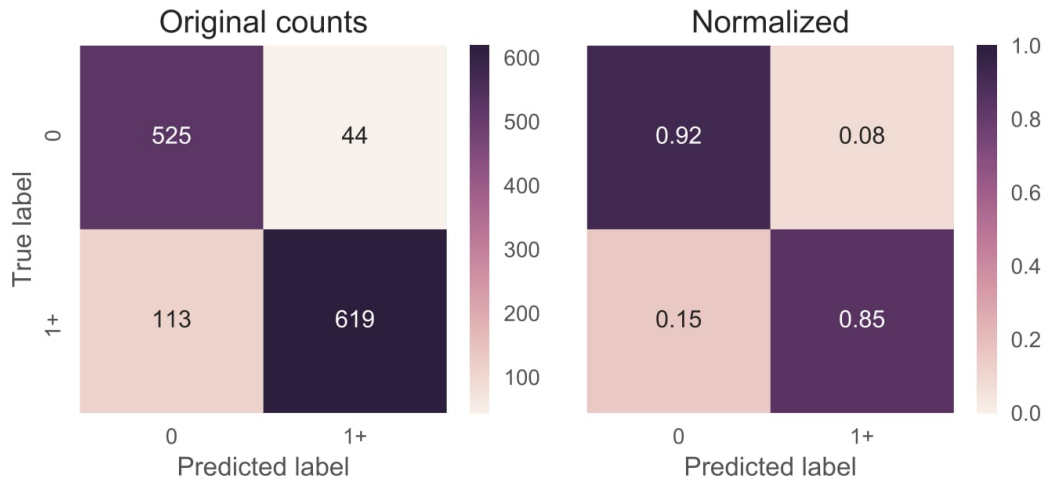

Figure 4.4: Confusion matrix for dishwasher activations on unseen houses. Model predictions are rounded to the nearest integer.

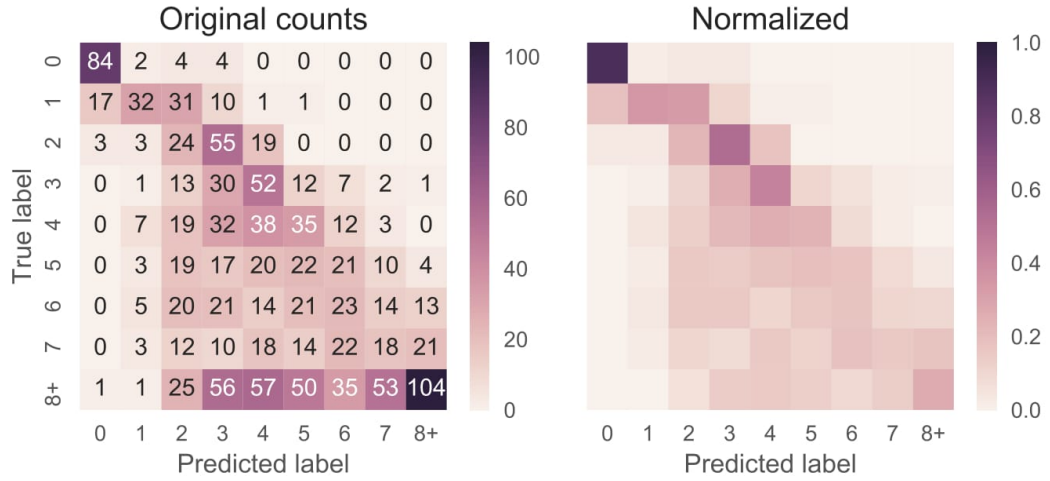

Figure 4.5: Confusion matrix for kettle activations on unseen houses. Model predictions are rounded to the nearest integer.

4.2.4 Results by house

As we saw in Section 3.4.2, it helps to explore the data by house to uncover patterns that may be obscured by simply looking at the headline statistics. Starting with the washing machine activations model, we see that the predictions are correlated with the target values, with the predictions for seen houses being a little bit tighter than for unseen houses (Figure 4.6). The predictions for washing machine energy are similar (Appendix C).

When visualizing the predictions of the fridge activations model by house, the problem of high between-house variance and low within-house variance becomes ap-

parent (Figure 4.7). The model struggles to predict the number of activations for individual houses. When it is unsure of the house, it has a tendency to predict a constant number of activations of around 20 or 25.

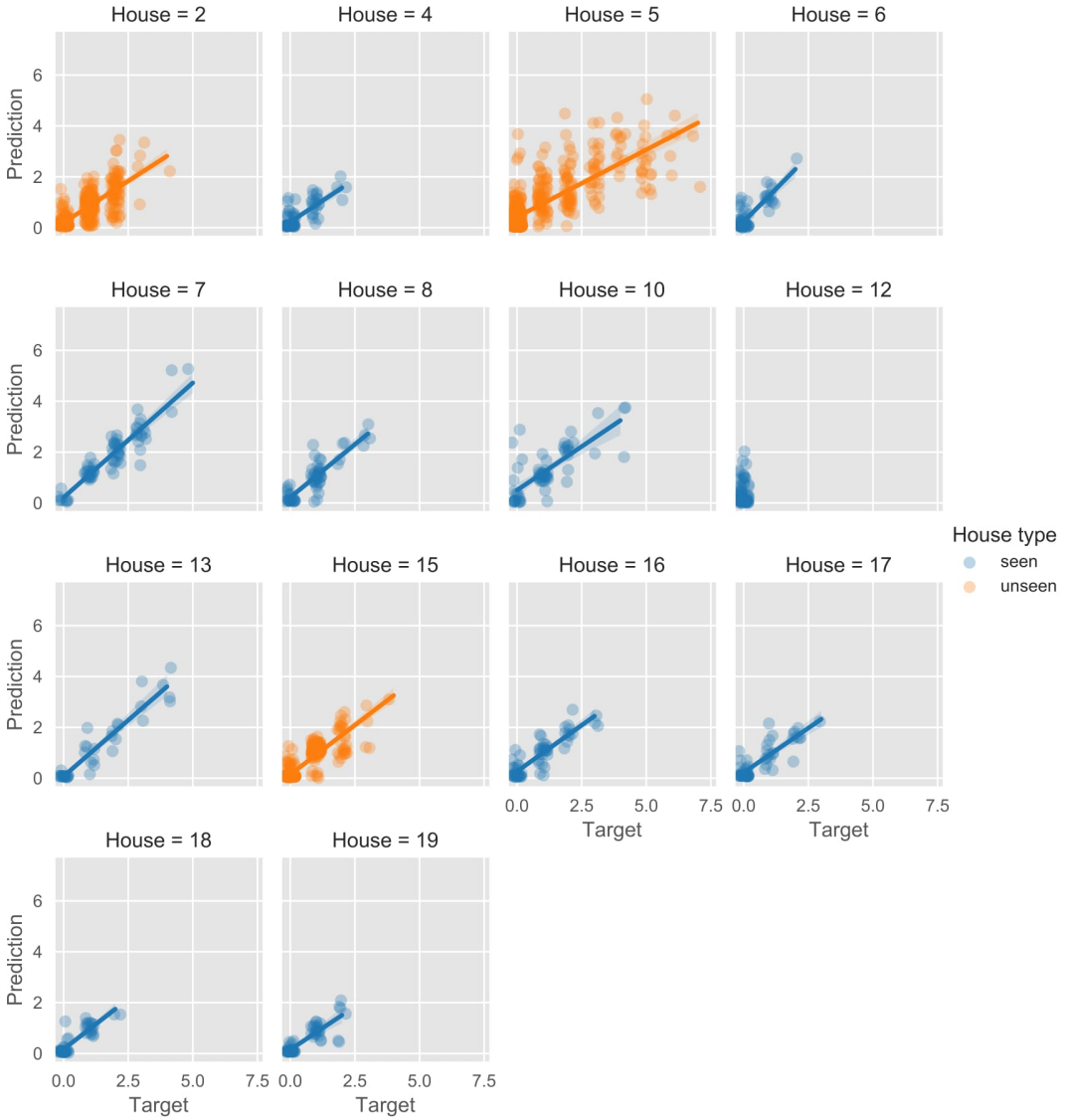

Figure 4.6: Predictions vs. targets for washing machine activations. The points are jittered along the x-axis to reduce overlap.

Plots for the other appliance models can be found in Appendix C.

