Smart home project

Barak Gahtan

All of the code is in my GitHub for future work.
https://github.com/BarakGahtan/smarthome_

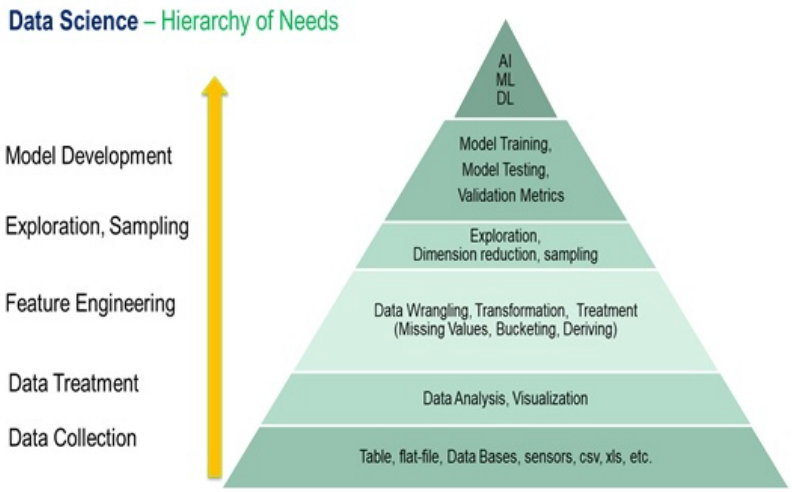
Introduction -

The main advantage of the following project was to get a hand on experience of the entire process of deep learning, from the raw files itself to generating a dataset. Transforming a raw dataset to a reliable working data loader, designing an architecture that can learn and predict from it.

The purpose of the project was above all to learn as much as possible and to get a real feeling of the preprocessing phase, which most of the time is "lost" during the courses of AI and DL. The initial intentions of the project were to design a smart home system that could predict many things, such as when to turn on the refrigerator or when to turn the alarm system of the house automatically.
Moreover, it was labeled as a research project that can lead to many directions, hence many different papers were read during the project in order to get ideas for future solutions of the problem.

A fundamental challenge in using deep learning is to obtain a good reliable set of data, that is well balanced and useful for analysis. The data itself and the ways of organizing and preprocessing the data, can have a very large effect on the outcome of the learning process.

As we can see from the figure, the model is as strong as its base is, therefore putting much effort into the preprocessing will yield better results in the future.
Most of the following project resided between the Data-Collection, and the Exploration sampling.



Preprocessing – The data and the problems with it

The given data was divided between a list of sensors:

- Alarm central
- Alarm Zone {i, i∈ 0,1,2,3}
- Alarm Aux channel {i, i∈ 0,1,..,9}
- Climate Control AAC aux channel {i, i∈ 0,1,2,3}
- Climate Control Thermostat {i, i∈ 0,1,2}
- Climate Control ThermoProbe {i, i∈ 0,1,2}
- Climate Control Valve Actuator {i, i∈ 0,..,9}
- Energy Management {i, i∈ 1,..,9}
- Water meter

At first, we needed to understand how each of the given data does is structure and what does it represent.

Alarm Central – example

A	B	C	D	E	F
Alarm.AlarmCentral	2016-01-22 22:26:37.18	null	null	null	null
Alarm.AlarmCentral	2016-01-22 22:26:47.10	1	null	null	null
Alarm.AlarmCentral	2016-01-22 22:26:47.18	1	0	null	null
Alarm.AlarmCentral	2016-01-22 22:26:47.26	1	0	1	null
Alarm.AlarmCentral	2016-01-22 22:26:47.27	1	0	1	1
Alarm.AlarmCentral	2016-01-22 22:50:36.40	null	null	null	null
Alarm.AlarmCentral	2016-01-22 22:50:47.17	1	null	null	null

As we can see the optional values are “null”, “0” or “1” – classifying that field as a binary or null, the central sensor is the manager of the following four zones that exist in the house.

Alarm Aux Channel – example

A	B	C
Alarm.AuxChannel.3	2016-01-22 22:26:37.27	null
Alarm.AuxChannel.3	2016-01-22 22:26:47.38	0
Alarm.AuxChannel.3	2016-01-22 22:50:36.43	null
Alarm.AuxChannel.3	2016-01-22 22:50:47.48	0
Alarm.AuxChannel.3	2016-01-22 22:57:14.77	null
Alarm.AuxChannel.3	2016-01-22 22:57:24.86	0
Alarm.AuxChannel.3	2016-01-23 08:39:15.61	null
Alarm.AuxChannel.3	2016-01-23 08:39:28.39	0
Alarm.AuxChannel.3	2016-01-23 08:51:31.45	null
Alarm.AuxChannel.3	2016-01-23 08:51:41.18	0
Alarm.AuxChannel.3	2016-01-24 12:47:07.86	1

As we can see the optional values are “null”, “0” or “1” – classifying that field as a binary or null, The Aux channel is a binary indicator if someone passed a certain sensor in the house. 0 – no change, 1 there was a change in contrast to the previous mark.

Climate Control AAC – example

ClimateControl.ACControl.1.9	2016-03-05 11:15:43.	Undefined	Undefined	null	Undefined
ClimateControl.ACControl.1.9	2016-03-05 11:18:28.	Undefined	Undefined	null	Undefined
ClimateControl.ACControl.1.9	2016-03-05 11:18:44.	Undefined	Undefined	null	Undefined
ClimateControl.ACControl.1.9	2016-03-05 13:06:48.	Auto	Cooling	22	Auto
ClimateControl.ACControl.1.9	2016-03-06 07:18:34.	Undefined	Off	null	Undefined
ClimateControl.ACControl.1.9	2016-03-06 08:10:14.	Auto	Cooling	22	Auto
ClimateControl.ACControl.1.9	2016-03-06 12:31:32.	Auto	Dehumidification	25	Auto

The different states of the AAC control in the house, for example “Cooling”, “Auto” or a specific temperature. Therefore that sensor’s values were mainly categorized.

ThermalProb Sensor – example –

ClimateControl.ThermalProbe.1	2016-01-22 22	20.1
ClimateControl.ThermalProbe.1	2016-01-22 23	20.2
ClimateControl.ThermalProbe.1	2016-01-22 23	20.1
ClimateControl.ThermalProbe.1	2016-01-22 23	20
ClimateControl.ThermalProbe.1	2016-01-22 23	19.9

The different measurements of the thermal probe in the house, for example 20.1 degrees. Therefore that sensor’s values were mainly floats.

Thermostat sensor – example

ClimateControl.Thermostat.1	2016-01-23 08:51:35.75	Heating	22
ClimateControl.Thermostat.1	2016-01-23 13:09:29.22	Heating	18
ClimateControl.Thermostat.1	2016-01-24 05:40:31.08	Heating	22
ClimateControl.Thermostat.1	2016-01-24 08:41:13.88	Heating	18
ClimateControl.Thermostat.1	2016-01-24 19:41:55.75	Heating	20

The different measurements of the thermal stat in the house, for example 22 degrees and “Heating”. Therefore, that sensor’s values were mainly floats and categorized.

Energy Management –

EnergyManagement.EnergyManagementActuator.1	2016-02-26 20:02:52.41	1125
EnergyManagement.EnergyManagementActuator.1	2016-02-26 20:03:00.41	721
EnergyManagement.EnergyManagementActuator.1	2016-02-26 20:03:01.43	477
EnergyManagement.EnergyManagementActuator.1	2016-02-26 20:03:02.43	176

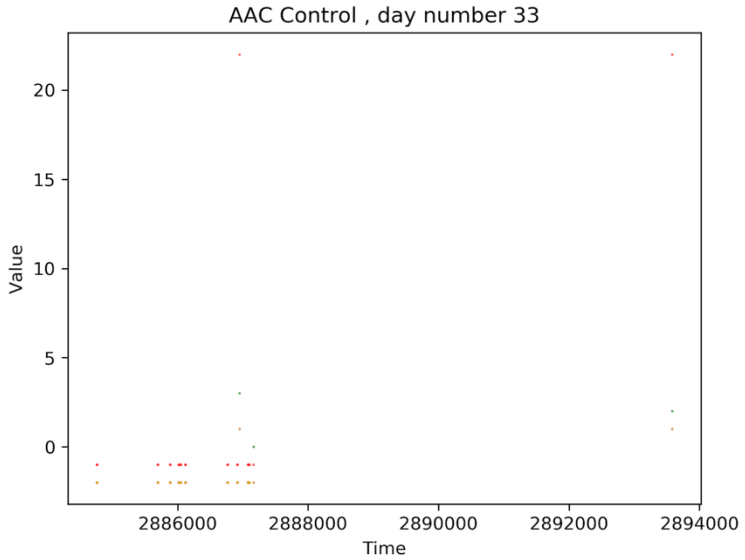
The different measurements of the energy managements stats in the house, for example 1125. Therefore, that sensor’s values were floats.

