ΧΑΡΟΚΟΠΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ
HAROKOPIO UNIVERSITY

Artificial Intelligence and Applications in the Internet of Things

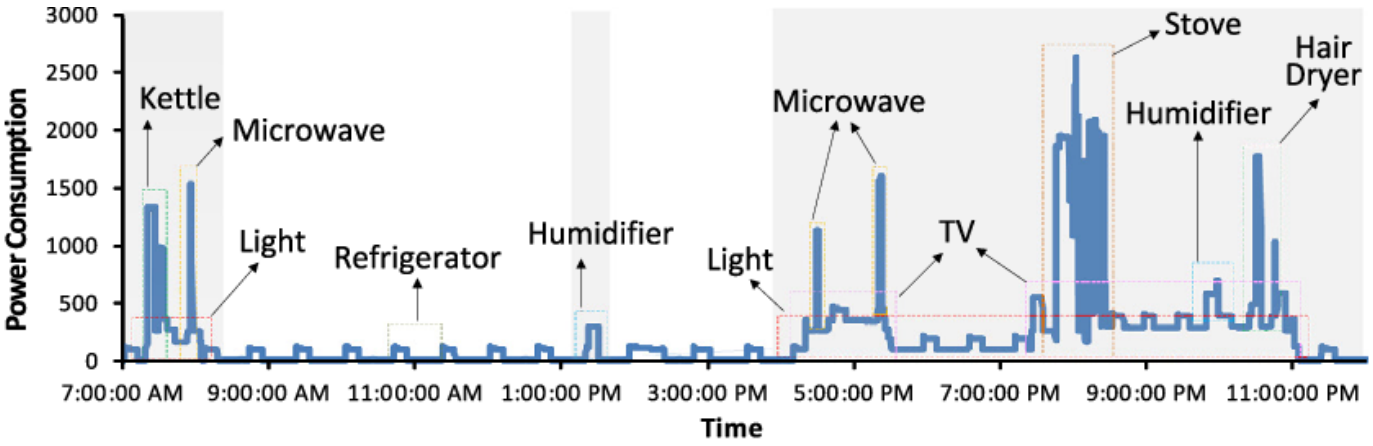
SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF INFORMATICS AND TELEMATICS

POSTGRADUATE STUDIES

WEB ENGINEERING

ENERGY DISAGGREGATION



Georgara Vasiliki 21107

Tserpe Paraskevi 21117

Professor:

Christos Diou, Assistant Professor in the area of Artificial Intelligence and Machine Learning

ATHENS

MAY 2022

Table of Contents

1. Introduction

End-use appliance energy conservation and real-time control are necessary owing to the change in dwelling structures brought about by urbanization and expansion brought about by the development of high-rise buildings worldwide. Smart meters, which made it possible to estimate appliance-specific power use from a building's overall power consumption data, also contributed to this trend. Using a single meter that monitors the total demand from several appliances, non-intrusive load monitoring (NILM) or energy disaggregation is a computational approach for calculating the power demand of individual appliances.

The generation of itemized power bills from a single, whole-home smart meter is one use-case. Energy efficiency regulations may benefit energy providers and users globally, resulting in social, environmental, and economic benefits. The end goal may be to assist consumers in reducing their energy use, assist grid operators in managing the grid, locate malfunctioning equipment, or conduct a survey of appliance usage patterns.

1.1. NILM benefits

According to a study, participants were able to cut their energy usage by an average of 7.7%, which is an outstanding result given that it was only the consequence of behavior change and not the use of smart thermostats or other automatic energy-saving technologies. It is beneficial to have a thorough understanding of your energy usage profile broken down into energy appliances in real-time. Knowing how the electrical load is operating can help you determine ways to lower your monthly power costs. Observing and evaluating may undoubtedly help minimize redundant energy consumption.

More flexible grid management is offered by NILM, which is greatly beneficial for incorporating more renewable energy, which is advancing and becoming more widely used. In order to stand out in the cutthroat deregulated retail markets, retail power suppliers are more particularly searching for creative ways to increase consumer interaction. Additionally, it can even develop new income streams based on customer wants that may be determined even more clearly from their consumption patterns. The demand from retail suppliers is rising since it is obvious that NILM is the best option for this use. On the other hand, utilities also view NILM as a useful demand management solution since it enables utilities to obtain real-time granular consumption data and to carry out load-shaping measures, such as tailoring Time-of-Use tariffs for each user based on their individual usage patterns.

Last but not least, it contributes to the electrical fault finding by analyzing the appliance’s signature on the aggregate energy waveform of the household. Identifying electrical faults leads to the understanding on how to correct them, or ask for professional help.

Last but not least, it assists in locating electrical faults by examining the appliance's signature on the household's total energy waveform. Finding electrical problems helps you learn how to fix them or get expert assistance.

2. Related work

More than 20 years ago, Hart conducted the first study on the issue of energy disaggregation (Hart 1992). Since then, the topic has been mostly inactive until recently, when the widespread use of smart meters made a significant quantity of fine-grained electrical data available for algorithm creation and testing.

There are several methods for NILM that make use of classification algorithms like decision trees. The most recent research construct a neural network method to approach NILM. We took our cues from a very particular research by Jack Kelly and William Knottenbelt titled "Neural NILM: Deep Neural Networks Applied to Energy Disaggregation," which demonstrated excellent performance.

3. Dataset

For the purposes of training our model, we used a public data set, termed the Reference Energy Disaggregation Data Set (REDD). The data is specifically geared towards the task of energy disaggregation: determining the component devices from an aggregated electricity signal. REDD consists of whole-home and circuit/device specific electricity consumption for a number of real houses over several months' time. For each monitored house, we the data consists of:

- the whole home electricity signal (current monitors on both phases of power and a voltage monitor on one phase) recorded at a high frequency (15kHz)
- up to 24 individual circuits in the home, each labeled with its category of appliance or appliances, recorded at 0.5 Hz
- up to 20 plug-level monitors in the home, recorded at 1 Hz, with a focus on logging electronics devices where multiple devices are grouped to a single circuit.

The dataset contains power data from 6 houses in the Boston area collected in the summer of 2011. We chose to work with the low_freq directory which contains average power readings for both the two power mains and the individual circuits of the house. The data is logged at a frequency of about once a second for a mains and once every three seconds for the circuits. It seems like each appliance has already the same sampling rate as the mans, because it is used a lot by NILM. That leads to no need of resampling. The directory is organized as follows:

- redd/low_freq/
    - house_{1..n}/ -- directories for each house
    - labels.dat -- device category labels for every channel
    - channel_{1..k}.dat -- time/wattage readings for each channel

The main directory consists of several house_i directories, each of which contain all the power readings for a single house. Each house subdirectory consists of a labels.dat and several channels_i.dat files. The labels file contains channel numbers and a text label indicating the general category of device on this channel, for example:

- 1 mains_1
- 2 mains_2
- 3 refrigerator
- 4 lighting
- ...

In cases where the circuit has different device types on it (for example, circuits that power multiple outlets), we have attempted to best categorize the main type of appliance on the circuit. Each channel_i.dat file contains UTC timestamps (as integers) and power readings (recording the apparent power of the circuit) for the channel:

- ...
- 1306541834 102.964
- 1306541835 103.125

- 1306541836 104.001
- 1306541837 102.994
- 1306541838 102.361
- 1306541839 102.589
- ...