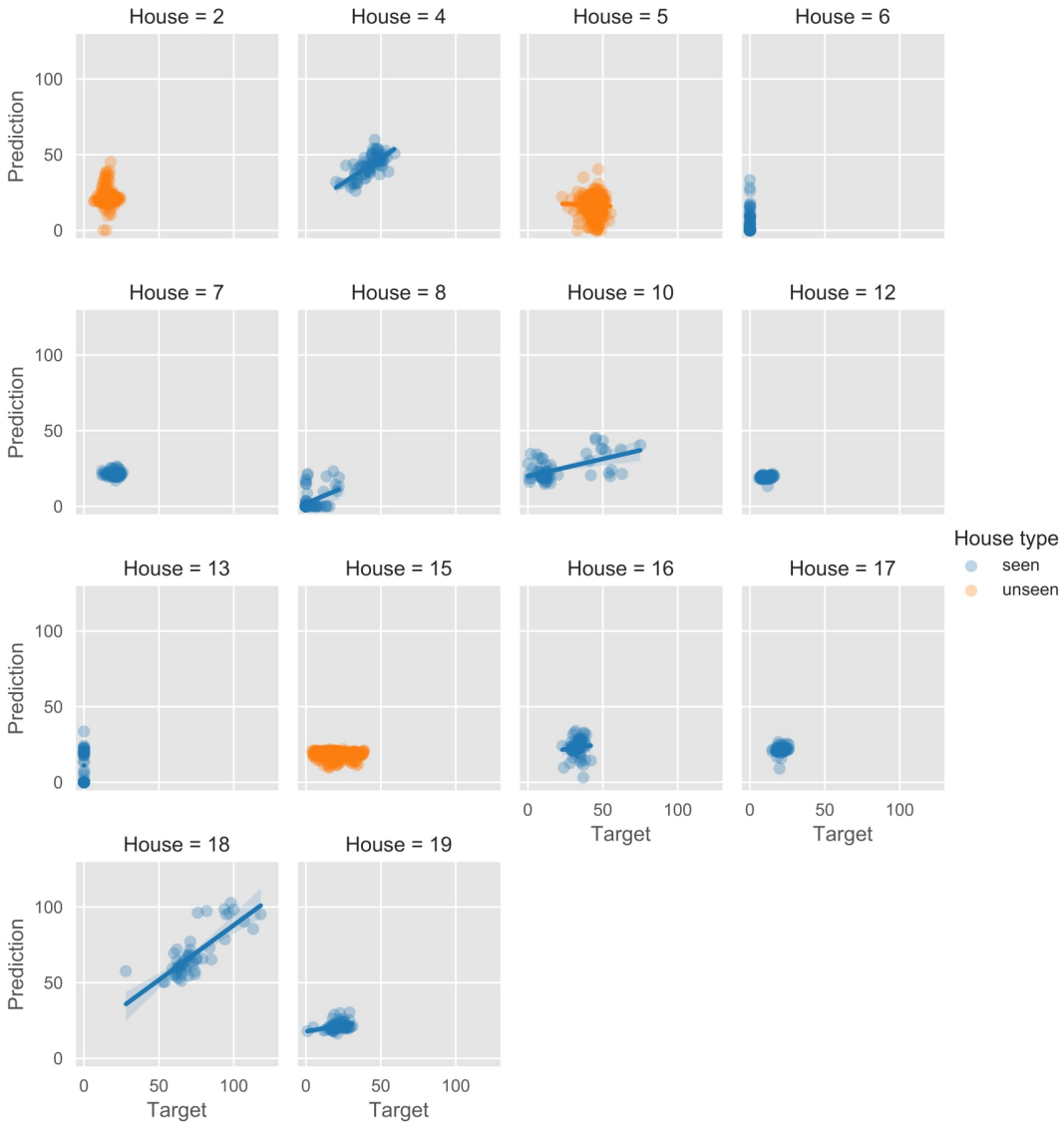

Figure 4.7: Predictions vs. targets for fridge activations.

4.3 Importance of synthetic data

In order to test the importance of the synthetic data, we retrain the networks with the best-performing architectures on only real data and recompute the evaluation metrics. The results show that the synthetic data is extremely important for generalization (Figure 4.8 and Table 4.2 for MSE; Appendix C for MAE).

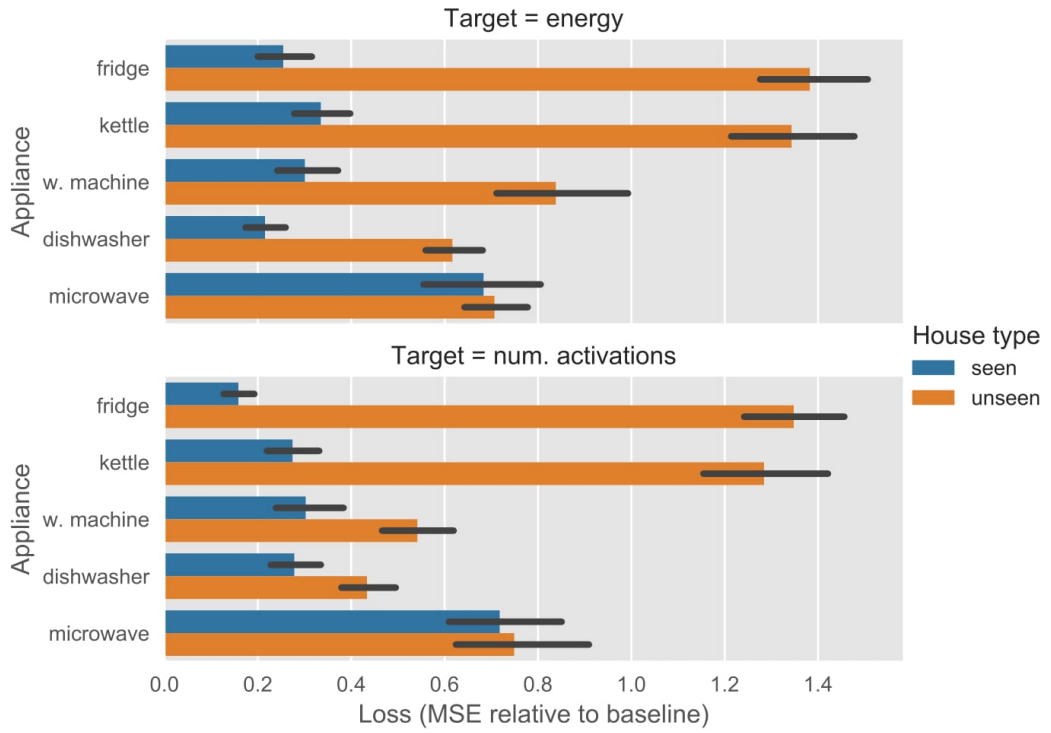

Figure 4.8: Test error for best-performing single-target models when models were not trained with synthetic data. The error bars represent 95% confidence intervals.

Outside of the fridge and the microwave, the generalization error is higher for all appliances and for both target variables. The most dramatic difference is with the kettle, where the error increases from 0.76 to 1.34 (around $+76\%$) for energy and 0.69 to 1.28 ($+86\%$) for the number of activations—meaning the kettle models lose their ability to generalize to the houses held out for the test set in this project.[4] This contrasts with performance on *seen* houses, where error decreases from 0.72 to 0.33 (-54%) for energy and 0.50 to 0.27 (-46%) for the number of activations. While less dramatic, the dishwasher models have a similar pattern for seen and unseen houses.

This shows that the inclusion of the synthetic data helps performance on unseen houses—but sometimes at the expense of performance on seen houses. However, this

[4]Here we ignore errors bars and just use the point estimates for simplicity.

		Loss (MSE relative to baseline)	
Appliance	House type	Energy	Num. activations
fridge	seen	0.25 ± 0.06	0.16 ± 0.03
	unseen	1.38 ± 0.12	1.35 ± 0.11
kettle	seen	0.33 ± 0.06	0.27 ± 0.06
	unseen	1.34 ± 0.13	1.28 ± 0.13
w. machine	seen	0.30 ± 0.07	0.30 ± 0.07
	unseen	0.84 ± 0.14	0.54 ± 0.08
dishwasher	seen	0.22 ± 0.04	0.28 ± 0.06
	unseen	0.62 ± 0.06	0.43 ± 0.06
microwave	seen	0.68 ± 0.13	0.72 ± 0.13
	unseen	0.71 ± 0.07	0.75 ± 0.15

Table 4.2: Test MSE relative to the baseline (with 95% confidence intervals) when models were not trained with synthetic data.

hit to performance on seen houses is not an issue since we care about generalization performance for the application domain.

4.4 Exploration of convolutional layer activations

Although neural networks are complex and often considered to be black boxes, there are some steps we can take to explore the networks' internal representations. One such step is visualizing the "activations"—or the values taken by the units—of the convolutional layers.[5]

The activations can be visualized in two main ways. The first is to define a loss function that maximizes the activations of the convolutional layer, and then use gradient ascent to generate patterns that cause the activations to fire most strongly [41]. The second is to feed the convolutional neural network an input, and then visualize the activations of the convolutional layer caused by that input.[6] We do the second, since it gives a better sense of how the networks perform on real input data.

Visualizing the activations of the first convolutional layers is likely not very infor-

[5]This is unfortunate terminology. The activations of hidden layers relates to the activation functions described in Section 2.1 and not the activations of appliances.

[6]This is equivalent to removing all layers after the convolutional layer of interest, and then simply performing prediction using forward propagation on the truncated network.

mative, since many models are sure to learn similar low-level features: an increase in the time series, a decrease, a spike, and so on. We therefore visualize the highest-level activations, which are part of the last convolutional layer. Since there are often many filters in this final convolutional layer[7]—with many of them sparse and somewhat redundant—we use principal component analysis (PCA)[8] to reduce the number of dimensions of the activations to ten.[9]

To start, we choose a house and day where all the target appliances are used (Figure 4.9). Outside of large spikes caused by the electric shower at around 7:00 or 8:00,[10] it is a fairly clean day of data with few anomalies. We then iterate through each appliance, load the energy prediction model for that appliance, input the aggregate signal for the day into the model, and visualize the activations of the final convolutional layer as the model makes its prediction.[11] Note again that the models do *not* see the signals of the appliances when making predictions.

We find that the washing machine energy model seems to be "looking" in the right places, correctly identifying the times of the day where the washing machine is used (Figure 4.10). The dishwasher energy model also correctly identifies times of day where its target appliance is used (Figure 4.11). The kettle model appears to fire for spikes of any kind, including spikes created by the kettle (Figure 4.12). The larger the spike, the larger the activations.

Meanwhile, it is unclear what information is being used by the microwave, fridge and multi-target models to make predictions (Figures 4.13, 4.14 and 4.15). In particular, the activations of the microwave model do not seem fire when the microwave is used. (Regardless, its prediction for the day is accurate.) Since the hidden layers of the multi-target model are shared by all target appliances, it makes sense that the activations do not have obvious associations with particular target appliances.

[7]Due to the fact that the number of filters doubles each convolutional layer, there can be up to $512 = 8 \times 2^{7-1}$ filters in the final convolutional layer if there are 7 convolutional layers and 8 starting filters.

[8]Roughly speaking, principal component analysis is a linear dimensionality reduction technique where the data is projected onto a lower-dimensional space in a way that retains as much of the original variance as possible.

[9]The top row in each visualization will represent the first principal component, meaning that it accounts for the most variance in the activations. The second row is orthogonal to the first and accounts for the second-largest amount of variance. And so on for the third through the tenth rows.

[10]Electric showers are not captured using IAMs, so we do not know for sure that this is its signature. However, it is a safe assumption given the time of day, the signature pattern and the usual power range for electric showers (roughly 9.5–11.0 kW).

[11]Energy is chosen over the number of activations as the target variable simply because the activations models have too few dimensions in the final convolutional layer, making the visualization of the activations less informative.