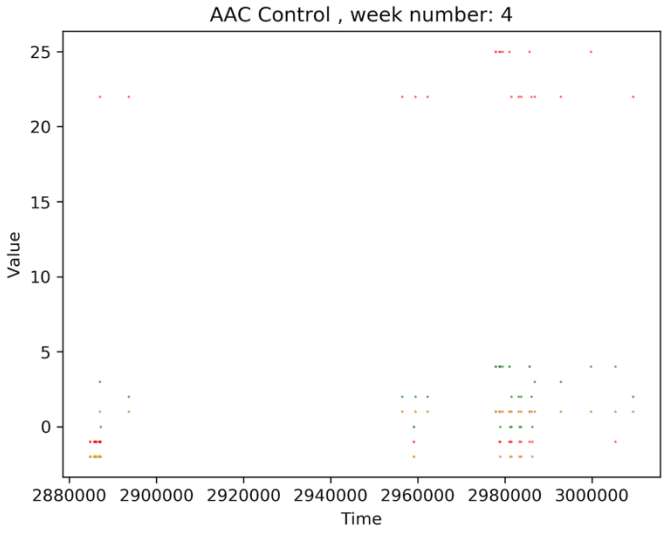
As we can see from the above examples, all of the sensors have many values, from very different domains. Some of the sensors have binary values, while others have categorical values or even different float numbers. That itself is a big problem in the world of preprocessing. Secondly, it should be noted that all of the sensors had time stamps that we wanted to use.

For start, we started to search online for papers that deal with data from different domains of numbers. For example, we read paper [1] regarding multi-dimensional scaling and thought maybe to scale all of the data to a certain range. Secondly another problem that it yielded was that different domains yield different noise [2]. After reading more papers regarding the subject, we came to a conclusion that scaling the data the same way, would not help us with the task we had in mind at the time, therefore we first wanted to visualize how does the data look. In addition, we decided to transfer the time stamps into UNIX time, so the representation would be the same for all of the sensors, in addition reducing the minimum time of all the sensors, so the UNIX time would start from zero.

In addition we thought of ways to combine the data, we thought maybe to join all the matrices of the data into one big matrix. We thought should we normalize each sensor.
The visualization phase of the data taught us a great deal on the data. We wanted to visualize the data per week or per day, to see if there are patterns or similarities between different sensors and time of the day \ week.
Following below are some example of graphs from different sensors -

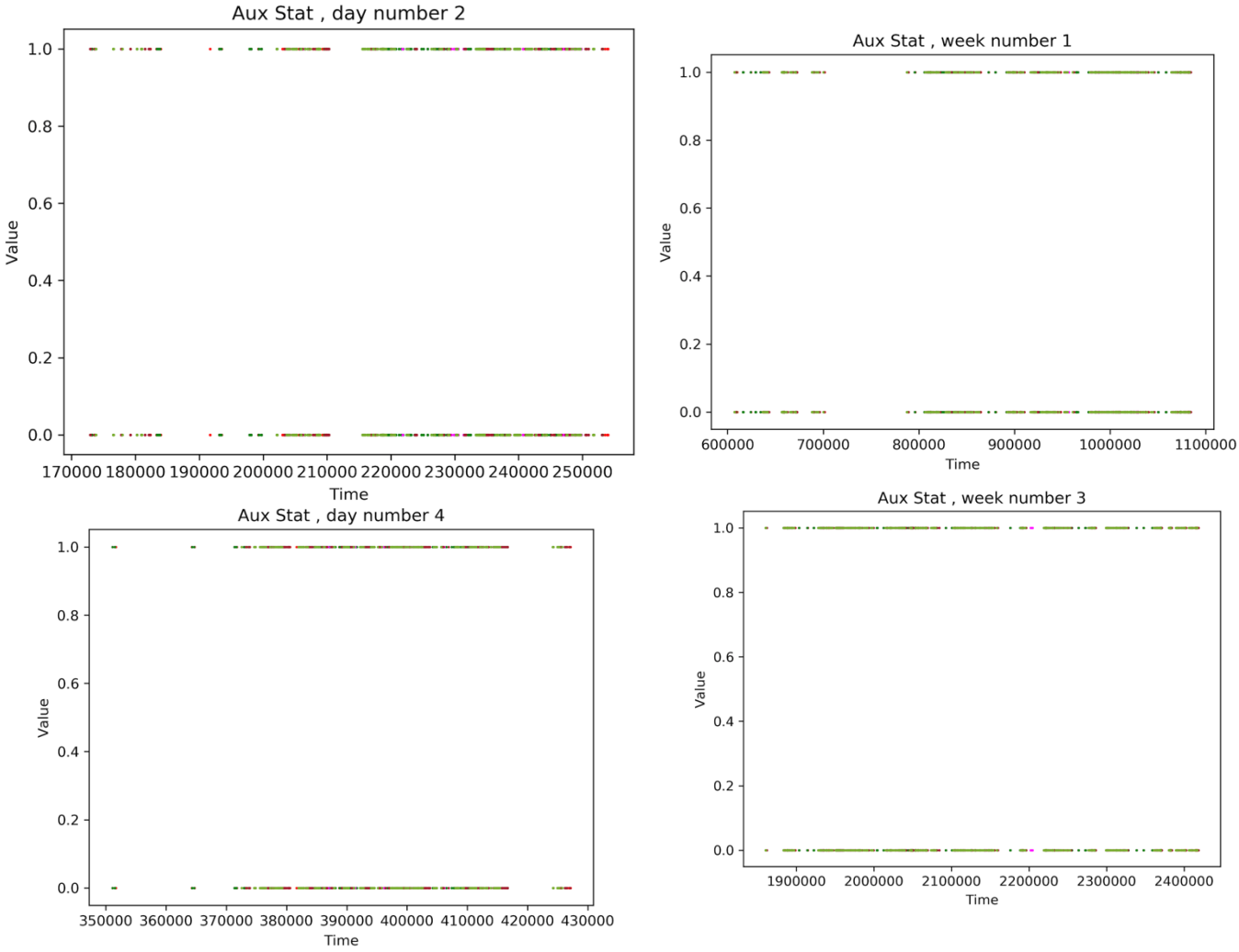
<u>An AAC graph</u>





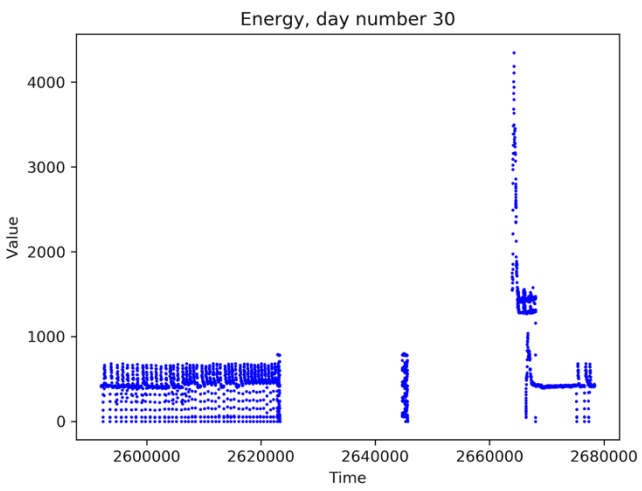
As we stated earlier, the AAC was categorized, hence the graph has dots representing different optional values.