4. Dataset Preparation

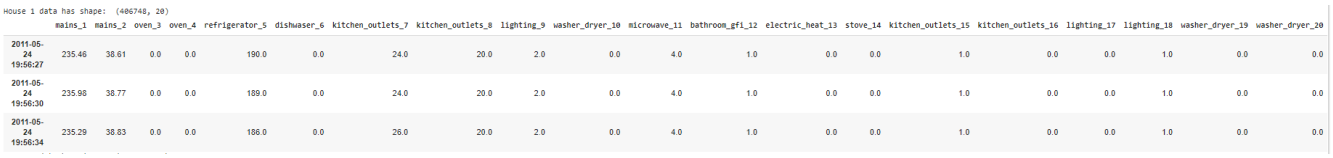
All our code is written in Python and we make use of Pandas and Numpy for data preparation. Also we use the GPU runtime in Google Collab to train our model.

For our convenience, we use the data from the first two houses of the REDD dataset and we present below the structure of each home.

House 1: {1: 'mains_1', 2: 'mains_2', 3: 'oven_3', 4: 'oven_4', 5: 'refrigerator_5', 6: 'dishwaser_6', 7: 'kitchen_outlets_7', 8: 'kitchen_outlets_8', 9: 'lighting_9', 10: 'washer_dryer_10', 11: 'microwave_11', 12: 'bathroom_gfi_12', 13: 'electric_heat_13', 14: 'stove_14', 15: 'kitchen_outlets_15', 16: 'kitchen_outlets_16', 17: 'lighting_17', 18: 'lighting_18', 19: 'washer_dryer_19', 20: 'washer_dryer_20'}

House 2: {1: 'mains_1', 2: 'mains_2', 3: 'kitchen_outlets_3', 4: 'lighting_4', 5: 'stove_5', 6: 'microwave_6', 7: 'washer_dryer_7', 8: 'kitchen_outlets_8', 9: 'refrigerator_9', 10: 'dishwaser_10', 11: 'disposal_11'}

We load the watt data in a pandas data frame in the format of labels as columns and timestamps as rows. You can see the final result in the figures below.



House 1 data has shape: (406748, 20)

	mains_1	mains_2	oven_3	oven_4	refrigerator_5	dishwaser_6	kitchen_outlets_7	kitchen_outlets_8	lighting_9	washer_dryer_10	microwave_11	bathroom_gfi_12	electric_heat_13	stove_14	kitchen_outlets_15	kitchen_outlets_16	lighting_17	lighting_18	washer_dryer_19	washer_dryer_20
2011-05-24 19:56:27	235.46	38.61	0.0	0.0	190.0	0.0	24.0	20.0	2.0	0.0	4.0	1.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0
2011-05-24 19:56:30	235.98	38.77	0.0	0.0	189.0	0.0	24.0	20.0	2.0	0.0	4.0	1.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0
2011-05-24 19:56:34	235.29	38.83	0.0	0.0	186.0	0.0	26.0	20.0	2.0	0.0	4.0	1.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0

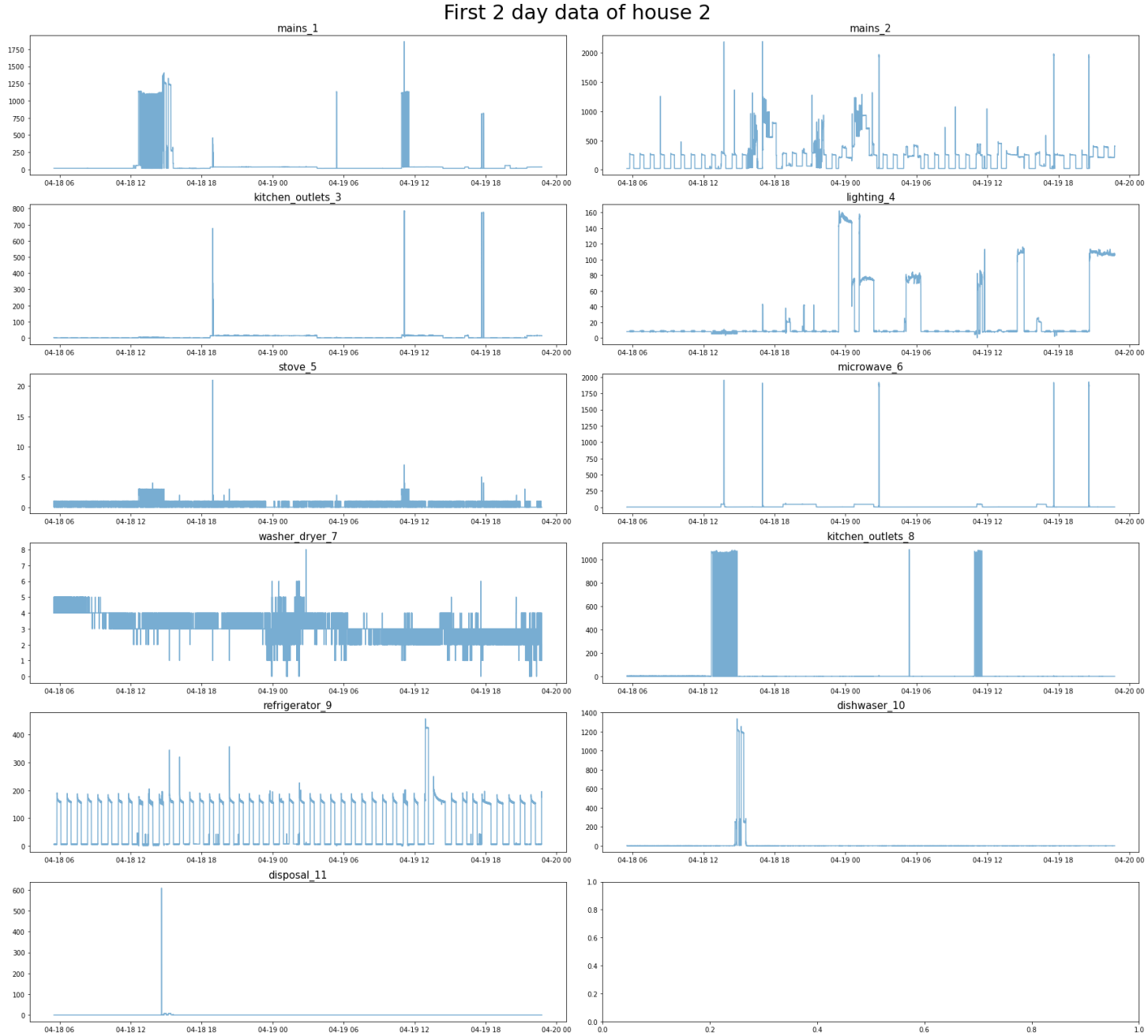
House 2 data has shape: (316840, 11)

	mains_1	mains_2	kitchen_outlets_3	lighting_4	stove_5	microwave_6	washer_dryer_7	kitchen_outlets_8	refrigerator_9	dishwaser_10	disposal_11
2011-05-22 23:59:01	10.84	252.61	0.0	9.0	0.0	5.0	0.0	2.0	158.0	0.0	0.0
2011-05-22 23:59:04	10.88	253.02	0.0	9.0	0.0	4.0	0.0	2.0	160.0	0.0	0.0
2011-05-22 23:59:08	10.84	252.77	0.0	9.0	0.0	4.0	0.0	2.0	157.0	0.0	0.0