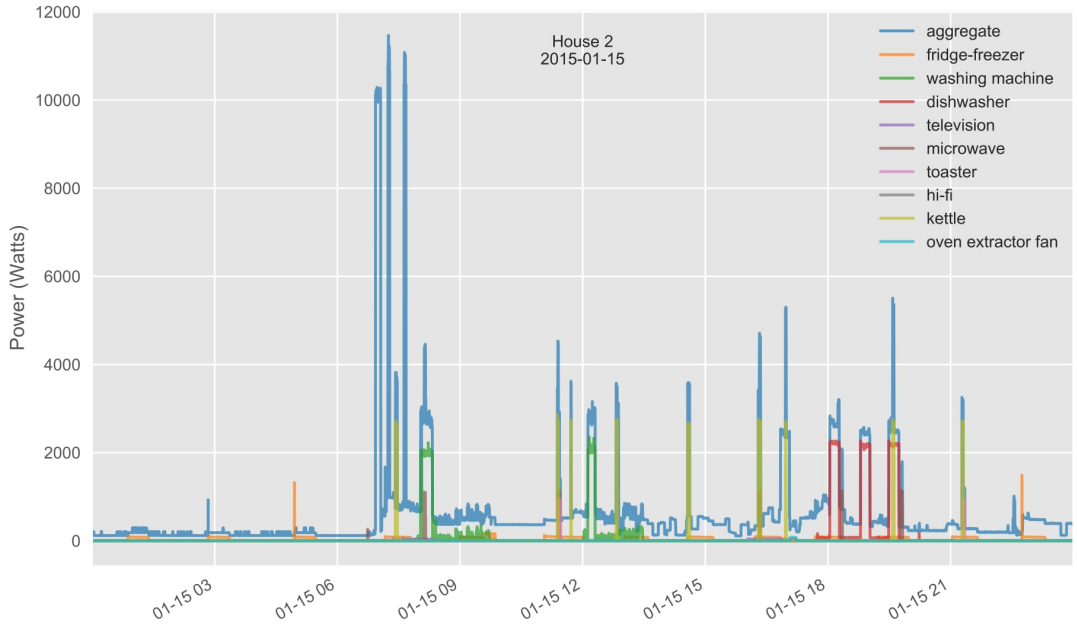

Figure 4.9: Example day of data where all target appliances are used.

It should be made clear that this type of analysis helps to understand where in the day the models are focusing their attention, but it does *not* say how the neural network processes these activations downstream in the dense layers. Furthermore, when a model activates during parts of the day when the target appliance is not being used, we cannot say whether the model is "distracted" or if the model is incorporating cross-appliance information into its predictions. An example of this is when the dishwasher model activates in the morning and at noon when the washing machine is being used.

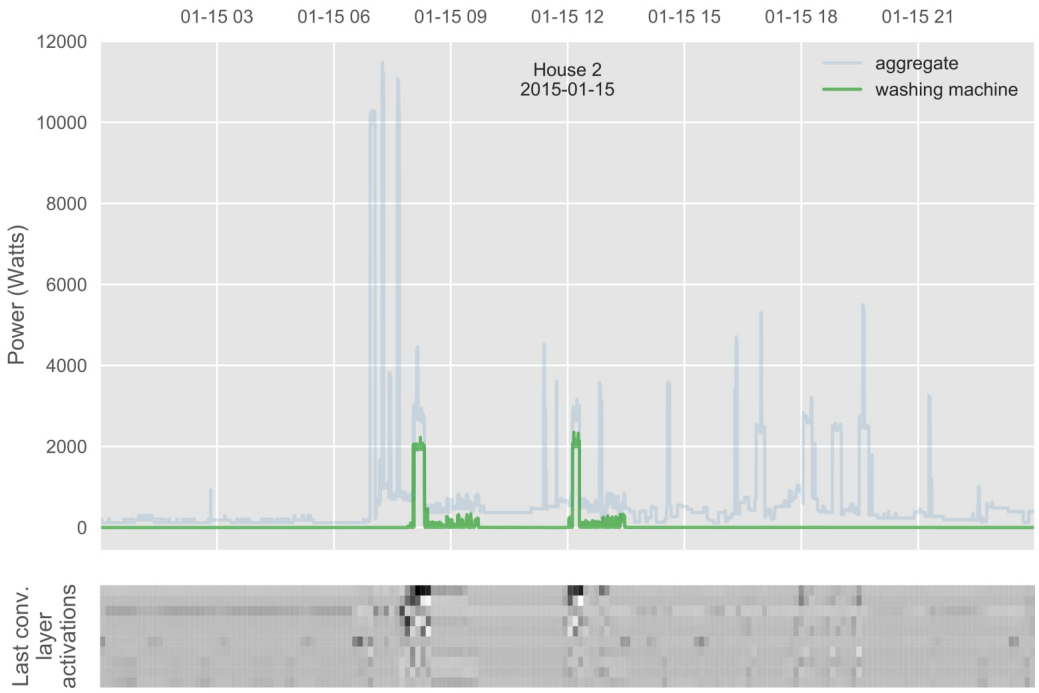

Figure 4.10: Dimensionality-reduced activations of the final convolutional layer of the washing machine model. The target energy is 1.19 kWh and the prediction is 1.11.

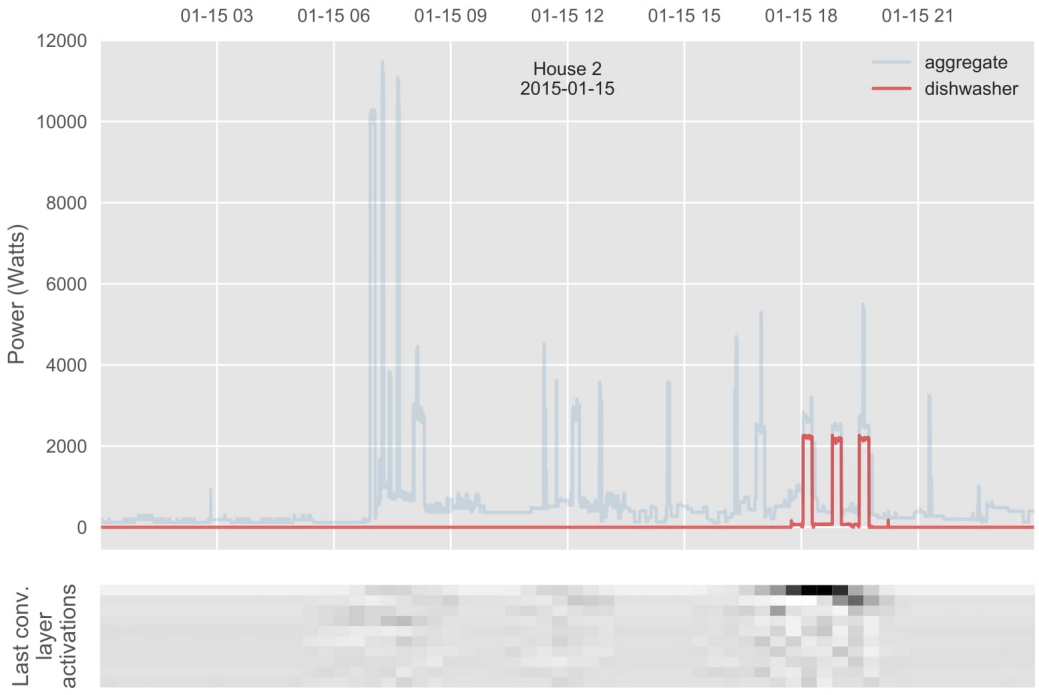

Figure 4.11: Dimensionality-reduced activations of the final convolutional layer of the dishwasher model. The target energy is 1.64 kWh and the prediction is 1.47.

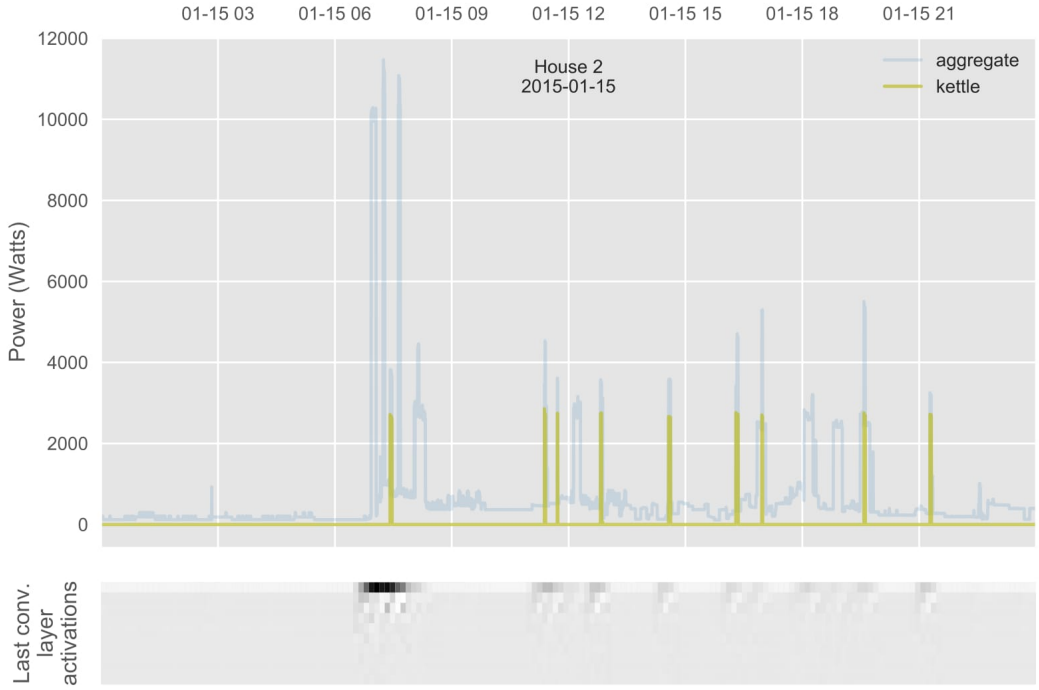

Figure 4.12: Dimensionality-reduced activations of the final convolutional layer of the kettle model. The target energy is 0.83 kWh and the prediction is 0.51.