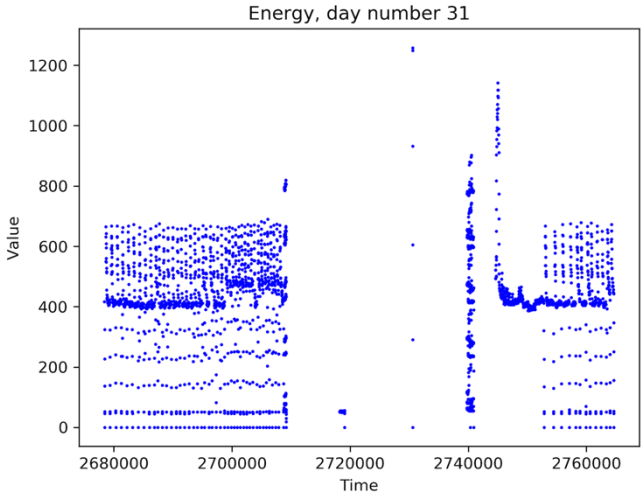
An Aux graph -

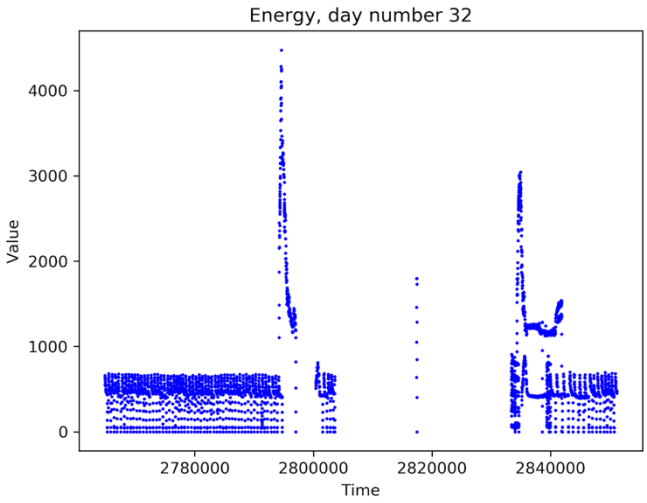


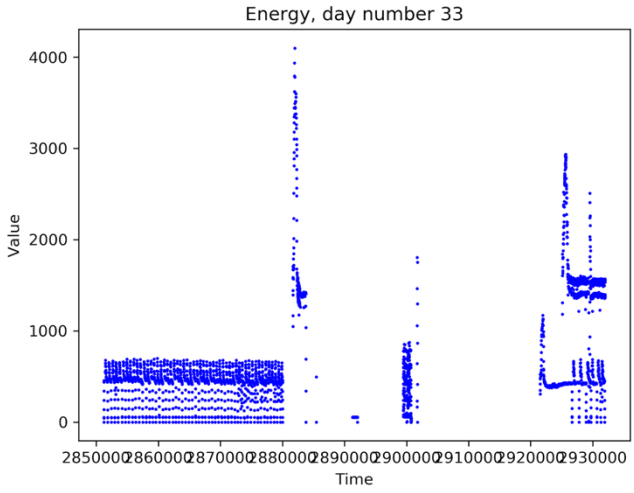
As we stated earlier, the AUX was binary, hence the graph has dots representing different optional values (0,1,null), but unlike the AAC graph, here we can see a little pattern between weeks and between days, the pattern itself is when there is a space in the graph. In addition to the fact that we see many samples, unlike the AAC.

An Energy graph for the consumption -



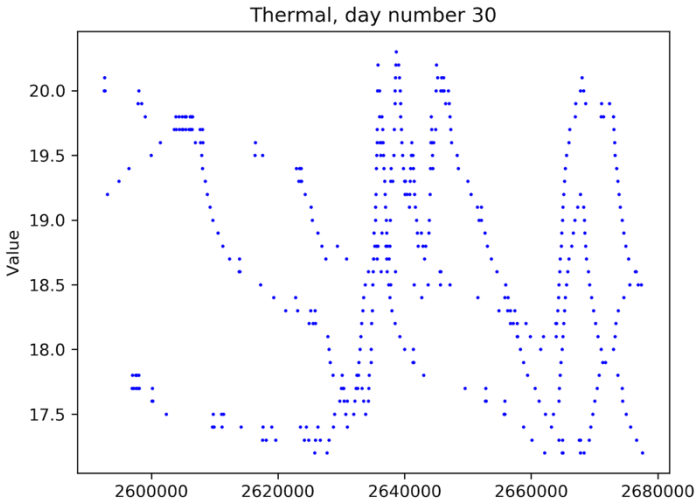


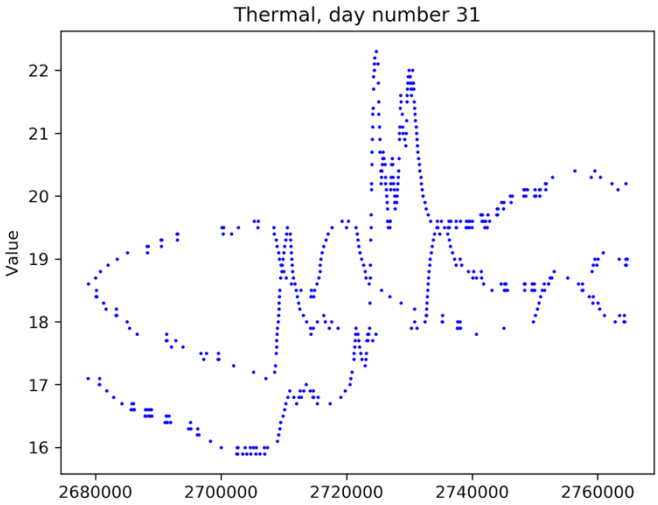


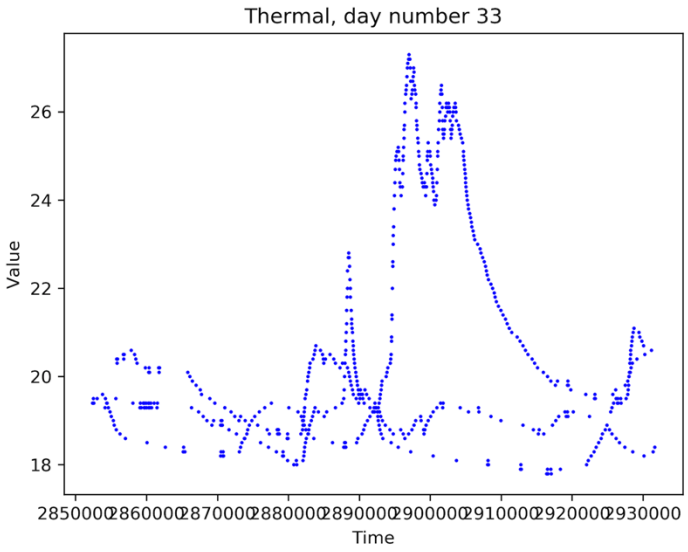


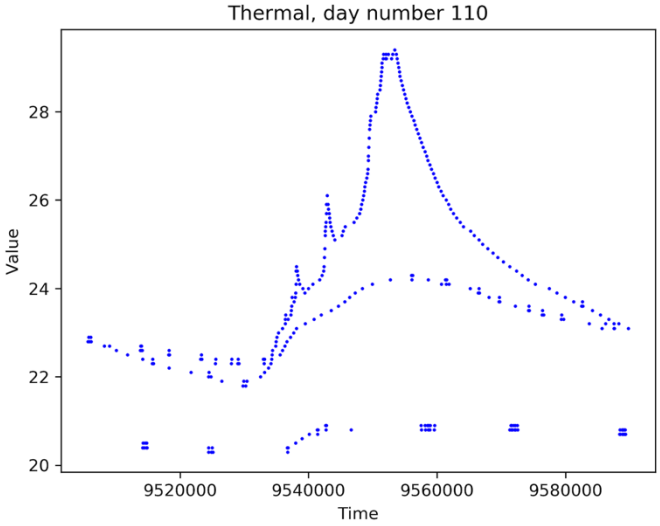
Those examples show the pattern as we can identify from the graphs regarding the energy consumption of the house during the days. We can see that the start of the day, the energy is quite idle (probably nighttime), then there are two peaks during the day, probably morning and evening, since those are the times most of the people are at the house.