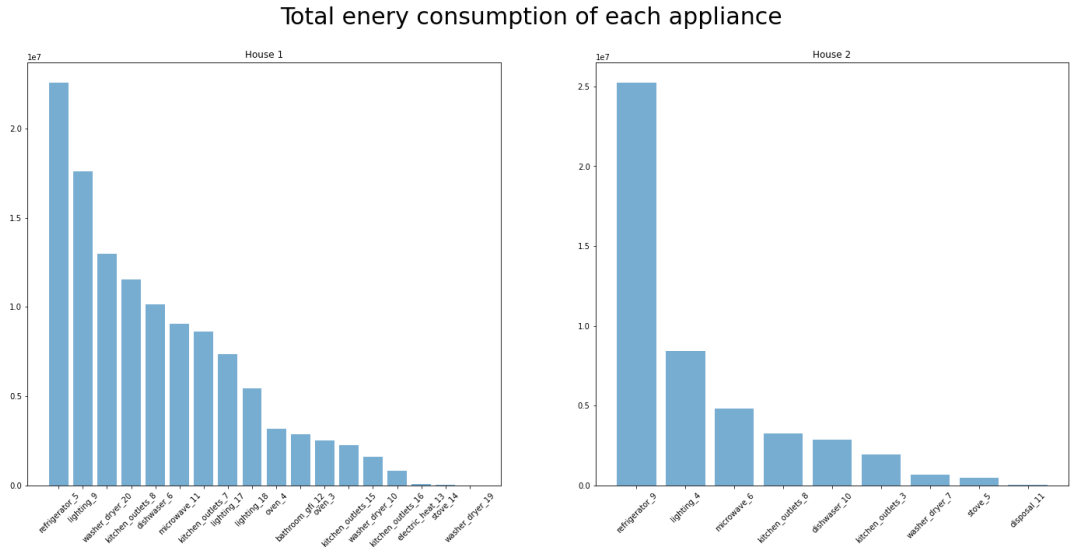
We observed that house 1 contains 23 dates from 18/04/2011 to 24/05/2011 and house 2 contains 16 dates starting from 18/04/2011 to 02/05/2011 and the date 22/05/2011.

To observe the appliances' and mains' waveforms, we plotted indicatively the first two days data of house 1 and house 2. As you can see in the diagrams that are depicted below, at the left column we see mains_1 and its appliances and at the right column we see mains_2 and its appliances. Some appliances like the oven have one-time spikes, in contrary to the refrigerator for example, that seems to be periodic. Note that, the mains waveforms are a combination average of all appliances.

First 2 day data of house 1



An additional observation comes from the diagrams below that represent the total energy consumption of each appliance. For both houses, the refrigerator seems to have the higher power consumption.



4.1. Training, validation and test data separation

We use data for training and test only from the first house. We assign to df_train the first 10 dates, df_val dates 11 to 16 and the remaining 17 to 23 to df_test. We use mains_1 and mains_2 to predict each appliance's consumption. For example, for the refrigerator, the structure is the following.

- Training data:
  - x_train: mains_1, mains_2
  - y_train: refrigerator_5
- Validation data:
  - x_val: mains_1, mains_2
  - y_val: refrigerator_5
- Test data:
  - x_test: mains_1, mains_2
  - y_test: refrigerator_5

5. Important knowledge

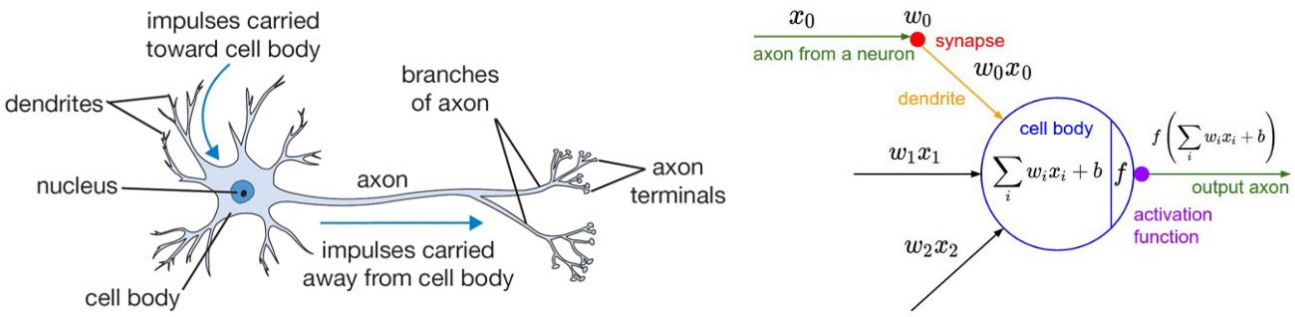
Before we continue with the model, we find it crucial to explain some terms that we have studied along the process.

5.1. Neural Networks

Deep learning and neural networks are hot subjects in computer science and the IT sector because they now provide the greatest answers to many issues in speech, picture, and natural language processing. Recently, several papers demonstrating AI that can learn to paint, make 3D models, develop user interfaces, some produce images given a text, and many more astonishing things are being done every day utilizing neural networks have been published.

A neural network is a collection of algorithms that aims to identify underlying links in a piece of data using a method that imitates how the human brain functions. In this context, neural networks are systems of neurons that can be either organic or synthetic in origin.

Since neural networks are capable of adapting to changing input, the network can produce the optimal outcome without having to change the output criterion. The artificial intelligence-based idea of neural networks is quickly gaining prominence in the design of trading systems.



biological neuron (left) and a common mathematical model (right)

The neuron, also known as a node or unit, is the fundamental computational component of a neural network. It computes an output after receiving input from various nodes or an external source. Each input

has a corresponding weight (w), which is determined by how significant it is in relation to other inputs. The weighted total of the inputs to the node is subjected to a function.

We can infer that a neural network is composed of neurons, that these neurons are biologically connected through synapses where information travels (these are the weights in our computational model), and that when we train a neural network, we want the neurons to fire whenever they recognize particular patterns in the data. To model the fire rate of the neurons, we use an activation function.

5.2. Convolutional Neural Networks

A filter modifies the input in a convolution. Convolutional networks use several filters to slice through the picture, map each one individually, and learn various elements of an input image. Imagine a tiny filter searching for, say, a dark edge as it moves across the image from top to bottom from left to right. Every time a match is discovered, it is displayed on an output picture.

A deep learning neural network called a convolutional neural network, or CNN, is made for processing organized arrays of input, like photographs. The state-of-the-art for many visual applications, such as image classification, convolutional neural networks are widely employed in computer vision. They have also found success in natural language processing for text categorization.

The patterns in the input picture, such as lines, gradients, circles, or even eyes and faces, are extremely well recognized by convolutional neural networks. Convolutional neural networks are extremely effective for computer vision because of this quality. Convolutional neural networks do not require any preparation and may function immediately on a raw picture, in contrast to older computer vision methods.

A feed-forward neural network with up to 20 or 30 layers is known as a convolutional neural network. The convolutional layer is a unique sort of layer that gives convolutional neural networks its strength.