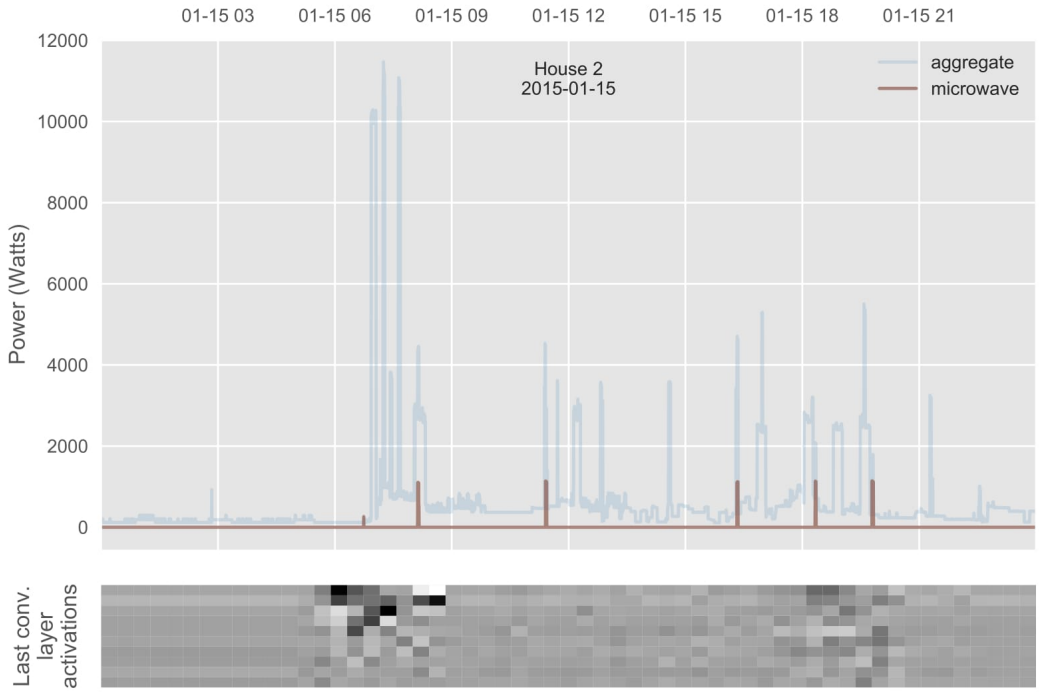

Figure 4.13: Dimensionality-reduced activations of the final convolutional layer of the microwave model. The target energy is 0.12 kWh and the prediction is 0.15.

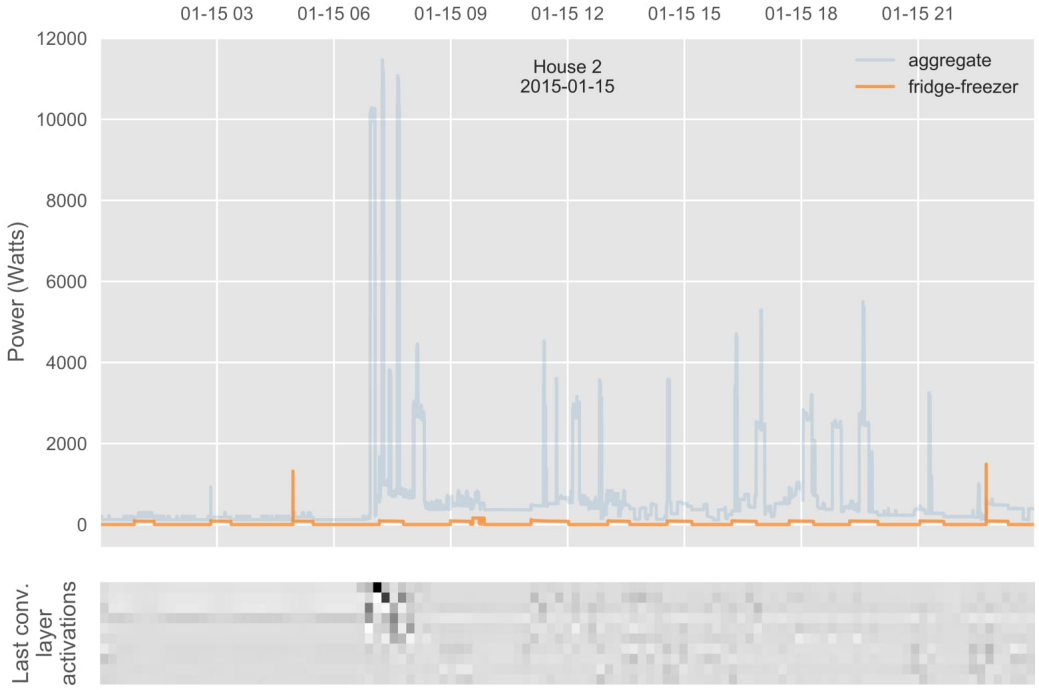

Figure 4.14: Dimensionality-reduced activations of the final convolutional layer of the fridge model. The target energy is 0.75 kWh and the prediction is 0.45.

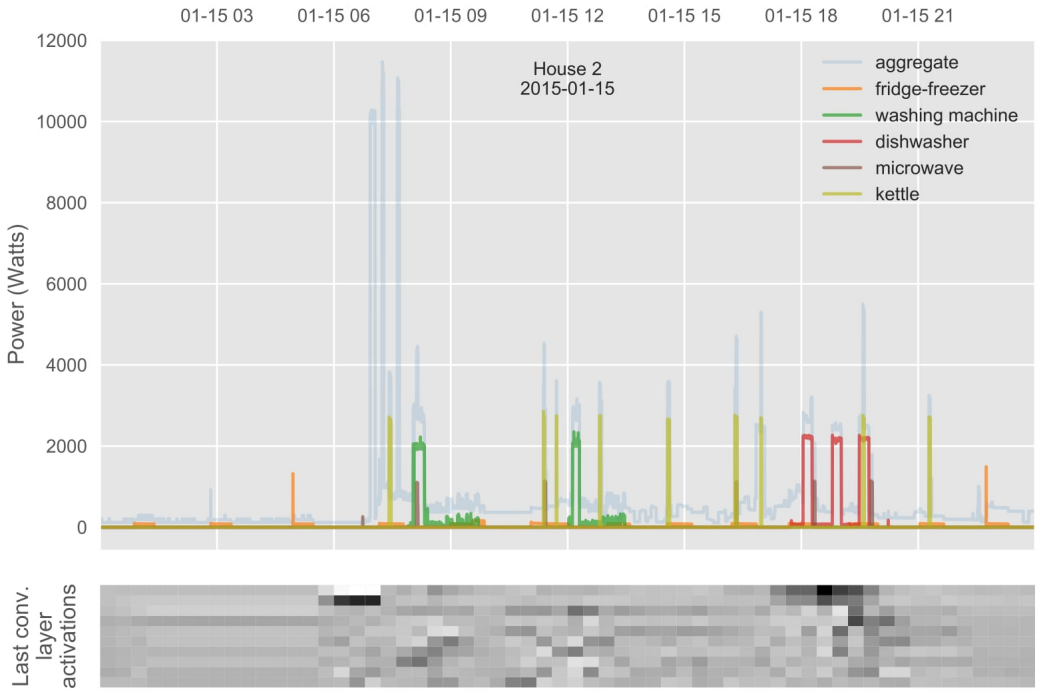

Figure 4.15: Dimensionality-reduced activations of the final convolutional layer of the multi-target model.

Chapter 5

Conclusion

5.1 Remarks and observations

In this project, neural networks are used to predict two target variables—energy and the number of activations—for five appliances. The results are quite promising. We find that neural networks are not only capable of summarizing energy usage in principle, but can also generalize to houses that were not seen during training and validation. This generalization is crucial for the application domain.

Nine of the twelve models have better generalization performance than the baseline. This means that the output statistics from these models would be better priors than national statistics for the AFHMM with LBM models. The neural networks perform best on appliances with complex signatures like the washing machine and the dishwasher, where the generalization error when predicting the number of activations is less than half the baseline error. The fridge models, on the other hand, generalize very poorly, having a generalization error that is worse than the baseline. The multi-target models also do not generalize as well as the single-target models.

The data augmentation process proves important to the success of the models for all appliances except for the fridge,[1] improving generalization performance by quite a bit. Without training on synthetic data, the kettle models do not generalize to unseen houses at all.

The predictions of some models are arguably strong enough to provide directly to consumers. For example, accuracies for the number of activations are 83% for the washing machine and 88% for the dishwasher.[2] However, there needs to be more

[1] The predictions for the fridge were poorer than baseline performance when the models were trained with and without synthetic data.

[2] This is calculated using the values in the confusion matrices in Section 4.2.3, where the tails of the

research to determine whether this accuracy is sufficient for changing behavior.

An exploration of the convolutional layer activations show that the washing machine, dishwasher and kettle models are using the target appliance signatures to a large degree when making predictions—which is desirable for generalization. On the other hand, it is unclear what information the fridge and microwave models are using to make predictions.

5.2 Drawbacks and limitations

Below are the known limitations in the project:

- **Small training set.** Although REFIT is large when compared to similar datasets, it is not as large as datasets typically used to train neural networks—especially since we are aggregating at the daily level. This means we only have around 3,100 real training examples and 40,000 synthetic training examples. Additionally, there are only 15–22 series for each type of target appliance.

- **Limited validation process.** Given the relative lack of data and the large number of models that needed to be trained through the random search process, we do not use cross-validation. Ideally, we would iteratively validate on each house while training on the others. The small number of houses in the validation holdout set leads to sensitivity in the validation error—and likely poorer results in general.

- **Sensitive test results.** Like with the validation set, we hold out only a few houses to test generalization error. This leads to sensitivity in the reported results.

- **Some models not using target appliance signature information.** The fridge model found it difficult to differentiate between between-house variance and within-house variance. It is possible that this could be remedied with more data and improving the methods that encourage generalization (e.g., further improving the data augmentation process so that the models do not learn to guess the house).

- **Multi-target models not flexible enough.** It is possible that the multi-target models do not perform well because they are not flexible enough (i.e., not enough hidden layers or filters), or because they do not have task-specific hidden layers.

distributions for the number of activations were combined to be one value.