A thermal graph for days -

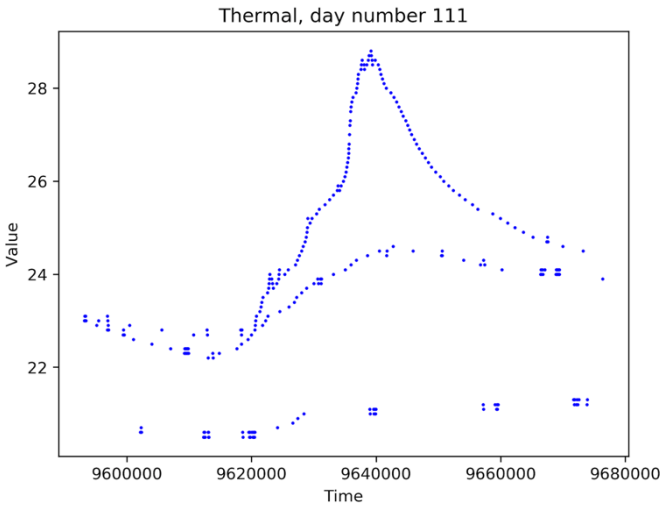




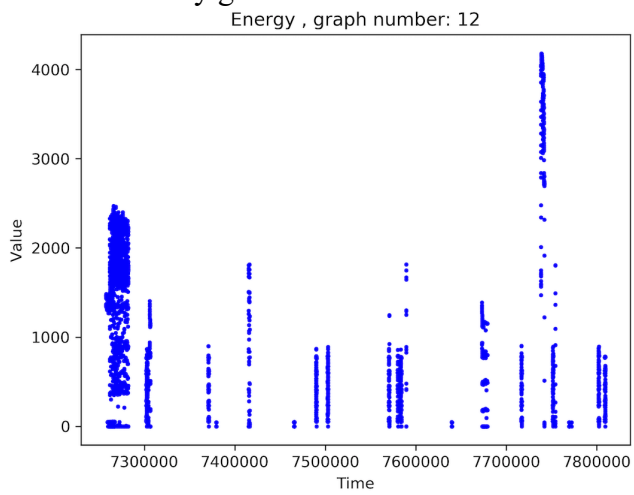


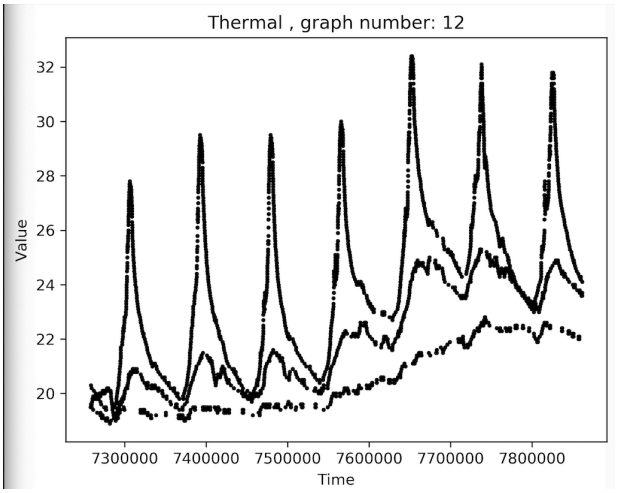


Those examples show the pattern as we can identify from the graphs regarding the thermal measurements of the house during the days. We can see the peak of the day is most of the time just after the middle of the day, probably around noon, which is very reasonable for thermal measurements.

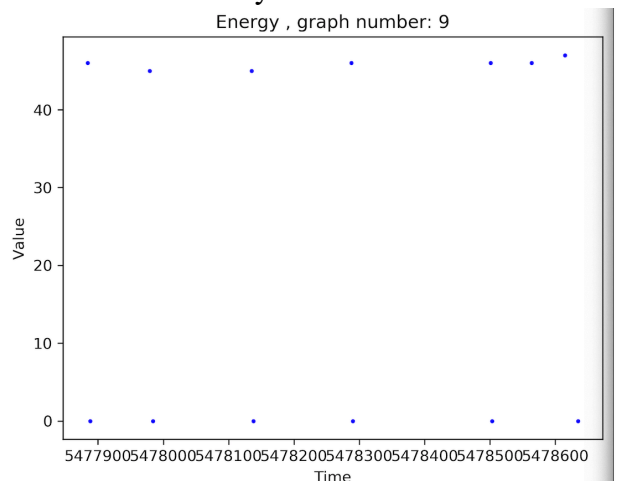


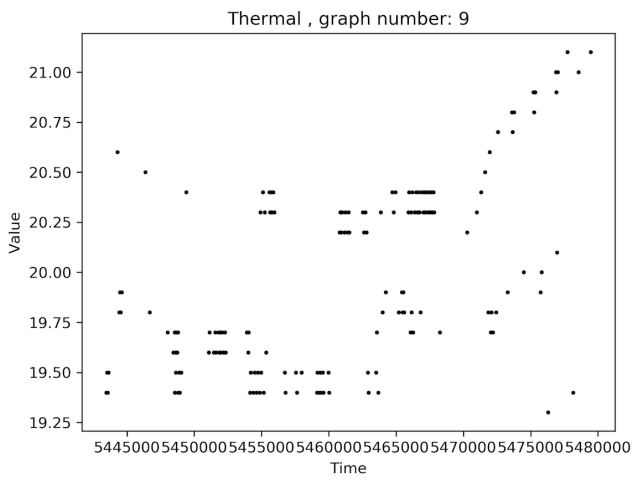
Those graphs brought us many insights to the data, we could firstly see for the first time how does the data look and get some idea if we have correlations between different sensors.
The following example illustrate the correlations between different sensors –
That is for example a week of the thermal measurements for week 12 – that we could see the similarities very good

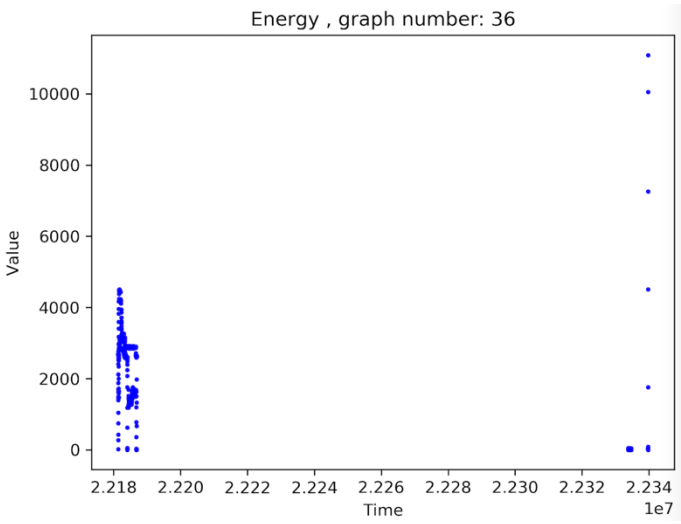


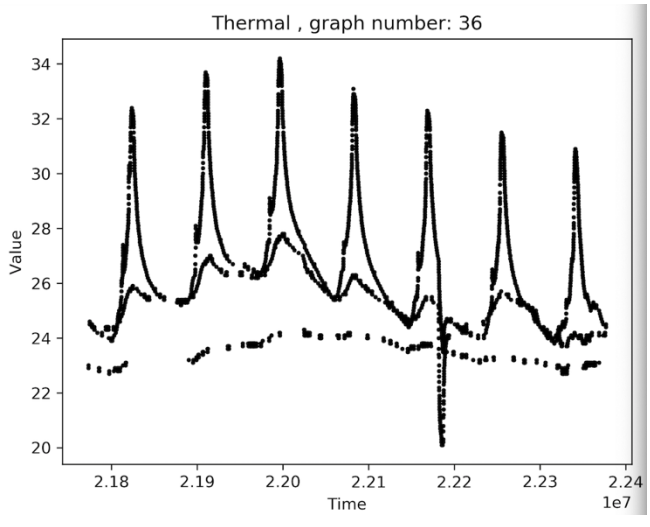


But in contrast many of the weeks are also









On one hand we can see that some of the weeks\ days have a clear similarities between the different sensors, but on the other hand we can see that some of the weeks\days are not consistent, not on the sampling rate (which is a big factor for data), and not on the similarities (mainly due to the sampling rate). As a result of that we came to a conclusion that we would need to find a way to learn our space with the following properties:

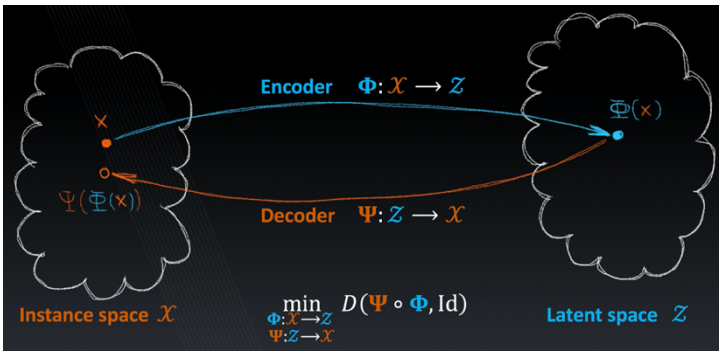
- Unbalanced data within sensors (per day the sampling rate it different within a sensor).
- Unbalanced data in compared to different sensors (some sensors have much more data than others).
- Different "domain" of field for the values of the sensors. (binary, floats, categories).

At that point of the project, we decided to use three sensors – energy, thermostat and AUX, because according to the graphs, they had the most recognizable patterns. We left the rest of the sensors for future research.

As a result of the properties we wanted to find a way to learn our space with the different sensors, we had to decide how to represent the data.

Possible Solutions

Firstly, we decided since our data has the properties mentioned above, and since that the data is not a well-organized data set, to use a VAE.
An autoencoder is a model which learns a representation of data in an unsupervised fashion (i.e without any labels). [4].



An autoencoder maps an instance x to a latent-space representation z. It has an encoder and a decoder part. The encoder is a neural net with different parameters, while the decoder is as well as neural net with parameters. Auto encoders can learn useful representations, but it is hard to use them as a generative model, because there is not distribution we can learn from in the latent space. Our main use of the VAE was to reconstruct samples within our own data with the properties mentioned above.

For us to successfully learn our latent space and generate samples, we needed to think of ways of how to insert the data into the encoder. We proposed five different solutions until we came to a final solution. Below is the evolution of the proposed solutions, followed by a brief explanation for each of them, and why it didn't work, or why was it decided that it is not good.

The five solutions –

1. Using graphs as images for learning.
2. Transferring the data into spectrograms while using a DFT.
3. Using an event-based camera alike solution.
4. Fusion of VAE using a unified grid.
5. Fusion of VAE using a unified normalized grid with different loss functions.

First solution –

Using generated raw graphs as image input for the VAE to learn.