5.3. LSTM

An artificial neural network called Long Short-Term Memory (LSTM) is employed in deep learning and artificial intelligence. LSTM features feedback connections as opposed to typical feedforward neural networks. Such a recurrent neural network may analyze complete data sequences in addition to single data points (such as photos) (such as speech or video). For instance, LSTM may be used for applications like speech recognition, robot control, machine translation, networked, unsegmented handwriting recognition, video games, and healthcare. The most frequently used neural network of the 20th century is LSTM.

A cell, an input gate, an output gate, and a forget gate make up a typical LSTM unit. The three gates control the flow of information into and out of the cell, and the cell remembers values across arbitrary time periods.

Given that there may be unexpected delays between significant occurrences in a time series, LSTM networks are ideally suited to categorizing, processing, and generating predictions based on time series data.

5.4. Dense Layer

In any neural network, a layer that is densely linked to its preceding layer indicates that every neuron in the layer is connected to every other neuron in the layer above it. In artificial neural network networks, this layer is the one that is most frequently utilized.

A model's dense layer neurons do matrix-vector multiplication and receive output from all of the neurons in the layer above them. When doing matrix vector multiplication, make sure that the row vector of the dense layer's output from the previous layers is identical to its column vector. According to the basic principle of matrix-vector multiplication, the row vector must have an equal number of columns as the column vector.

5.5. Activation Functions

Binary decisions about the neural network's output are made using the activation function or transfer function. The values that are produced are mapped between 0 and 1, -1 to 1, etc (depending upon the function).

The Activation Functions can be basically divided into 2 types:

- Linear Activation Function
- Non-linear Activation Functions

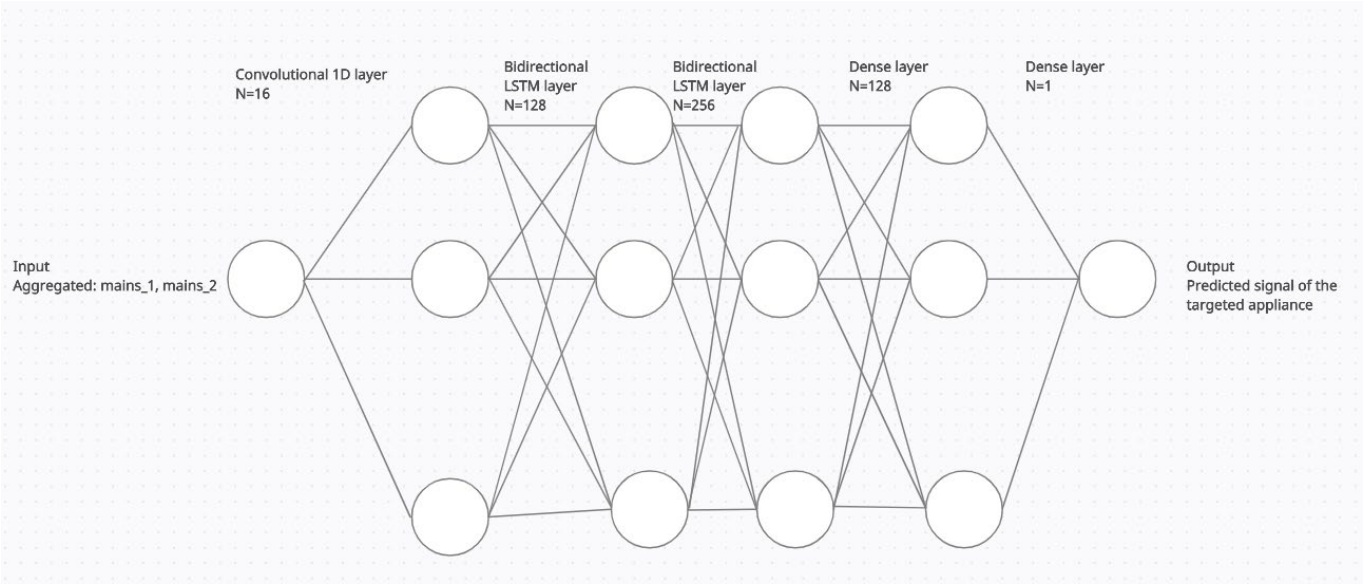
6. Algorithm

First of all, being in an early stage in our study, we tested as a target appliance the refrigerator, which gives a periodic wave. Our goal is to test more appliances in the future. According to the Neural NILM study, we used all the aforementioned Neural Network architectures. You can see the implemented model in the following figure.

```
#rnndisagreggator
def build_fc_model():
    fc_model = Sequential()
    fc_model.add(Conv1D(16, 4, activation="linear", input_shape=(2,1), padding="same", strides=1))
    fc_model.add(Bidirectional(LSTM(128, return_sequences=True, stateful=False), merge_mode='concat'))
    fc_model.add(Bidirectional(LSTM(256, return_sequences=False, stateful=False), merge_mode='concat'))

     # Fully Connected Layers
    fc_model.add(Dense(128, activation='tanh'))
    fc_model.add(Dense(1, activation='linear'))
    fc_model.add( Dropout(0.2) )

    fc_model.compile(loss='mse', optimizer='adam')

    fc_model.summary()
    return fc_model

fc_model_1 = build_fc_model()
```

At first, our design solution is based on the neural network concept. We used the sequential model for building our layers. A Sequential model is appropriate for a plain stack of layers where each layer has exactly one input tensor and one output tensor. After that, we use a convolutional 1D layer with 16 filters (outputs), filter size 4 to set a window in our sequence, linear activation function and padding same, which results in padding with zeros evenly to the left/right or up/down of the input such that output has the same height/width dimension as the input. To process our sequence (timeseries), we use a bidirectional LSTM layer with 128 neurons which leads to one more bidirectional LSTM with 256 neurons. We continue with fully connected layers, using a dense layer of 128 neurons with the TanH function, concluding to a dense layer with 1 neuron (output), activated with the linear function.

In this way, our output will be a new predicted signal of energy consumption for our targeted appliance each time. In the figure below, we can see the neural network architecture.



Neural Network architecture

To evaluate our model we use **model.compile** with loss the MSE and optimizer Adam. The MSE is an estimator that measures the average of the squares of the errors—that is, the average squared difference between the estimated values and the actual value. Adam optimization is a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments.

We extract different metrics in the compile method as well. Those metrics are accuracy, recall, precision and f1 score.

The proportion of accurately predicted data points among all the data points is known as accuracy. The number of true positives and true negatives divided by the total number of true positives, true negatives, false positives, and false negatives is how it is more precisely described.

Recall literally is how many of the true positives were recalled (found), i.e. how many of the correct hits were also found.

Precision (your formula is incorrect) is how many of the returned hits were true positive i.e. how many of the found were correct hits.