5.3 Further work

Below are avenues for future research:

- **Address limitations discussed in Section 5.2.** This can mean using cross-validation, incorporating more data (perhaps from other studies), making the multi-target models more flexible, or further improving the data augmentation process.

- **Further optimize the neural networks.** Hyperparameter search explored a relatively simple architecture design with a broad range of possible hyperparameters. We can try more sophisticated architectures, or we can further explore the range of hyperparameters that work well on the simple architecture design. We can also try a more sophisticated procedure to search for hyperparameters, like Baysian optimization.

- **Compare summarization performance to NILM.** We can run NILM models like PointNet on the aggregate signals used in this project's test set to predict timepoint-by-timepoint signatures of the target appliances. We can then compute the summary statistics based on the predicted signals and compare the results to those in this project.[3]

- **Test results of AFHMM with LBM using improved priors.** We can run AFHMM with LBM using the improved priors and see to what degree the results are improved. Given that simply using national averages as priors reduces errors by 50% in some cases, it is possible that improved priors can lead to additional significant gains.

- **Test effectiveness of feedback accuracy.** This is more closely related to behavioral economics or psychology than machine learning, but we can explore how accurate feedback must be in order to inspire behavioral change. For example, when the dishwasher is used once or more, our model gets this right 85% of the time. This sounds good, but to what degree would the false negatives (where the model predicts no activations) inspire consumer distrust in the feedback? What about false positives? How should uncertainty in the predictions be communicated on consumers? Answering these types of questions can help to identify performance benchmarks for future research.

[3] However, it is not clear how to compute the number of activations using NILM, since `Electric.get_activations()` may be ineffective on noisy predicted signals.

Appendix A

Data issues

A.1 Known data issues

The data issues known before the start of this project are described below:

- **Aggregate signal measurement error.** The device used to measure the aggregate power signal (a CurrentCost transmitter) has a relative error of around 6%.
- **Solar panel interference.** The CurrentCost transmitter is unable to distinguish the direction of power flow, so solar panel generation for houses 1,[1] 11 and 21 register as positive power [3]. This results in a characteristic bell-shaped curve around midday that overwhelms other signals (Figure A.1). The aggregate signals for these houses are therefore not used in training, validation or testing. However, the individual appliance signals are used when creating synthetic training data (Section 3.6.4).
- **Meters not synchronized.** The readings of IAMs are not synchronized in time with the those of the aggregate signal. Specifically, the timestamp of any change in value in an IAM is seconds before the associated change in aggregate signal. This means that subtracting an appliance signal from the aggregate signal would leave spiky artifacts.
- **Disagreeing power step changes.** Changes in value between IAMs and the aggregate signal do not correspond.
- **Disagreeing power levels.** The sum of the IAM readings are sometimes higher than the aggregate reading as indicated by the Issues column in the CSV files.

[1] See Appendix A.2 on a discrepancy regarding solar panels on House 3.

- **Gaps in the data.** There are large gaps in the data due to internet outages and other technical problems. This is most notable in February 2014 (reminder: Figure 3.1). The average uptime for houses is 88% on average [3].
- **Unrecorded energy usage.** Only up to 55% of aggregate energy is captured by the IAMs [3].[2] This has one important implication for model performance: If there exists a target appliance that is not connected to an IAM, then we could be punishing our models for correctly detecting the signature of this target appliance in the aggregate signal.
- **Switched appliances.** There is "noncompliance" by the participants of the study in that they sometimes switch which appliances were connected to which IAMs. However, for the cleaned dataset, the researchers who headed the study in seem to have created series that stitch together the correct power series [10].

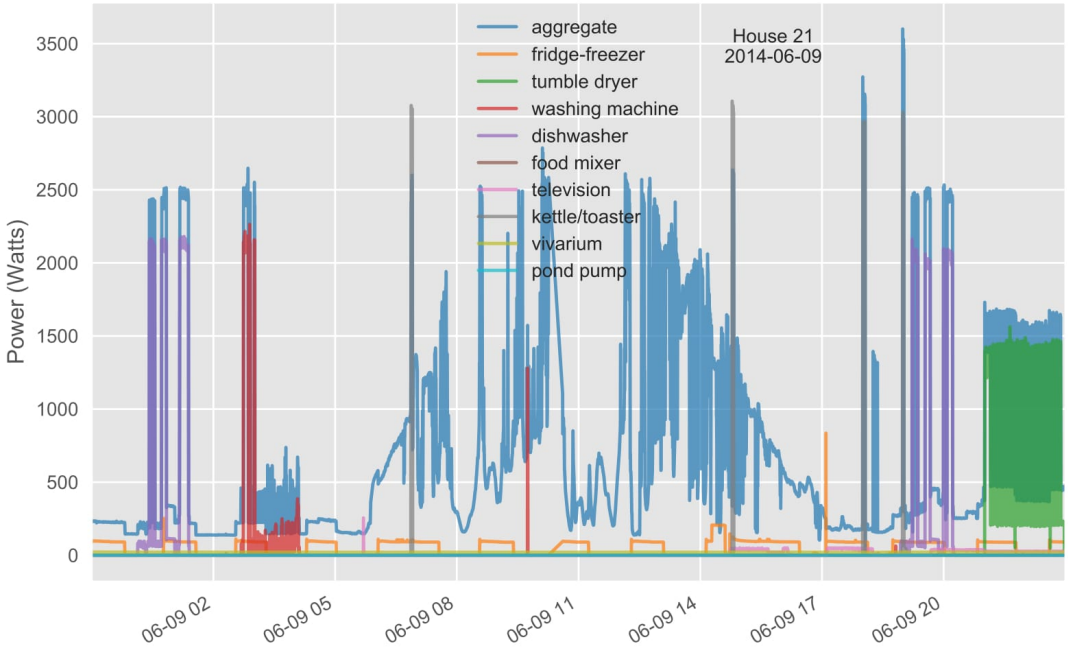

Figure A.1: Houses with solar panels had a characteristic bell-shaped curve.

A.2 Discovered data issues

In addition to the known data issues, some were discovered through additional data exploration:

[2] Lighting itself is 16% of electricity use [42].

- **Incorrect appliance labels.** There are discrepancies between the appliance column names in the raw and cleaned README files. Visualizing the signals seems to suggest that the labels from the raw README are correct.[3] In total, there are ten labeling discrepancies between the raw and cleaned README files that affected our target variables.

- **Strings of large, repeating power values (Figure A.4).** This bizarre data error is found in both aggregate series and appliance series.

- **Low correlation between IAMs and aggregate (Figure A.2).** This is caused by appliances that are not connected to IAMs, IAM signals that do not seem register in the aggregate, and other issues.

- **Varying daily recordings.** That is, the number of recordings vary quite a bit across houses and days (Figure A.3). This is not necessarily a mistake, but it needs to be taken into account when preparing the data for modeling.

- **Incorrectly identified solar interference.** Although it is stated that House 3 had solar panel interference [3], it actually seems to be House 1.

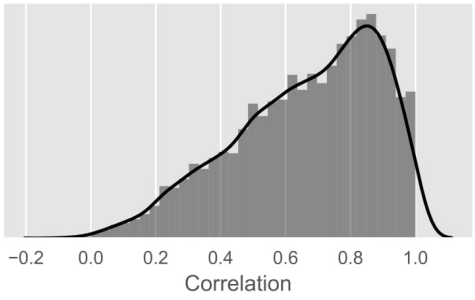

Figure A.2: Distribution of daily correlation between sum of IAMs and aggregate.

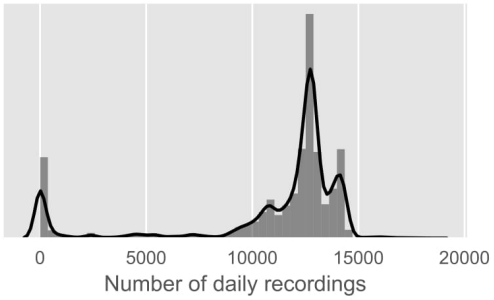

Figure A.3: Distribution of number of daily recordings.

[3]For example, Appliance 4 for House 12 is labeled as a computer site when the signature resembles a kettle—which is the label in the raw README.

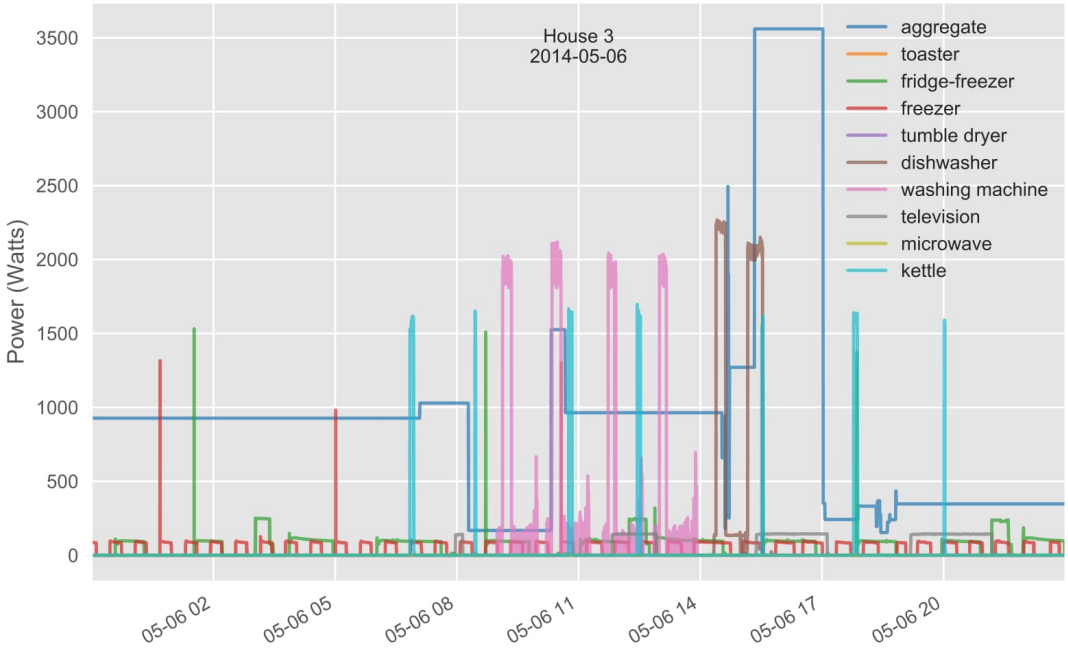

Figure A.4: An example of a day with large, repeated power values for one of the power series.