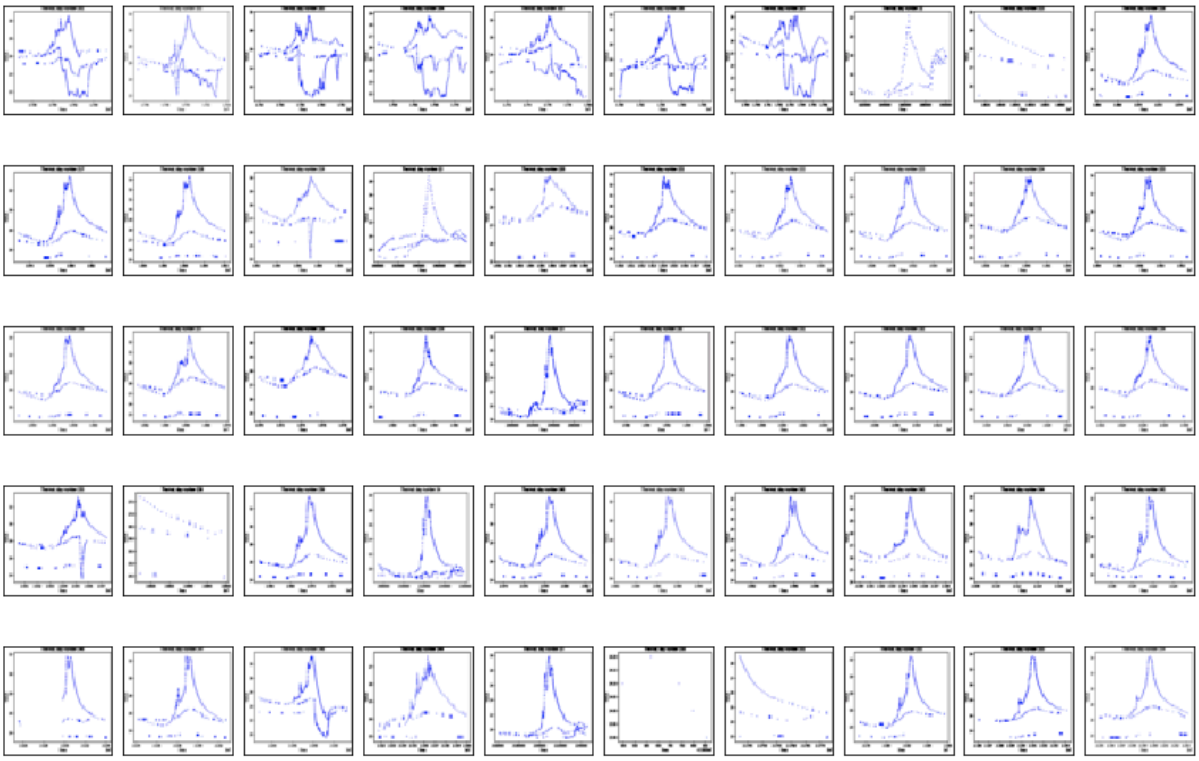
We decided at first to use 2D convolutions for the neural network implemented within the encoder and decoder for the first solution.
The neural network consisted of 1 in channels and 1024 out channels, we thought it is enough of inside layers.
For example, a picture of the encoder code -

```
class EncoderCNN(nn.Module):
    def __init__(self, in_channels=1, out_channels = 1024):
        super().__init__()
        modules = []

        modules.append(nn.Conv2d(in_channels=in_channels, out_channels=64, kernel_size=2, stride=1, padding=1))
        modules.append(torch.nn.BatchNorm1d(64))
        modules.append(torch.nn.LeakyReLU(0.2, inplace=True))
        modules.append(nn.Conv2d(64, 128, kernel_size=2, stride=1, padding=1))
        modules.append(torch.nn.BatchNorm1d(128))
        modules.append(torch.nn.LeakyReLU(0.2, inplace=True))
        modules.append(nn.Conv2d(128, 256, kernel_size=2, stride=1, padding=1))
        modules.append(torch.nn.BatchNorm1d(256))
        modules.append(torch.nn.LeakyReLU(0.2, inplace=True))
        modules.append(nn.Conv2d(256, 512, kernel_size=2, stride=1, padding=1))
        modules.append(torch.nn.BatchNorm1d(512))
        modules.append(torch.nn.LeakyReLU(0.2, inplace=True))
        modules.append(nn.Conv2d(512, out_channels, kernel_size=2, stride=1, padding=1))
        modules.append(torch.nn.BatchNorm1d(out_channels)) #1024 x 314( stride 1 minus 2 and approx 307-310)
        self.cnn = nn.Sequential(*modules)#.double() #Jbarak change (double)

    def forward(self, x):
        return self.cnn(x)#.double()
```

```
class DecoderCNN(nn.Module):
    def __init__(self, in_channels=1024, out_channels=1):
        super().__init__()
        modules = []
        modules.append(torch.nn.ReLU(inplace=True))
        modules.append(torch.nn.BatchNorm1d(in_channels))
        modules.append(nn.ConvTranspose2d(in_channels, 512, kernel_size=2, stride=1, padding=1))
        modules.append(torch.nn.ReLU(inplace=True))
        modules.append(torch.nn.BatchNorm1d(512))
        modules.append(nn.ConvTranspose2d(512, 256, kernel_size=2, stride=1, padding=1))
        modules.append(torch.nn.ReLU(inplace=True))
        modules.append(torch.nn.BatchNorm1d(256))
        modules.append(nn.ConvTranspose2d(256, 128, kernel_size=2, stride=1, padding=1))
        modules.append(torch.nn.ReLU(inplace=True))
        modules.append(torch.nn.BatchNorm1d(128))
        modules.append(nn.ConvTranspose2d(128, 64, kernel_size=2, stride=1, padding=1))
        modules.append(torch.nn.ReLU(inplace=True))
        modules.append(torch.nn.BatchNorm1d(64))
        modules.append(nn.ConvTranspose2d(64, out_channels, kernel_size=2, stride=1, padding=1, output_padding=0))
        self.cnn = nn.Sequential(*modules)#.double()

    def forward(self, h):
        # Tanh to scale to [-1, 1] (same dynamic range as original images).
        return torch.tanh(self.cnn(h))#.double()
```

The initial thoughts were to use the visualized data as images that we would like to learn to see patterns. As illustrated in prior chapters, there exist some patterns, therefore I thought to use those images of graphs as input to learn in the traditional learning ways of constructing images.

```
Found 311 images in dataset folder.
```



The fallback of this solution was the fact that many of the graphs were too much with same RGB color, and much less dots were on the graph; therefore, it was really an unlearnable task in some way.

Second solution – Transferring the data into spectrograms while first using a DTFT and from there applying an FFT\STFT.

After I came to a conclusion that we would need a way to transform the data into something more useable we started to think of ways to transform it. We decided that we would want to use the data as a 1D array consisting of a signal, the signal would be the connected samples from the data. On that signal the proposed solution was to do a FFT or STFT, so we could show the data in a 2D way as a spectrogram. In that way would insert the VAE a constructed image of a spectrogram to try to learn the latent space.
From discrete time to continuous time.

$$X^f(\theta) = \frac{1}{T_s}\sum_{k=-\infty}^{\infty} X^F\left(w = \frac{\theta - 2\pi k}{T_s}\right)$$

$$\theta \in [-\pi, \pi]\colon\ X^f(\theta) = \frac{1}{T_s}X^F\left(w = \frac{\theta}{T_s}\right)$$

The proposed solution rose many problems such as, how to pad the signal, how many zeroes should we add to it, what is the supposed length of it. Since the samples rate is not unified it was not a good solution to use, because it even implicated the unbalanced data much more, but it helped us to come to a conclusion that we needed to have an unified grid.

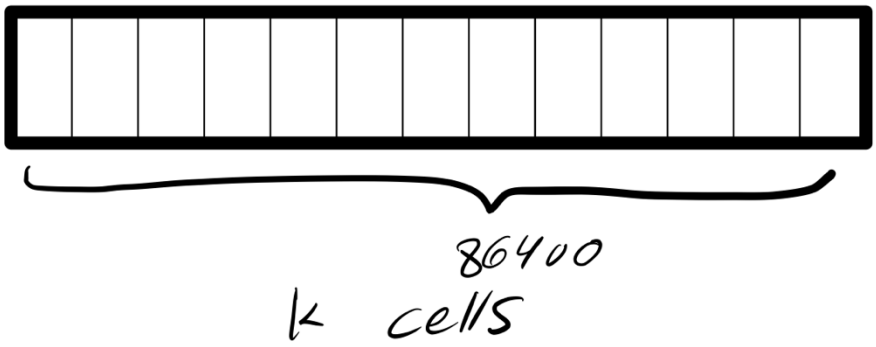
Third solution – Using an event-based camera alike solution.
We came to know that there exit some cameras that are called event cameras. Event cameras, such as the Dynamic Vision Sensor (DVS), are bio-inspired vision sensors that output pixel-level brightness changes instead of standard intensity frames. They offer significant advantages over standard cameras, namely a very high dynamic range, no motion blur, and a latency in the order of microseconds. However, because the output is composed of a sequence of asynchronous events rather than actual intensity images, traditional vision algorithms cannot be applied, so that new algorithms that exploit the high temporal resolution and the asynchronous nature of the sensor are required.
Pattern recognition algorithms, such as learning-based methods, have made significant progress with event cameras by converting events into synchronous dense, image-like representations and applying traditional machine learning methods developed for standard cameras.