By definition, F1-score is the harmonic mean of precision and recall. It combines precision and recall into a single number using the following formula:

$$\text{F1-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

We also use the **model.fit** function of TensorFlow and give x_train_1 (aggregated signal of mains_1 and mains_2 of house 1, dates 1 through 10), y_train_1 (appliance signal of house 1, dates 1 through 10), batch size 512, epochs 200 and validation split 0,33.

```
from keras import backend as K

def recall_m(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    recall = true_positives / (possible_positives + K.epsilon())
    return recall

def precision_m(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    return precision

def f1_m(y_true, y_pred):
    precision = precision_m(y_true, y_pred)
    recall = recall_m(y_true, y_pred)
    return 2*((precision*recall)/(precision+recall+K.epsilon()))



adam = Adam(lr = 1e-5)
fc_model_1.compile(loss='mean_squared_error', optimizer=adam,metrics=['acc',f1_m,precision_m, recall_m])
start = time.time()
checkpointer = ModelCheckpoint(filepath="./fc_oven_h1_2.hdf5", verbose=0, save_best_only=True)

hist_fc_1 = fc_model_1.fit( X_train1, y_train1, batch_size=512, verbose=1, epochs=200,validation_split=0.33, callbacks=[checkpointer])
loss, accuracy, f1_score, precision, recall = fc_model_1.evaluate(X_test1, y_test1, verbose=0)

print('Finish trainning. Time: ', time.time() - start)
```

We created functions in order to measure the metrics mentioned above, according to the formulas.

7. Results

The first appliance that we examined is the oven. The model.fit function results in the following results.

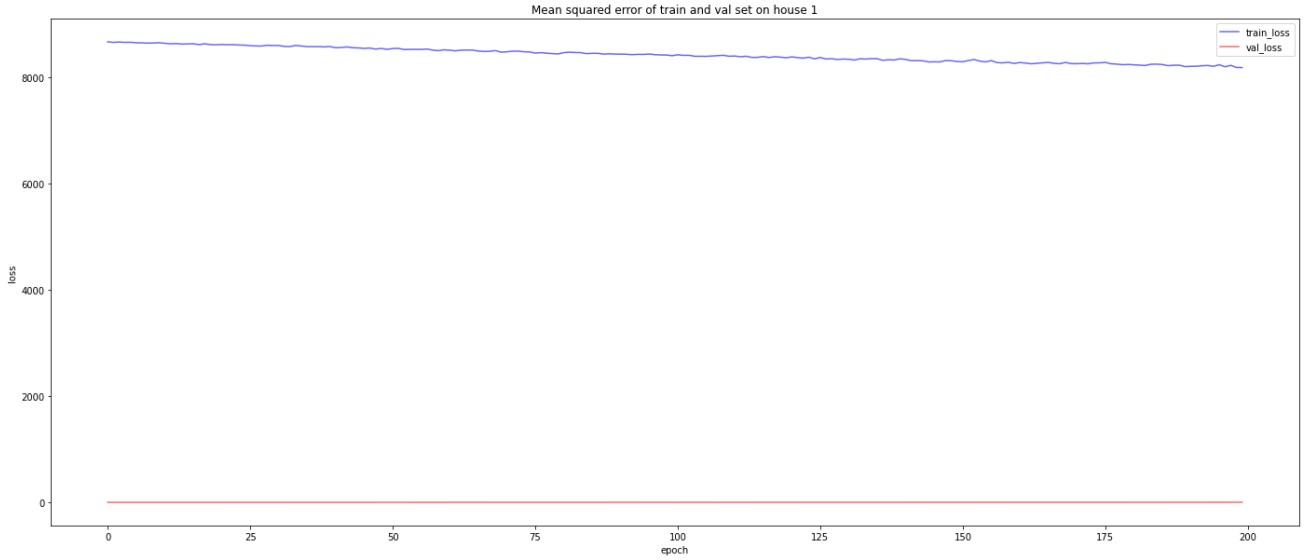
```
Epoch 200/200
282/282 [==============================] - 3s 12ms/step - loss: 8183.5708 - acc: 0.9581 - f1_m: 0.5550 - precision_m: 0.4379 - recall_m: 0.7939
Finish trainning. Time:  697.5928535461426
```

In the test set, we found an accuracy of 94%.

```
print(loss, accuracy, f1_score, precision, recall)
```

```
24270.966796875 0.9489644765853882 0.03786759823560715 11025.3603515625 0.0398859940469265
```

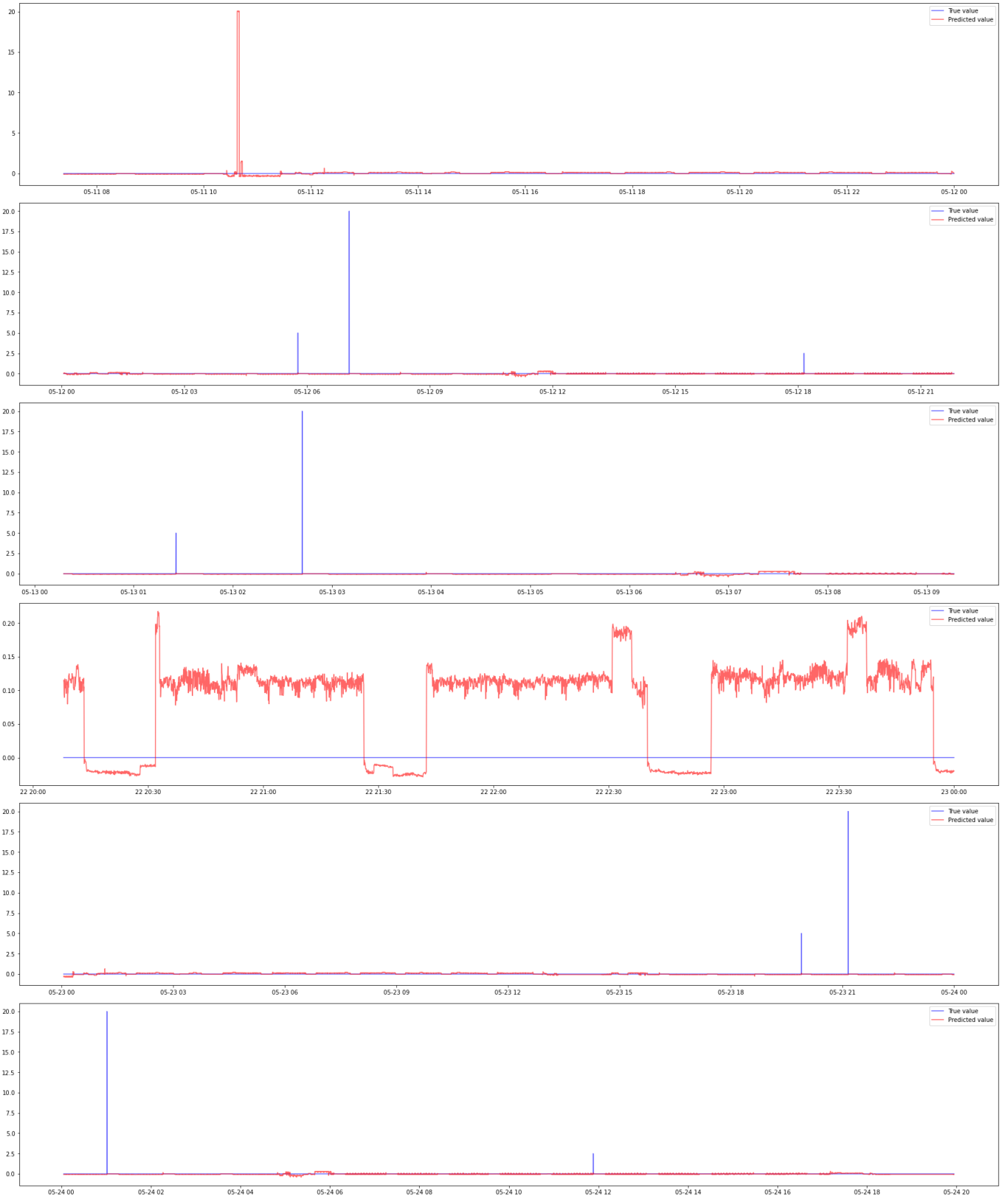
We plotted the mean square error of the training loss and validation loss for the oven and we conclude to the following result. As the validation loss is zero for every epoch we can conclude that for the validation dates (11 to 16), the oven was always in off state, meaning that it didn't operate at all for these days, so the selection of these dates for the oven is a little unfortunate. We could try some other dates in our future work.