Appendix B

Additional algorithms

Data: (1) `unaligned`, the list of ordered unaligned values; (2) `standard`, the list of ordered values to align to; (3) `padder`, the index value to use when the starting value(s) of `align` are greater than the first value of `standard` (default `0` in this project)
Result: `idx`, the list of indices of `unaligned` that align it with the values of `standard`

```
1  u = 0
2  idx = []
3  for s in range(len(standard)) do
4  |  while u < len(unaligned) and unaligned[u] <=
   |   standard[s] do
5  |  |  u += 1
6  |  end
7  |  idx.append(u-1)
8  end
9  for i in range(len(standard)) do
10 |  if idx[i]==-1 then
11 |  |  idx[i] = padder
12 |  else
13 |  |  break
14 |  end
15 end
16 return idx
```

Algorithm 2: Python-style pseudo-code that describes how timestamps are standardized. It returns the indices `idx` of `unaligned` such that `unaligned[idx[i]]` is that maximum value of `unaligned` for which `unaligned[idx[i]] <= standard[i]`, for all `i`. `idx` can then be used to standardize power series that have unaligned timestamps `unaligned`.

Data: (1) List of houses; (2) list of target appliances.
Result: (1) $X_{N\times D}$ feature matrix where N is the number of training samples and D is the standardized number of timesteps in a day; (2) $Y_{N\times K}$ feature matrix where K is the number of target variables (one for each target appliance).

```
Initialize X and Y as empty matrices
foreach house do
    Get list of all dates with viable data for the house
    foreach date do
        Load x, the aggregate series for house/date
        Standardize number of timesteps of x
        Initialize y, vector of target variables, as empty list
        foreach target appliance do
            Calculate y, energy used by the target appliance, from x
            Append y to y
        end
        Append x to X as row
        Append y to Y as row
    end
end
return X, Y
```

Algorithm 3: Creation of data using real aggregate signals, which is used in training, validation and testing. For simplicity, this algorithm only shows the calculation of the energy target variables. In the actual code, there are two Y matrices (and consequently two $\boldsymbol{y}$, y_a, y_s, and y_d)—one for energy and one for number of activations. Also excluded are additional vectors that are returned that specify the house and date of each observation.

Data: (1) List of houses that have *not* been held out for validation and testing; (2) list of target appliances; (3) list of dates that have *not* been held out for validation and testing.
Result: (1) $X_{N\times D}$ feature matrix where N is the number of training samples and D is the standardized number of timesteps in a day; (2) $Y_{N\times K}$ feature matrix where K is the number of target variables (one for each target appliance).

Initialize X and Y as empty matrices
foreach *house* **do**
  foreach *date* **do**
    if *date is not viable for house* **then**
      continue to next date
    end
    Initialize $\boldsymbol{x}$, synthetic aggregate series, with zeros
    Initialize $\boldsymbol{y}$, vector of target variables, as empty list
    foreach *target appliance* **do**
      Initialize $\boldsymbol{x}_a$, power series for the target appliance, with zeros
      Initialize y_a, target variable used by target appliance, as zero
      foreach *power series in house with target appliance* **do**
        if *swap signal (with* $p = 0.5$*)* **then**
          Load $\boldsymbol{x}_s$, a power series for the target appliance for a random house/date
        else
          Load $\boldsymbol{x}_s$, the power series for the target appliance for the current house/date
        end
        Calculate y_s, energy used in the target appliance power series, from $\boldsymbol{x}_s$
        Standardize number of timesteps of $\boldsymbol{x}_s$
        $\boldsymbol{x}_a \leftarrow \boldsymbol{x}_a + \boldsymbol{x}_s$
        $y_a \leftarrow y_a + y_s$
      end
      $\boldsymbol{x} \leftarrow \boldsymbol{x} + \boldsymbol{x}_a$
      Append y_a to $\boldsymbol{y}$
    end
    foreach *distractor appliance* **do**
      if *include distractor appliance (with* $p = 0.5$*)* **then**
        Load $\boldsymbol{x}_d$, the power series for the distrctor appliance for the current house/date
        Standardize number of timesteps of $\boldsymbol{x}_d$
        $\boldsymbol{x} \leftarrow \boldsymbol{x} + \boldsymbol{x}_d$
      end
    end
    Append $\boldsymbol{x}$ to X as row
    Append $\boldsymbol{y}$ to Y as row
  end
end
return X, Y

Algorithm 4: Creation of synthetic training data. For simplicity, this algorithm only shows the calculation of the energy target variables. In the actual code, there are two Y matrices (and consequently two $\boldsymbol{y}$, y_a, y_s, and y_d)—one for energy and one for number of activations. Also excluded are additional vectors that are returned that specify the target house and date of each observation.

Appendix C

Additional evaluation data

		Loss (MAE)	
Appliance	House type	Energy	Num. activations
fridge	seen	0.24 ± 0.020	6.7 ± 0.52
	unseen	0.40 ± 0.019	14 ± 0.65
kettle	seen	0.20 ± 0.011	2.0 ± 0.15
	unseen	$0.21 \pm$ 9.4e-3	2.3 ± 0.15
w. machine	seen	0.17 ± 0.018	0.29 ± 0.026
	unseen	0.30 ± 0.025	0.37 ± 0.030
dishwasher	seen	0.16 ± 0.021	0.16 ± 0.022
	unseen	0.40 ± 0.028	0.33 ± 0.025
microwave	seen	$0.070 \pm$ 5.7e-3	1.6 ± 0.12
	unseen	$0.068 \pm$ 3.3e-3	1.7 ± 0.097

Table C.1: Test MAE loss with 95% confidence intervals. The units for energy are kWh, and the units for number of activations are counts.

		Loss (MAE)	
Appliance	House type	Energy	Num. activations
fridge	seen	0.20 ± 0.018	5.6 ± 0.47
	unseen	0.38 ± 0.019	13 ± 0.63
kettle	seen	$0.11 \pm 9.9\text{e-}3$	1.3 ± 0.12
	unseen	0.25 ± 0.014	3.3 ± 0.19
w. machine	seen	0.19 ± 0.020	0.29 ± 0.032
	unseen	0.37 ± 0.028	0.45 ± 0.037
dishwasher	seen	0.16 ± 0.020	0.14 ± 0.020
	unseen	0.56 ± 0.036	0.35 ± 0.026
microwave	seen	$0.072 \pm 6.0\text{e-}3$	1.6 ± 0.13
	unseen	$0.067 \pm 3.0\text{e-}3$	1.5 ± 0.094

Table C.2: Test MAE loss when trained only on synthetic data, with 95% confidence intervals. The units for energy are kWh, and the units for number of activations are counts.

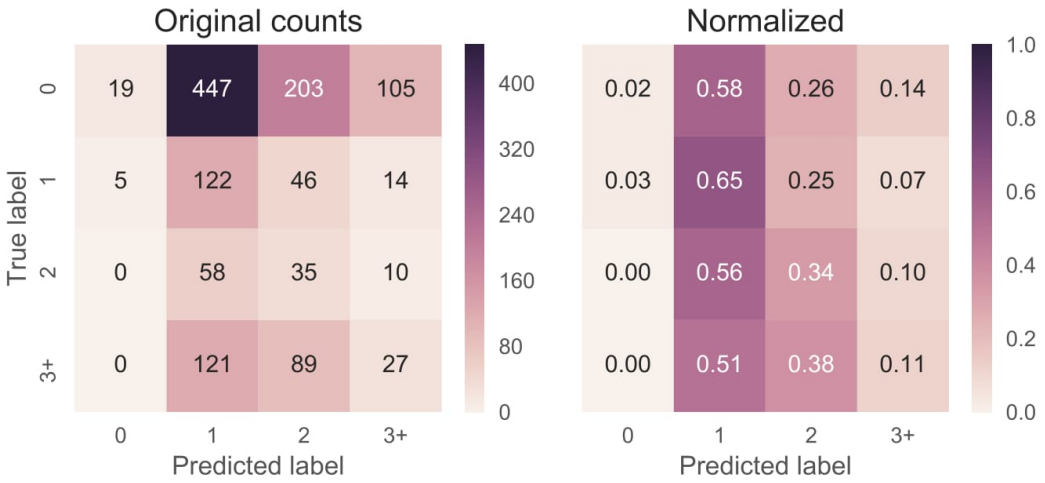

Figure C.1: Confusion matrix for microwave activations on unseen houses. Model predictions are rounded to the nearest integer.