Our thinking in that direction was to use a pixel as a sensor. That direction did not come to fruits, but it helped in a very big way, after that reading of ways ([5], [6]) , we came to a conclusion that we have to have an unified grid for all of the sensors in order to preprocess the data to be able to help us.

Fourth Solution –

After we came to a conclusion that all of the data should reside in an unifed grid, we thought how to transform the data. Let us remember that for each sensor, we had a time and value, or categorical values, or floats or binary.
First, we calculated the number of days we have in out limited data set.
Secondly, we calculated the average number of samples we have per day and per week.
When we calculated it per week, we saw that we had too little samples to work with, therefore the weekly data got canceled and we decided to work with per day data.

Once we decided to work with per day data set, we divided the total number of seconds there is in a day 86400 by the average number of samples per sensor. We used the $average\{sensor_{i-avg} \mid i\ is\ a\ different\ sensor\}$, define it as K.
Once we had K, we had a unified grid to work with as illustrated:

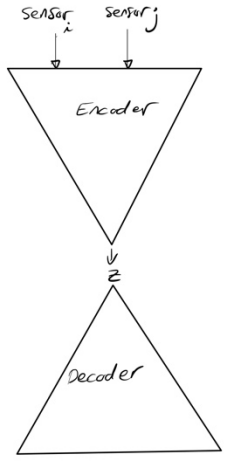


Each cell represented a unit time of the grid.
The first cell is from time 0 in a day until time 0+unit and so on.
After developing the grid, in order for us not to lose samples, we made a choice of using a ZOH solution, as each unit of time will consists of the sum of all of the previous samples that occurred until that unit of that sensor. That way, when in the future we would want to learn, we would actually learn the gradient of "integral" of the data, and that would indicate the change.

The fourth solution is the base of the solution we ultimately used.

The fifth solution – Normalization of the data and fusing the sensors in a VAE
Finally, we had a unified way of representing the data from different sensors and different domains. At that point we decided we are going to implement the VAE for 1D arrays,
Once we had our unified way of representing all of the data, we could train the VAE model.

```python
class EncoderCNN(nn.Module):
    def __init__(self, in_channels=1, out_channels = 1024):
        super().__init__()
        modules = []
        modules.append(nn.Conv1d(in_channels=in_channels, out_channels=64, kernel_size=2, stride=1, padding=1))
        modules.append(torch.nn.BatchNorm1d(64))
        modules.append(torch.nn.LeakyReLU(0.2, inplace=True))
        modules.append(nn.Conv1d(64, 128, kernel_size=2, stride=1, padding=1))
        modules.append(torch.nn.BatchNorm1d(128))
        modules.append(torch.nn.LeakyReLU(0.2, inplace=True))
        modules.append(nn.Conv1d(128, 256, kernel_size=2, stride=1, padding=1))
        modules.append(torch.nn.BatchNorm1d(256))
        modules.append(torch.nn.LeakyReLU(0.2, inplace=True))
        modules.append(nn.Conv1d(256, 512, kernel_size=2, stride=1, padding=1))
        modules.append(torch.nn.BatchNorm1d(512))
        modules.append(torch.nn.LeakyReLU(0.2, inplace=True))
        modules.append(nn.Conv1d(512, out_channels, kernel_size=2, stride=1, padding=1))
        modules.append(torch.nn.BatchNorm1d(out_channels)) #1024 x 314( stride 1 minus 2 and approx 307-310)
        self.cnn = nn.Sequential(*modules)#.double() #barak change (double)

    def forward(self, x):
        return self.cnn(x)#.double()


class DecoderCNN(nn.Module):
    def __init__(self, in_channels=1024, out_channels=1):
        super().__init__()
        modules = []
        modules.append(torch.nn.ReLU(inplace=True))
        modules.append(torch.nn.BatchNorm1d(in_channels))
        modules.append(nn.ConvTranspose1d(in_channels, 512, kernel_size=2, stride=1, padding=1))
        modules.append(torch.nn.ReLU(inplace=True))
        modules.append(torch.nn.BatchNorm1d(512))
        modules.append(nn.ConvTranspose1d(512, 256, kernel_size=2, stride=1, padding=1))
        modules.append(torch.nn.ReLU(inplace=True))
        modules.append(torch.nn.BatchNorm1d(256))
        modules.append(nn.ConvTranspose1d(256, 128, kernel_size=2, stride=1, padding=1))
        modules.append(torch.nn.ReLU(inplace=True))
        modules.append(torch.nn.BatchNorm1d(128))
        modules.append(nn.ConvTranspose1d(128, 64, kernel_size=2, stride=1, padding=1))
        modules.append(torch.nn.ReLU(inplace=True))
        modules.append(torch.nn.BatchNorm1d(64))
        modules.append(nn.ConvTranspose1d(64, out_channels, kernel_size=2, stride=1, padding=1, output_padding=0))
        self.cnn = nn.Sequential(*modules)#.double()
```



Solving using the fifth solution

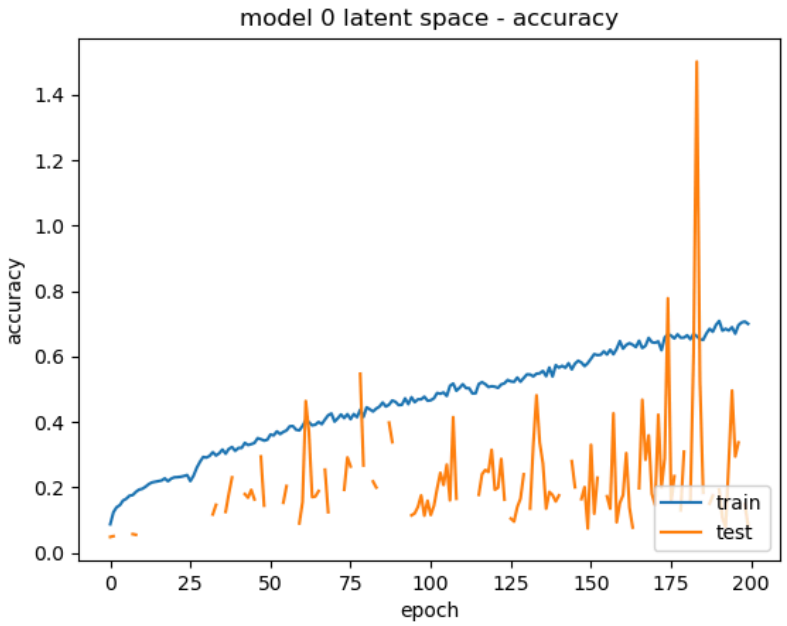
Now we could learn our own latent space in order to construct the 1D arrays after the encoder\decoder.
We trained our VAE with the same model, but with 7 different inputs.
Model 0 – only AUX data – the binary data.
Model 1 – only Thermal data.
Model 2 – only Energy data.
Model 3 – Thermal and Aux.
Model 4 – Energy and Aux.
Model 5 – Energy and Thermal
Model 6 – Energy, Thermal and Aux.

The reason for the different data as inputs was due to the fact we wanted to know if the addition of different sensors, will have an improvement on the studying of the latent space.
After training the models indeed the accuracy of the construction was much better as we added more sensor.