We observe that the curve is the same but, as we expected, they differ a lot in the value, because the oven does not have a periodic waveform.

The plot for the real and predicted consumption of the oven for the 6-day test dataset can be seen below. As we expected, we predict that the oven is off but we don't meet the peaks.

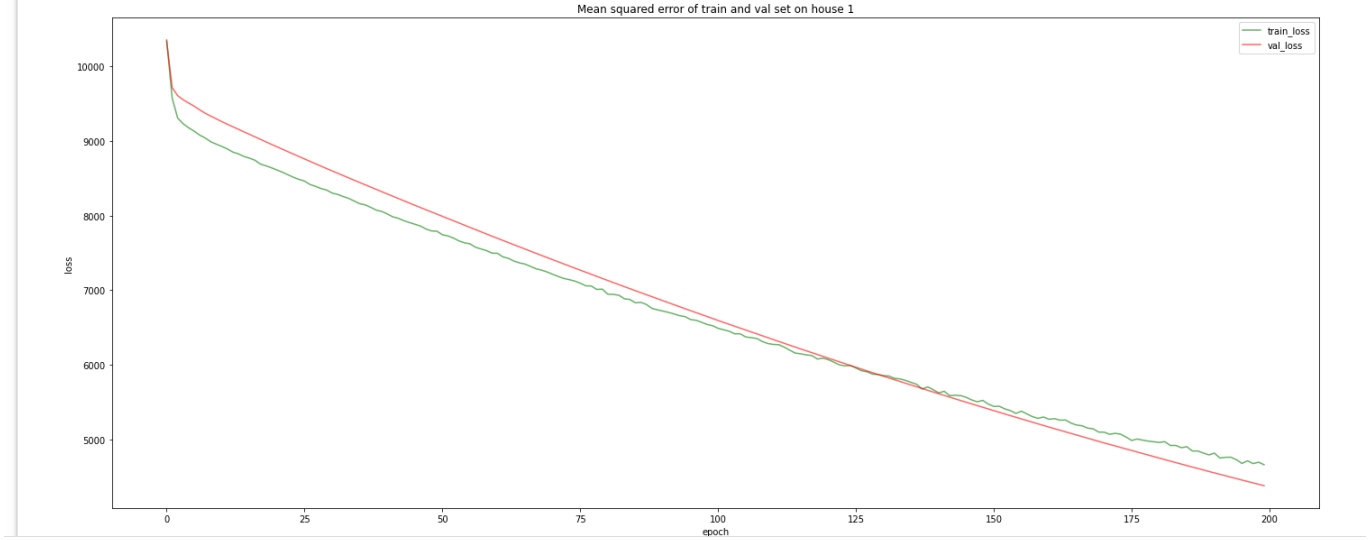
FC model: real and predict oven on 6 test day of house 1



We also tested the refrigerator, which has a periodic waveform.

To conclude, we can see that the validation loss and training loss are not that much different, and actually converge.

```
train_loss = hist_fc_2.history['loss']
val_loss = hist_fc_2.history['val_loss']
def plot_losses(train_loss, val_loss):
    plt.rcParams["figure.figsize"] = [24,10]
    plt.title('Mean squared error of train and val set on house 1')
    plt.plot( range(len(train_loss)), train_loss, color = 'g', alpha = 0.6, label='train_loss' )
    plt.plot( range(len( val_loss )), val_loss, color = 'r', alpha = 0.6, label='val_loss' )
    plt.xlabel( 'epoch' )
    plt.ylabel( 'loss' )
    plt.legend()

plot_losses(train_loss, val_loss)
```



Moreover, after training our model, we observed that the model can predict with a relative accuracy the refrigerator energy consumption. We used the code presented below to plot each appliance's energy consumption.

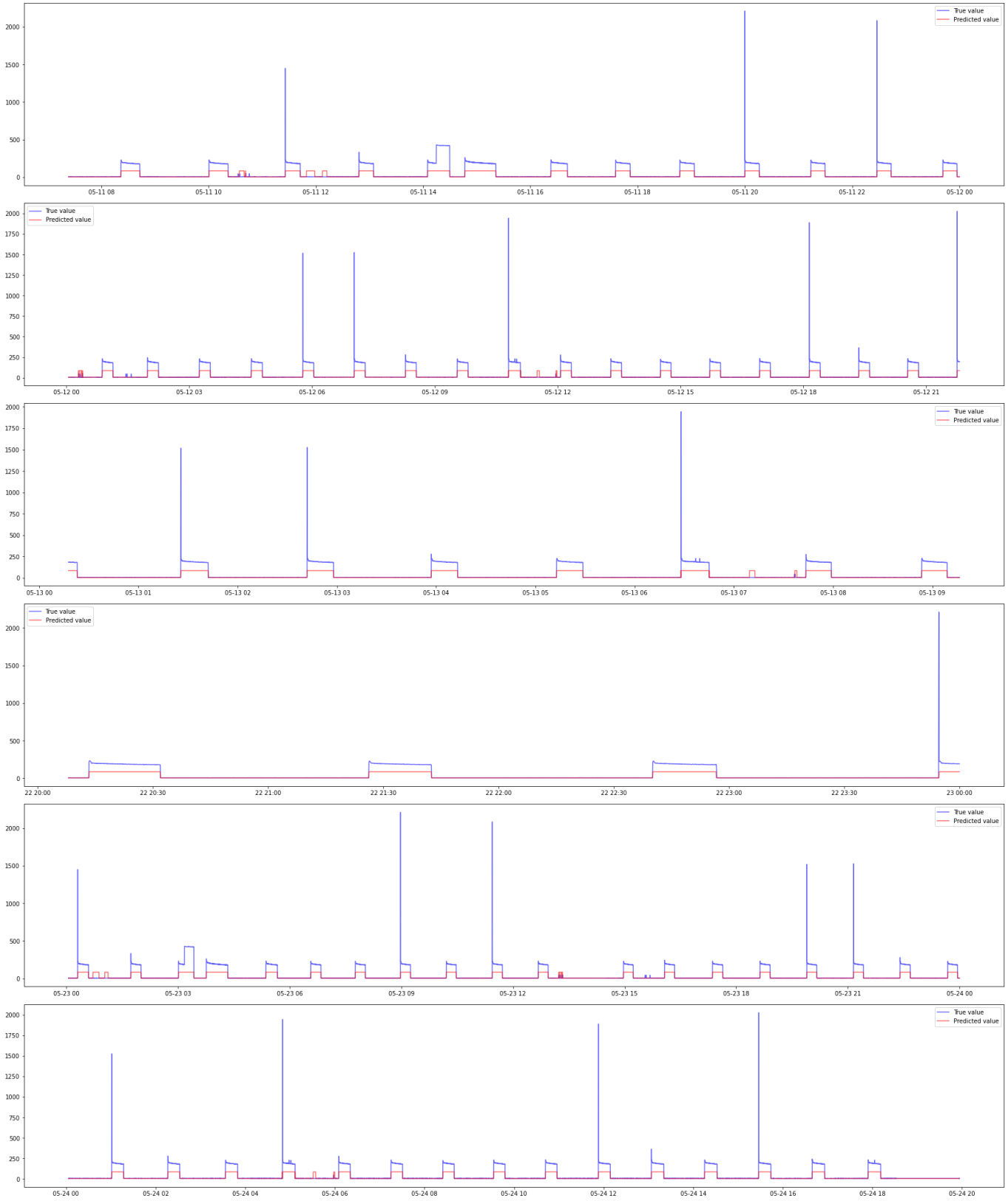
```
[ ]  # Plot real and predict refrigerator consumption on six days of test data
     def plot_each_app(df, dates, predict, y_test, title, look_back = 0):
         num_date = len(dates)
         fig, axes = plt.subplots(num_date,1,figsize=(24, num_date*5) )
         plt.suptitle(title, fontsize = '25')
         fig.tight_layout()
         fig.subplots_adjust(top=0.95)
         for i in range(num_date):
             if i == 0: l = 0
             ind = df[dates[i]].index[look_back:]
             axes.flat[i].plot(ind, y_test[l:l+len(ind)], color = 'blue', alpha = 0.6, label = 'True value')
             axes.flat[i].plot(ind, predict[l:l+len(ind)], color = 'red', alpha = 0.6, label = 'Predicted value')
             axes.flat[i].legend()
             l = len(ind)
```