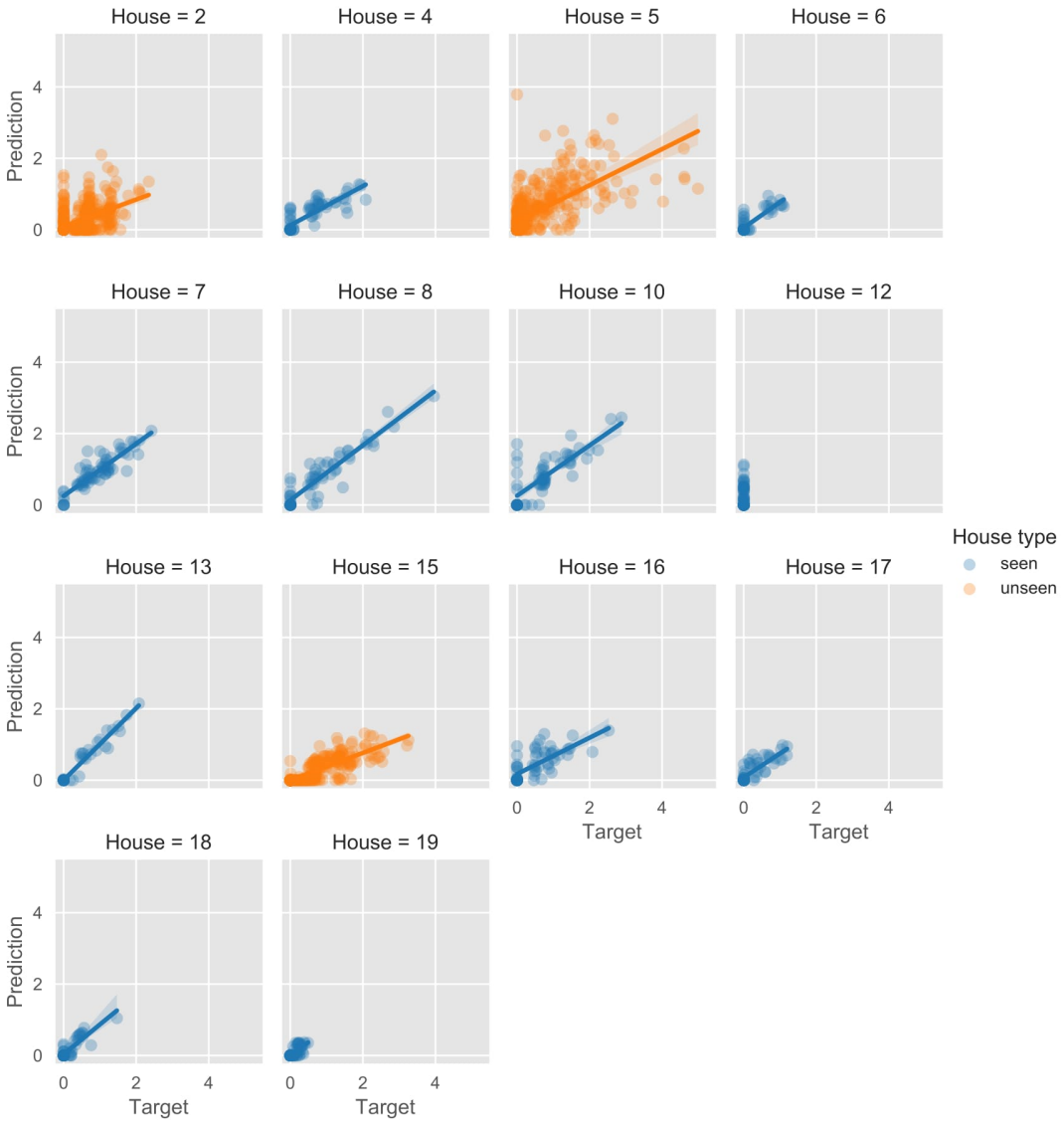

Figure C.2: Predictions vs. targets for washing machine energy. The points are jittered along the x-axis to reduce overlap.

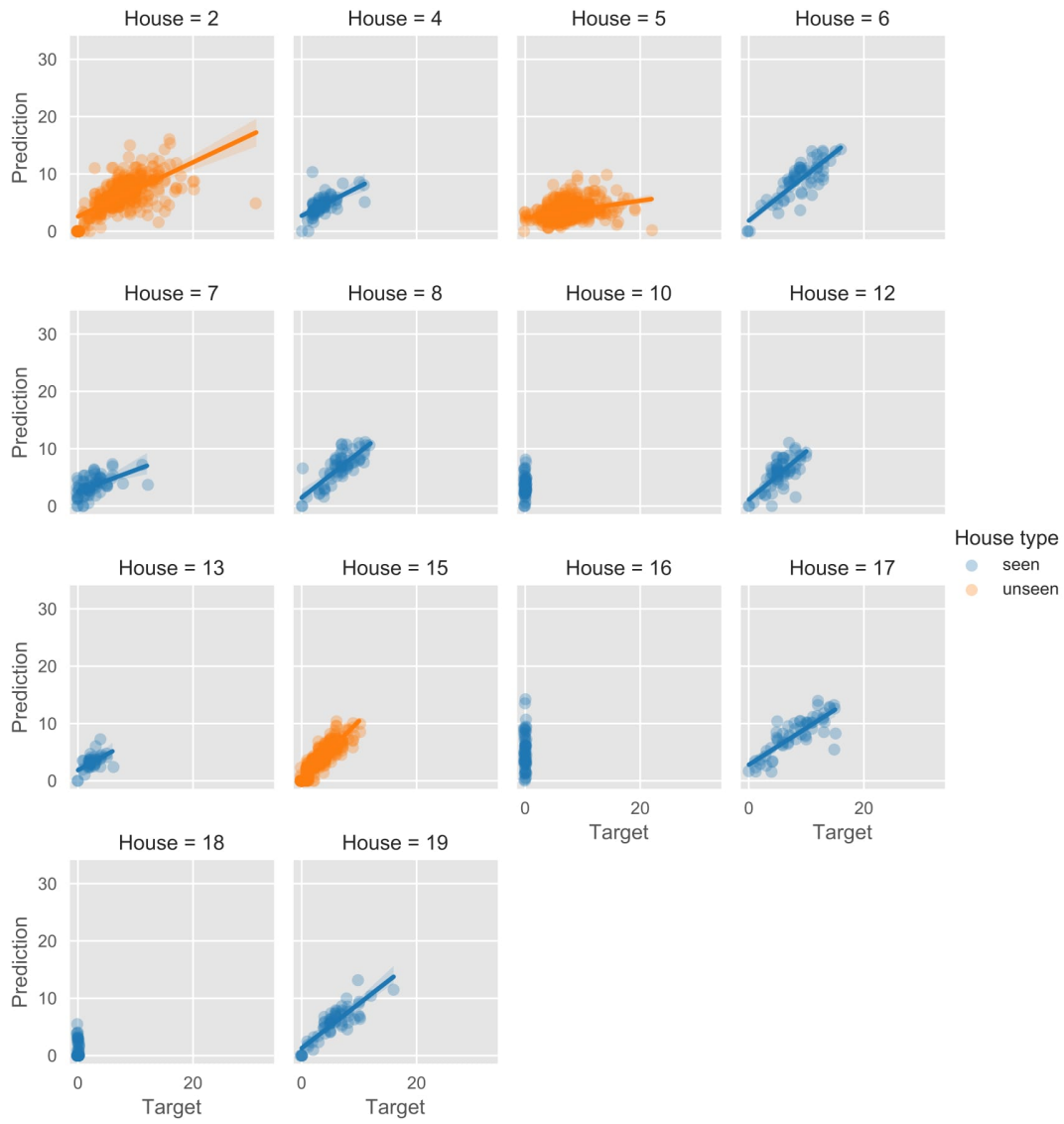

Figure C.3: Predictions vs. targets for kettle activations.

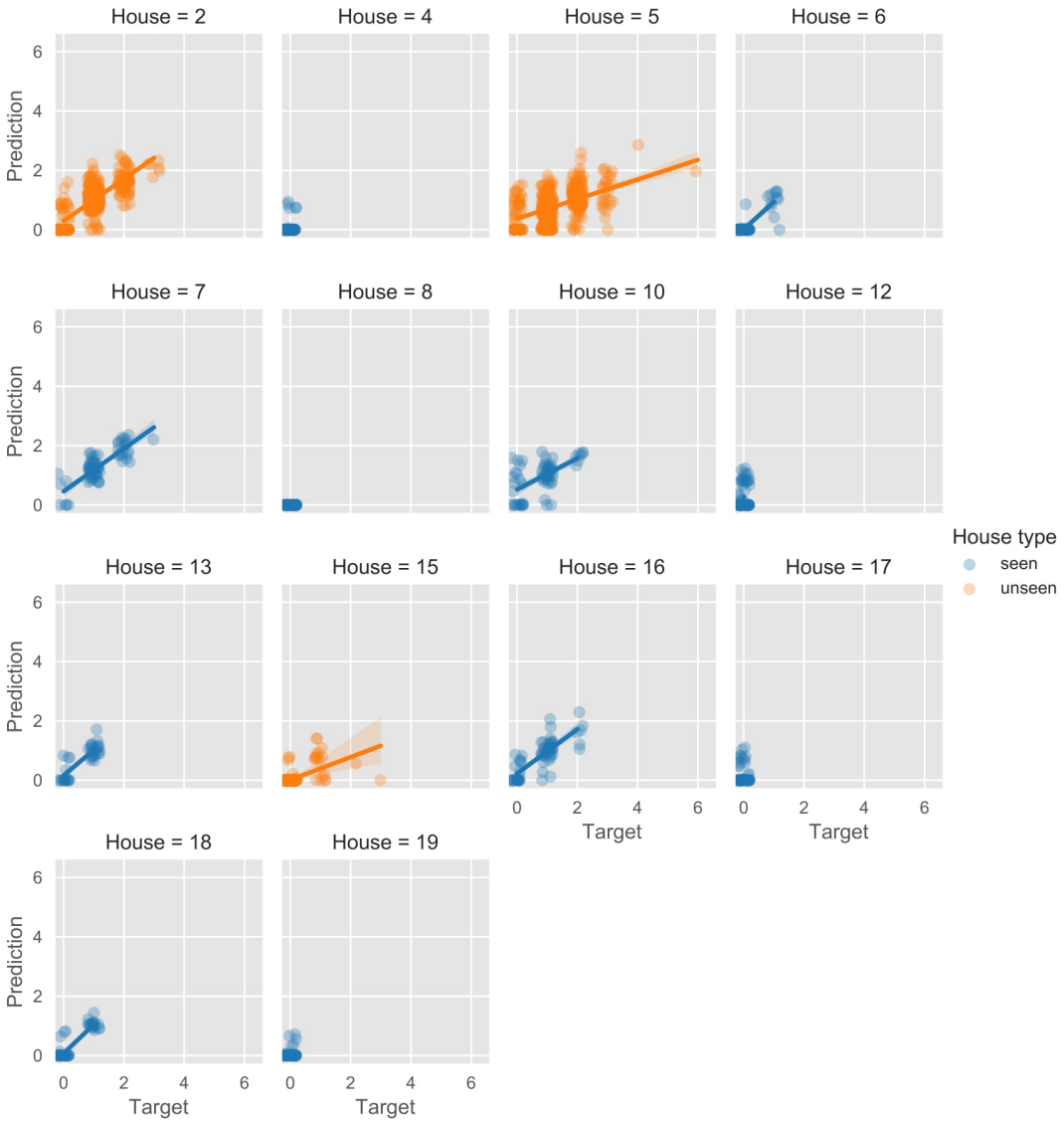

Figure C.4: Predictions vs. targets for dishwasher activations. The points are jittered along the x-axis to reduce overlap.