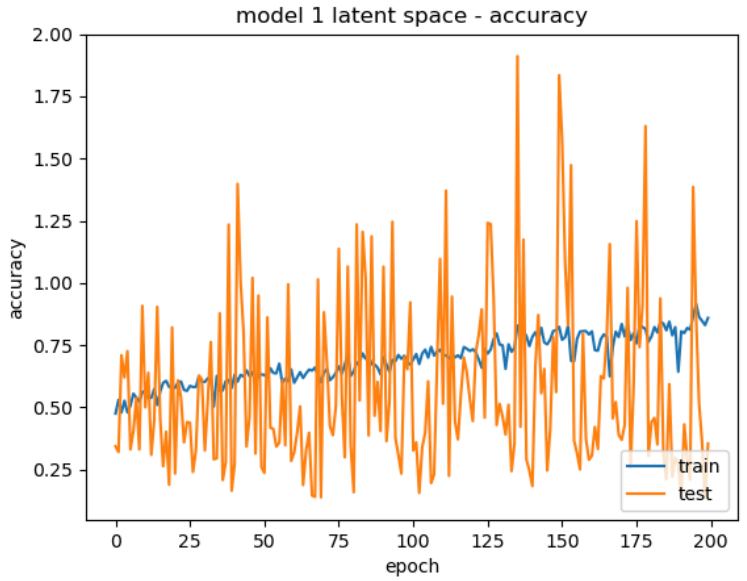
We normalized the entire data per sensor, so all of the values would within the range of [0,1].
We trained with the MSE loss functions,
The parameters used for training

```
hypers['batch_size'] = 4
hypers['h_dim'] = 128
hypers['z_dim'] = 64
hypers['x_sigma2'] = 0.001
hypers['learn_rate'] = 0.000001
hypers['betas'] = (0.85, 0.999)
```

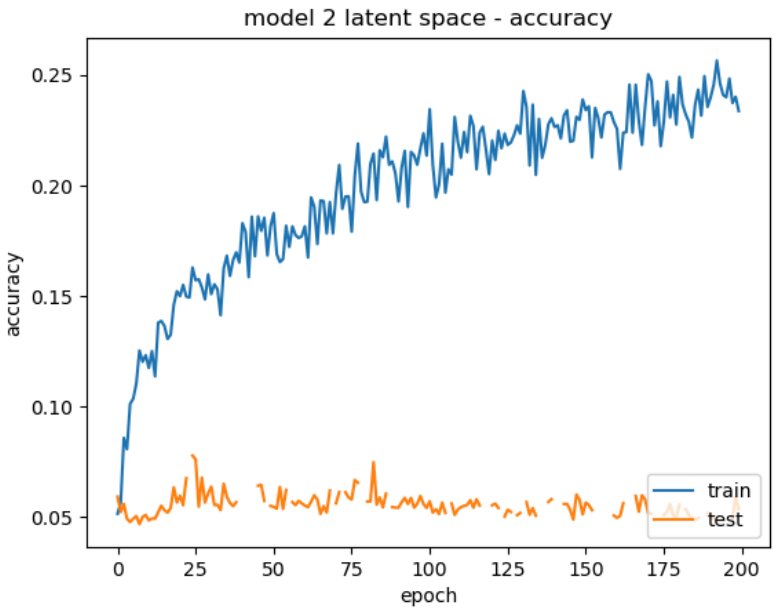
The learning of the latent space of the sensor, average loss and accuracy for the reconstruction of the data:
Model 0 – only AUX data – the binary data.



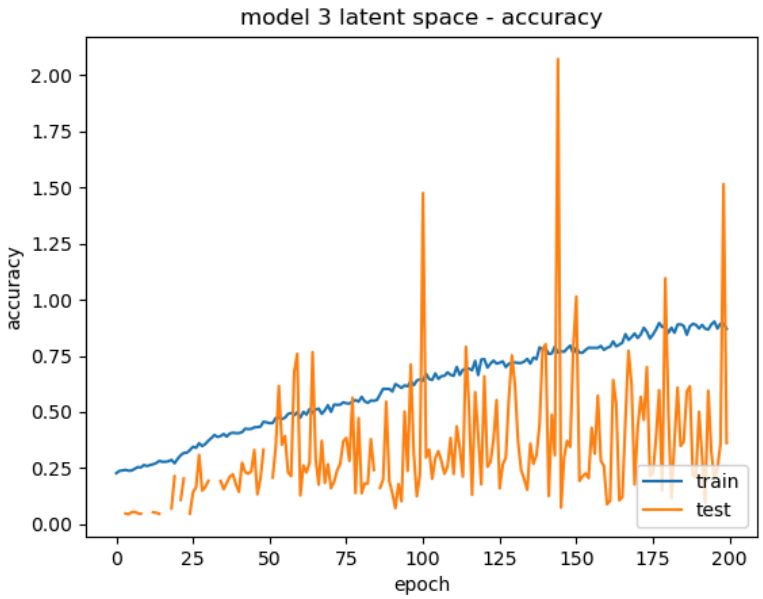
Model 1 – only Thermal data.



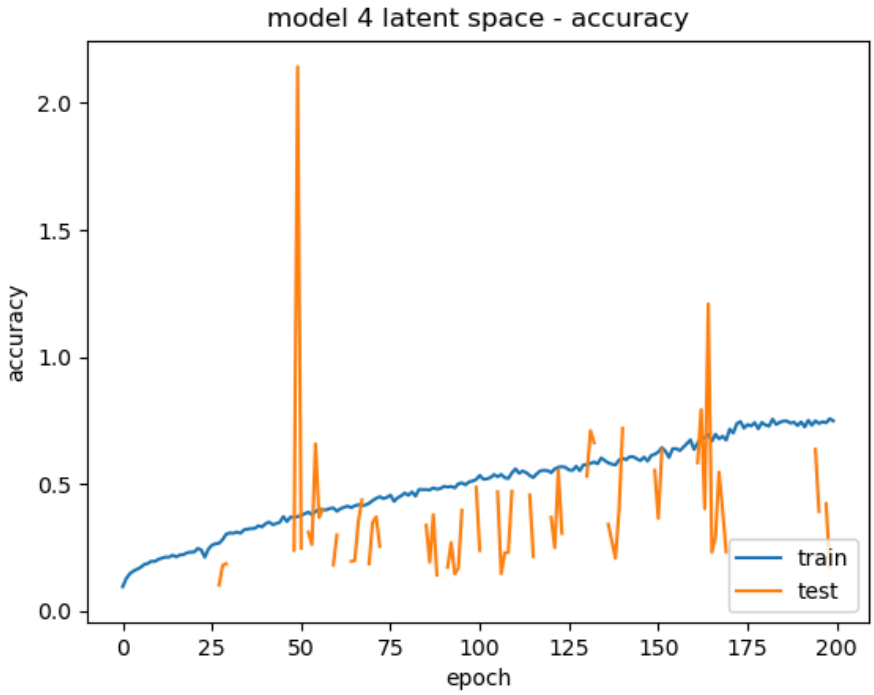
Model 2 – only Energy data:



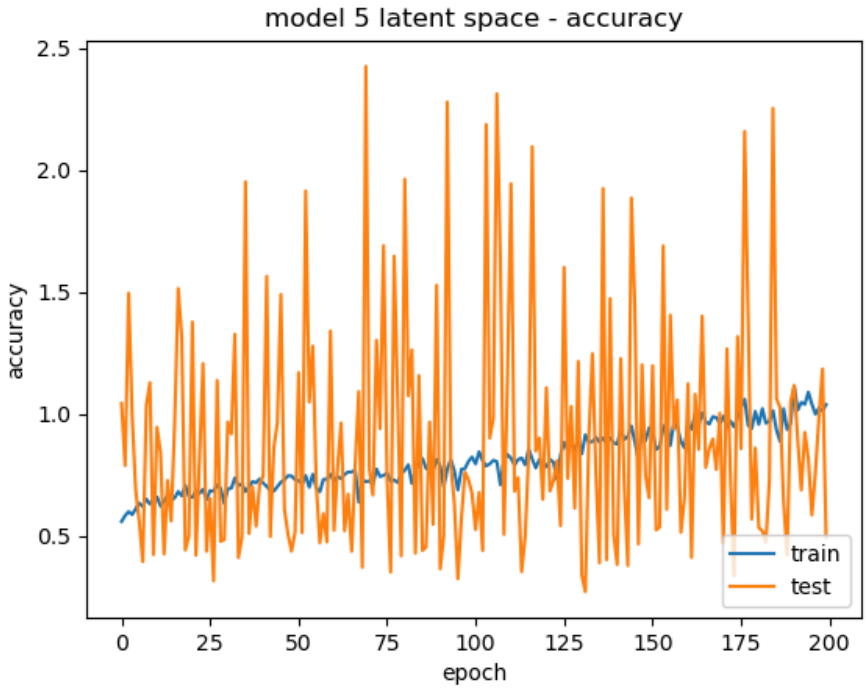
Model 3 – Thermal and Aux.



Model 4 – Energy and Aux.



Model 5 – Energy and Thermal



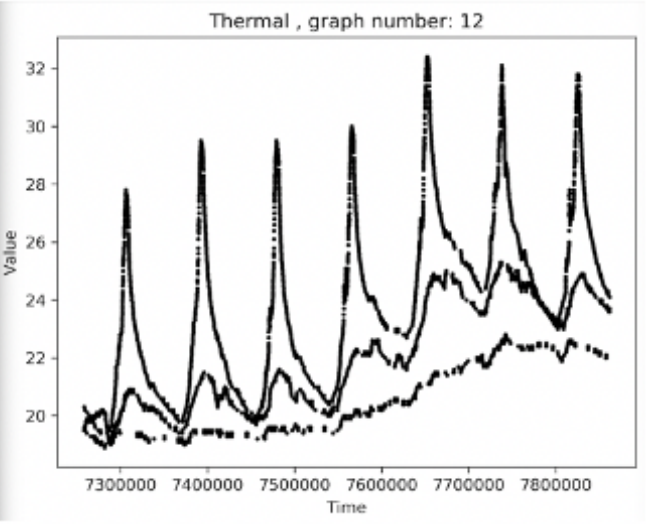
Model 6 – Energy, Thermal and Aux.