In our case, we give as parameters the y_test which is the true value of the refrigerator's energy consumption for days 17 to 23, and our predicted signal and we get the result in the figure.

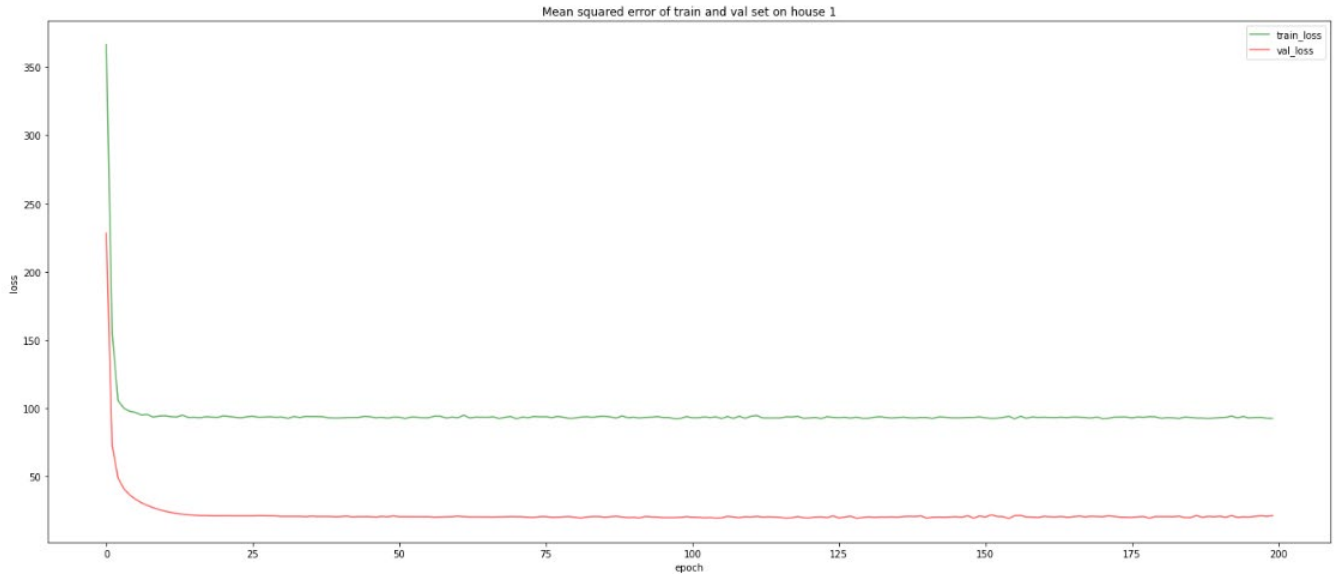
With red color we show the predicted signal and with blue the true signal is depicted.

We predict if the appliance is on and the exact value of consumption and we can observe that we predicted accurately in most cases.

FC model: real and predict refrigerator on 6 test day of house 1



Next, we examined the kitchen outlets.

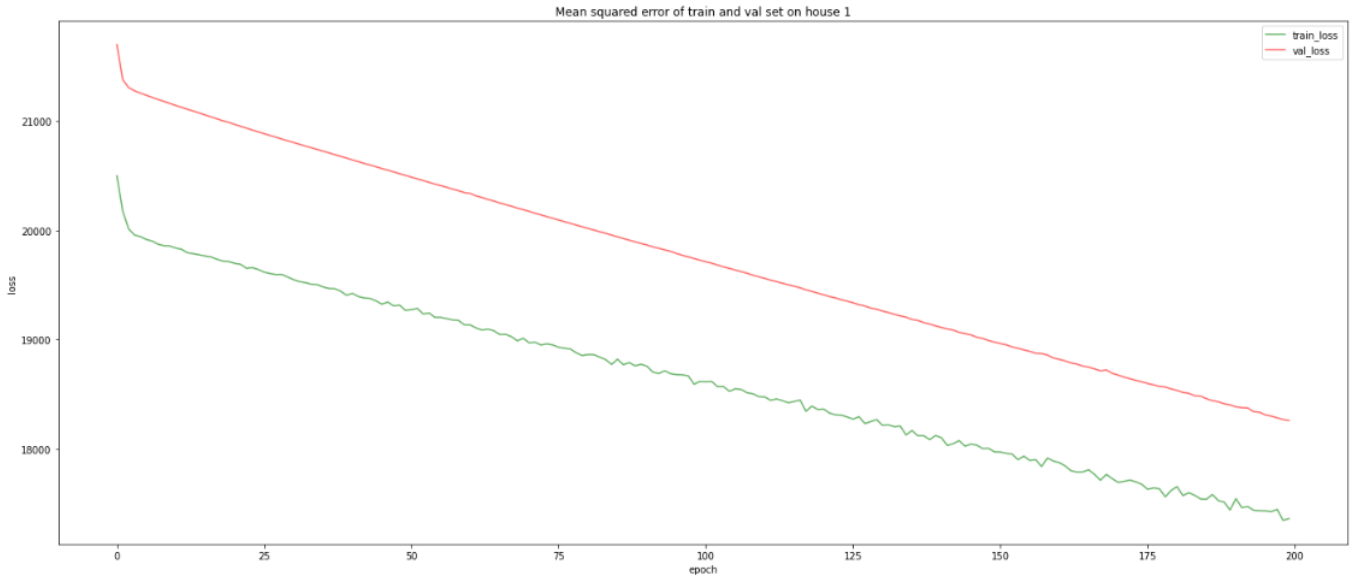


We observe the same kind of curve, but in a different set of values.