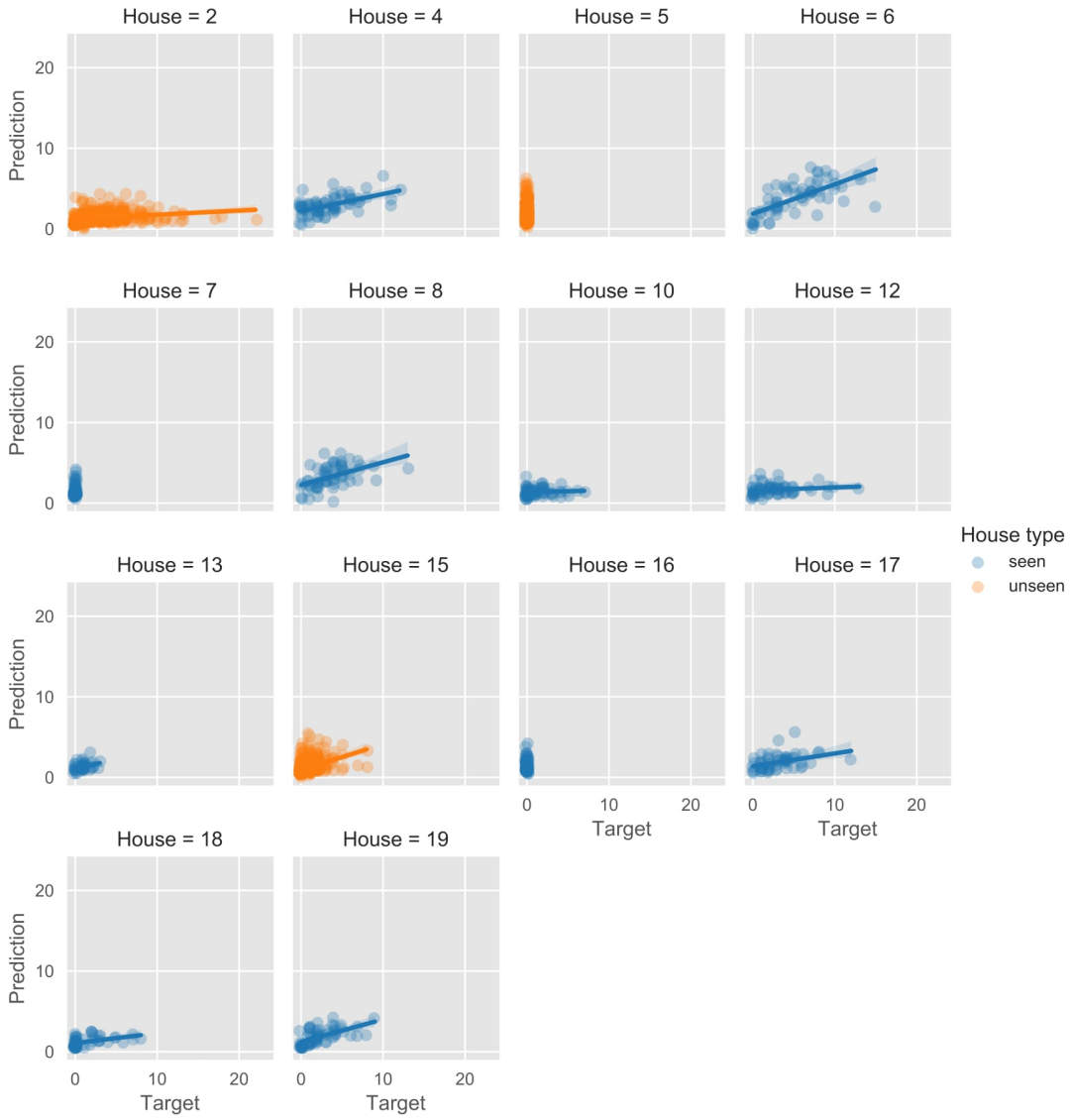

Figure C.5: Predictions vs. targets for microwave activations. The points are jittered along the x-axis to reduce overlap.

Bibliography

[1] J. J. Fei-Fei Li, Andrej Karpathy, "Cs231n: Convolutional neural networks for visual recognition." Stanford University course. Website available at http://cs231n.stanford.edu/. Accessed 13 Mar 2016.

[2] S. Renals, "Machine learning practical." University of Edinburgh course. Website available at http://www.inf.ed.ac.uk/teaching/courses/mlp/2016. Accessed 13 Mar 2016.

[3] 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, 2017.

[4] S. Ruder, "An overview of multi-task learning in deep neural networks," *arXiv preprint arXiv:1706.05098*, 2017.

[5] C. Zhang, M. Zhong, Z. Wang, N. Goddard, and C. Sutton, "Sequence-to-point learning with neural networks for nonintrusive load monitoring," *arXiv preprint arXiv:1612.09106*, 2016.

[6] 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*, pp. 55–64, ACM, 2015.

[7] C. Fischer, "Feedback on household electricity consumption: a tool for saving energy?," *Energy efficiency*, vol. 1, no. 1, pp. 79–104, 2008.

[8] "IDEAL home energy advice project." Website available at http://www.energyoracle.org/. Accessed 04 Apr 2017.

[9] C. Armel, "Energy disaggregation, precourt energy efficiency center," 2011.

[10] D. Murray, J. Liao, L. Stankovic, V. Stankovic, R. Hauxwell-Baldwin, C. Wilson, M. Coleman, T. Kane, and S. Firth, "A data management platform for personalised real-time energy feedback," *EU Science Hub*, 2015.

[11] M. Zhong, N. Goddard, and C. Sutton, "Latent bayesian melding for integrating individual and population models," in *Advances in Neural Information Processing Systems*, pp. 3618–3626, 2015.

[12] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[13] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014.

[14] Y. Zheng, Q. Liu, E. Chen, Y. Ge, and J. L. Zhao, "Time series classification using multi-channels deep convolutional neural networks," in *International Conference on Web-Age Information Management*, pp. 298–310, Springer, 2014.

[15] Z. Wang and T. Oates, "Encoding time series as images for visual inspection and classification using tiled convolutional neural networks," in *Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[16] J. C. B. Gamboa, "Deep learning for time-series analysis," *arXiv preprint arXiv:1701.01887*, 2017.

[17] G. E. Batista, X. Wang, and E. J. Keogh, "A complexity-invariant distance measure for time series," in *Proceedings of the 2011 SIAM International Conference on Data Mining*, pp. 699–710, SIAM, 2011.

[18] H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, and E. Keogh, "Querying and mining of time series data: experimental comparison of representations and distance measures," *Proceedings of the VLDB Endowment*, vol. 1, no. 2, pp. 1542–1552, 2008.

[19] V. Lempitsky and A. Zisserman, "Learning to count objects in images," in *Advances in Neural Information Processing Systems*, pp. 1324–1332, 2010.

[20] I. Stoianov and M. Zorzi, "Emergence of a'visual number sense'in hierarchical generative models," *Nature neuroscience*, vol. 15, no. 2, pp. 194–196, 2012.

[21] S. Seguí, O. Pujol, and J. Vitria, "Learning to count with deep object features," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 90–96, 2015.

[22] J. Z. Kolter and T. Jaakkola, "Approximate inference in additive factorial hmms with application to energy disaggregation," in *Artificial Intelligence and Statistics*, pp. 1472–1482, 2012.

[23] R. Caruana, "Multitask learning," in *Learning to learn*, pp. 95–133, Springer, 1998.

[24] F. Chollet *et al.*, "Keras." `https://github.com/fchollet/keras`, 2015.

[25] J. Kelly and W. Knottenbelt, "Uk-dale: A dataset recording uk domestic appliance-level electricity demand and whole-house demand," *ArXiv e-prints*, vol. 59, 2014.

[26] J.-P. Zimmermann, M. Evans, J. Griggs, N. King, L. Harding, P. Roberts, and C. Evans, "Household electricity survey: A study of domestic electrical product usage," *Intertek Testing & Certification Ltd*, 2012.

[27] A. Filip, "Blued: A fully labeled public dataset for event-based non-intrusive load monitoring research," in *2nd Workshop on Data Mining Applications in Sustainability (SustKDD)*, p. 2012, 2011.

[28] J. Z. Kolter and M. J. Johnson, "Redd: A public data set for energy disaggregation research," in *Workshop on Data Mining Applications in Sustainability (SIGKDD), San Diego, CA*, vol. 25, pp. 59–62, 2011.

[29] "Refit: Smart homes and energy demand reduction." Website available at http://www.refitsmarthomes.org/. Accessed 2 Apr 2017.

[30] A. Ng, "Machine learning." Stanford University course. Website available at https://www.coursera.org/learn/machine-learning. Accessed 1 Jul 2017.

[31] P. Y. Simard, D. Steinkraus, J. C. Platt, *et al.*, "Best practices for convolutional neural networks applied to visual document analysis.," in *ICDAR*, vol. 3, pp. 958–962, Citeseer, 2003.

[32] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[33] N. M. Nawi, W. H. Atomi, and M. Z. Rehman, "The effect of data pre-processing on optimized training of artificial neural networks," *Procedia Technology*, vol. 11, pp. 32–39, 2013.

[34] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, vol. 13, no. Feb, pp. 281–305, 2012.

[35] D. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," *CoRR*, vol. abs/1511.07289, 2015.

[36] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," *CoRR abs/1609.03499*, 2016.

[37] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting.," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[38] F. Chollet, "mnist_cnn.py." GitHub. Website available at https://github.com/fchollet/keras/blob/master/examples/mnist_cnn.py. Accessed 1 Jul 2017.

[39] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015.

[40] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[41] F. Chollet, "How convolutional neural networks see the world." Keras blog. Website available at https://blog.keras.io/how-convolutional-neural-networks-see-the-world.html. Accessed 1 Jul 2017.

[42] L. Stankovic, V. Stankovic, J. Liao, and C. Wilson, "Measuring the energy intensity of domestic activities from smart meter data," *Applied Energy*, vol. 183, pp. 1565–1580, 2016.