Below we present the real and predicted diagrams on the 6-day test set. What we can conclude is that we can predict pretty accurately that the outlets are in an on/off state, because the prediction is about waveforms and not actual consumption. In every square pulse we observe a peak.

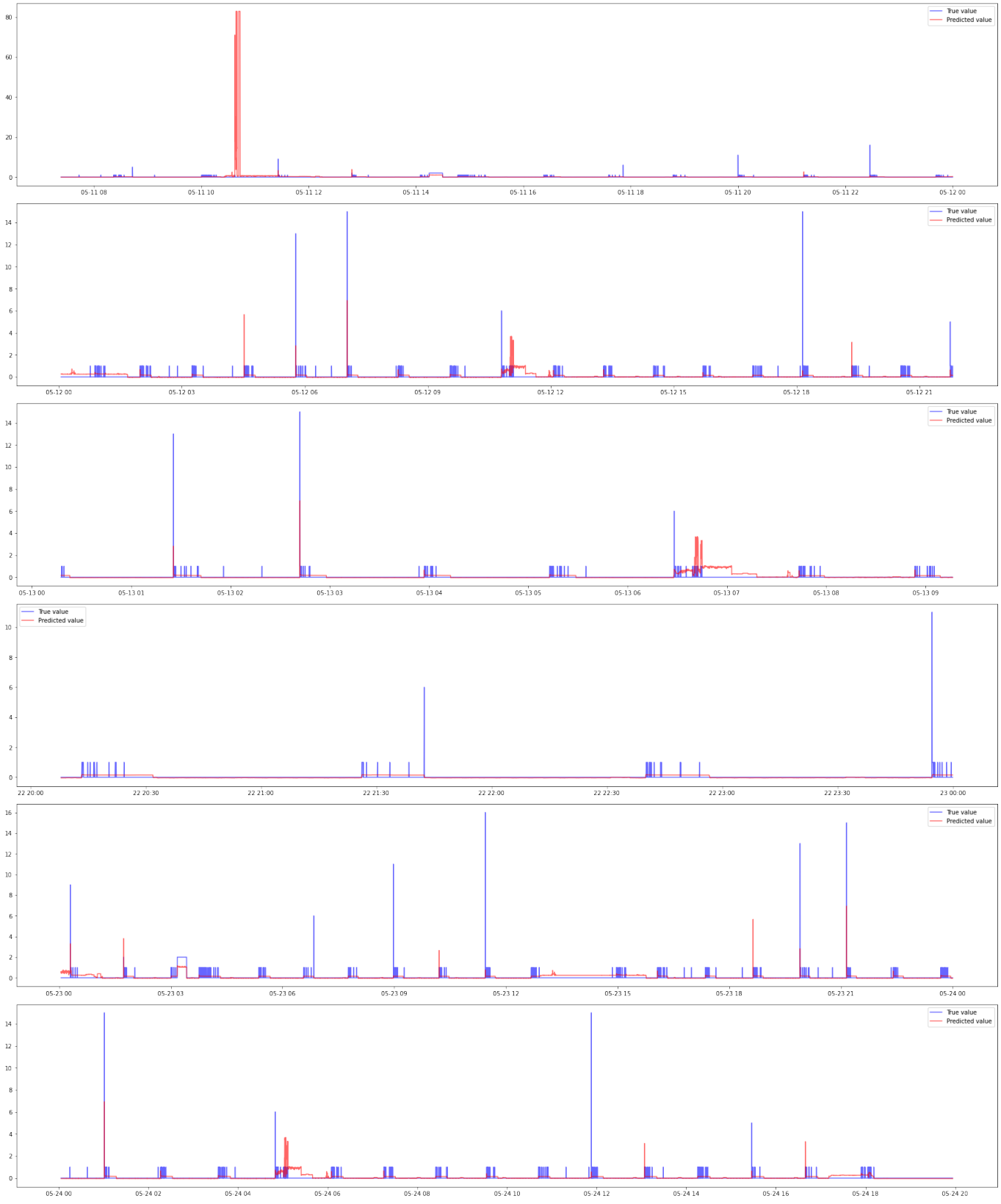
FC model: real and predict kitchen outlets on 6 test day of house 1

Last but not least, we examine the energy consumption of the dishwasher.



For the last time, the training loss and the validation loss are different, value-wise, but we can predict some peaks, but not all. Some days don’t have a very good prediction.

FC model: real and predict dishwasher on 6 test day of house 1



9. Conclusion and Future Work

In this assignment, we examined the energy disaggregation problem, and tried to evaluate a model that predicts the energy consumption of the oven and the fridge. The tested prediction had good results, mostly for appliances that appeared as a periodic waveform, e.g. the refrigerator. Other appliances that had an on/off usage, seemed to have been predicted less accurately.

Also, it should be mentioned that in a future work, we plan to validate the error rate of a model that would predict if the appliance is always on/always off. It is suggested that more experiments should be executed considering the number of neurons in CNN and Dense layers, and that we should study more houses and even, different datasets,

10. Acknowledgements

[1] Jack Kelly, & William Knottenbelt (2015). Neural NILM. In Proceedings of the 2nd ACM International Conference on Embedded Systems for Energy-Efficient Built Environments. ACM.

[2] Verma, A., Anwar, A., Mahmud, M., Ahmed, M., & Kouzani, A.. (2021). A Comprehensive Review on the NILM Algorithms for Energy Disaggregation.

[3] Siddiqui, A., & Sibal, A. (2020). Energy Disaggregation in Smart Home Appliances: A Deep Learning Approach. Energy.

[4] J. Zico Kolter, & Matthew J. Johnson (2011). REDD: A Public Data Set for Energy Disaggregation Research. In IN SUSTKDD.

[5] B. Zhao, L. Stankovic and V. Stankovic, "On a Training-Less Solution for Non-Intrusive Appliance Load Monitoring Using Graph Signal Processing," in IEEE Access, vol. 4, pp. 1784-1799, 2016, doi: 10.1109/ACCESS.2016.2557460.

[6] Makonin, Stephen; Popowich, Fred; Bajic, Ivan V.; Gill, Bob; Bartram, Lyn (2015). Exploiting HMM Sparsity to Perform Online Real-Time Nonintrusive Load Monitoring. IEEE Transactions on Smart Grid, (), 1–11.

[7] Veronica Piccialli; Antonio M. Sudoso; (2021). Improving Non-Intrusive Load Disaggregation through an Attention-Based Deep Neural Network . Energies, (), –.

[8] https://github.com/JackKelly/neuralnilm

[9] Energy Disaggregation and NILM – If You Can Measure It, You Can Improve It

[10] A Gentle Introduction To Neural Networks Series — Part 1

[11] Neural Network - Investopedia

[12] Convolutional Neural Network - DeepAI

[13] A Beginner's Guide to Convolutional Neural Networks (CNNs)

[14] Long short-term memory

[15] A Complete Understanding of Dense Layers in Neural Networks

[16] How to get accuracy, F1, precision and recall, for a keras model?