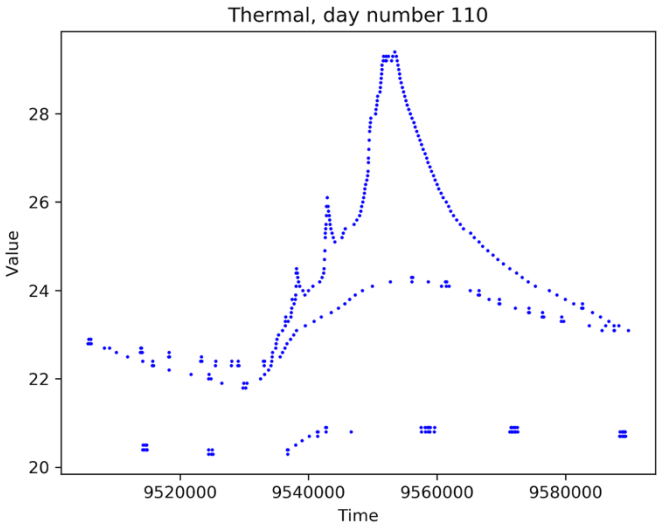
We can see that we can see an increase in the learning of the latent space, as well as the fact that we could learn a latent space from within our sensor.
The number of Aux dataset is much bigger than the energy and thermal.
The dataset of the Aux is much harder to reconstruct, since first of all it is mainly binary numbers originally, secondly there was not a big pattern to learn, in contrast to the thermal or energy data set, where we could see a clear pattern.
Since model number 5 produced the best test accuracy for the reconstruction and the most fluid\ continues one, it means there weren't many null values in the reconstruction, that means it was the best model for the reconstruction – as a results of that, we decided to use only the thermal and energy for the output of the system,
Since first it was the most correlated ones, and secondly, those two were reconstructed the best.

Possible Output of the System -

After analyzing our data, we decided we want to define a task for the system to learn, as that the reconstruction of the data will not be the only results. We decided that we would want to predict the time in the day that the thermal sensor would be at its peak.

Let's recall for example pictures of the samples of the thermal per week and per day.



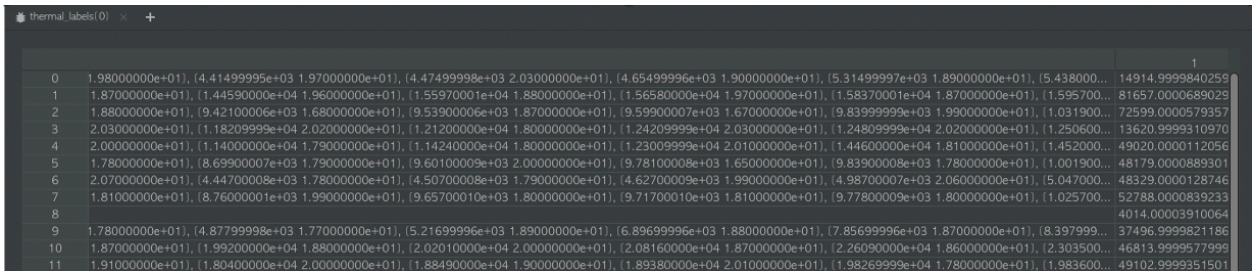


so, the task defined is given data from the sensors, we would want to predict the time in the day the thermal sensor would be at its peak.

We can derive many useful information from that such as if the thermal is at its peak, should we turn on a specific appliance etc.

For us to be able to predict, we labeled some of the thermal data. Given a 1D array of a unified grid, and a label of float representing the time in the day.

As shown in the picture below.

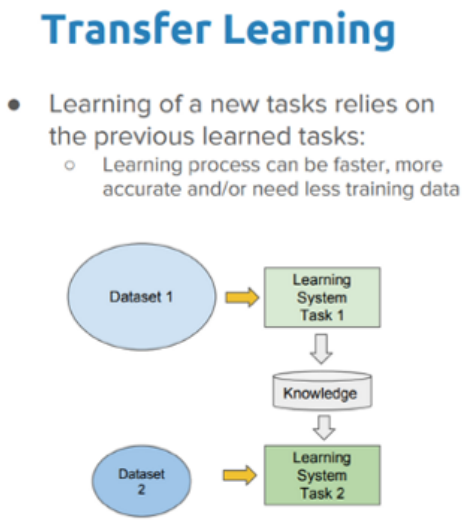


thermal_labels[0]

		1
0	1.98000000e+01), (4.41499995e+03 1.97000000e+01), (4.47499998e+03 2.03000000e+01), (4.65499996e+03 1.90000000e+01), (5.31499997e+03 1.89000000e+01), (5.438000...	14914.9999840259
1	1.87000000e+01), (1.44590000e+04 1.96000000e+01), (1.55970001e+04 1.88000000e+01), (1.56580000e+04 1.97000000e+01), (1.58370001e+04 1.87000000e+01), (1.595700...	81657.0000689029
2	1.88000000e+01), (9.42100006e+03 1.68000000e+01), (9.53900006e+03 1.87000000e+01), (9.59900007e+03 1.67000000e+01), (9.83999999e+03 1.99000000e+01), (1.031900...	72599.0000579357
3	2.03000000e+01), (1.18209999e+04 2.02000000e+01), (1.21200000e+04 1.80000000e+01), (1.24209999e+04 2.03000000e+01), (1.24809999e+04 2.02000000e+01), (1.250600...	13620.9999310970
4	2.00000000e+01), (1.14000000e+04 1.79000000e+01), (1.14240000e+04 1.80000000e+01), (1.23009999e+04 2.01000000e+01), (1.44600000e+04 1.81000000e+01), (1.452000...	49020.0000112056
5	1.78000000e+01), (8.69900007e+03 1.79000000e+01), (9.60100009e+03 2.00000000e+01), (9.78100008e+03 1.65000000e+01), (9.83900008e+03 1.78000000e+01), (1.001900...	48179.0000889301
6	2.07000000e+01), (4.44700008e+03 1.78000000e+01), (4.50700008e+03 1.79000000e+01), (4.62700009e+03 1.99000000e+01), (4.98700007e+03 2.06000000e+01), (5.047000...	48329.0000128746
7	1.81000000e+01), (8.76000001e+03 1.99000000e+01), (9.65700010e+03 1.80000000e+01), (9.71700010e+03 1.81000000e+01), (9.77800009e+03 1.80000000e+01), (1.025700...	52788.0000839233
8		4014.00003910064
9	1.78000000e+01), (4.87799998e+03 1.77000000e+01), (5.21699996e+03 1.89000000e+01), (6.89699996e+03 1.88000000e+01), (7.85699996e+03 1.87000000e+01), (8.397999...	37496.9999821186
10	1.87000000e+01), (1.99200000e+04 1.88000000e+01), (2.02010000e+04 2.00000000e+01), (2.08160000e+04 1.87000000e+01), (2.26090000e+04 1.86000000e+01), (2.303500...	46813.9999577999
11	1.91000000e+01), (1.80400000e+04 2.00000000e+01), (1.88490000e+04 1.90000000e+01), (1.89380000e+04 2.01000000e+01), (1.98269999e+04 1.78000000e+01), (1.983600...	49102.9999351501

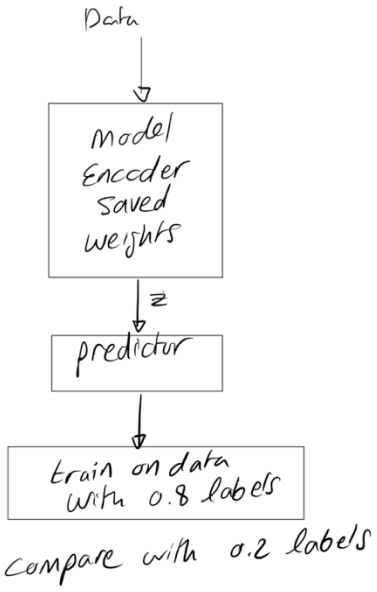
The Design of the system –

In order to leverage the initial training, we have done with the VAE, we decided to apply transfer learning [7][8].
Transfer learning - is a research problem in machine learning (ML) that focuses on storing knowledge gained while solving one problem and applying it to a different but related problem.



In our case, transfer learning helped us to design our system. We wanted to leverage the training phase we have already done earlier (the reconstruction of the data from within our own latent space) therefore the following scheme was thought of:

Partially loading a model or loading a partial model are common scenarios when transfer learning or training a new complex model. Leveraging trained parameters, even if only a few are usable, will help to warmstart the training process and hopefully help the model converge much faster than training from scratch.



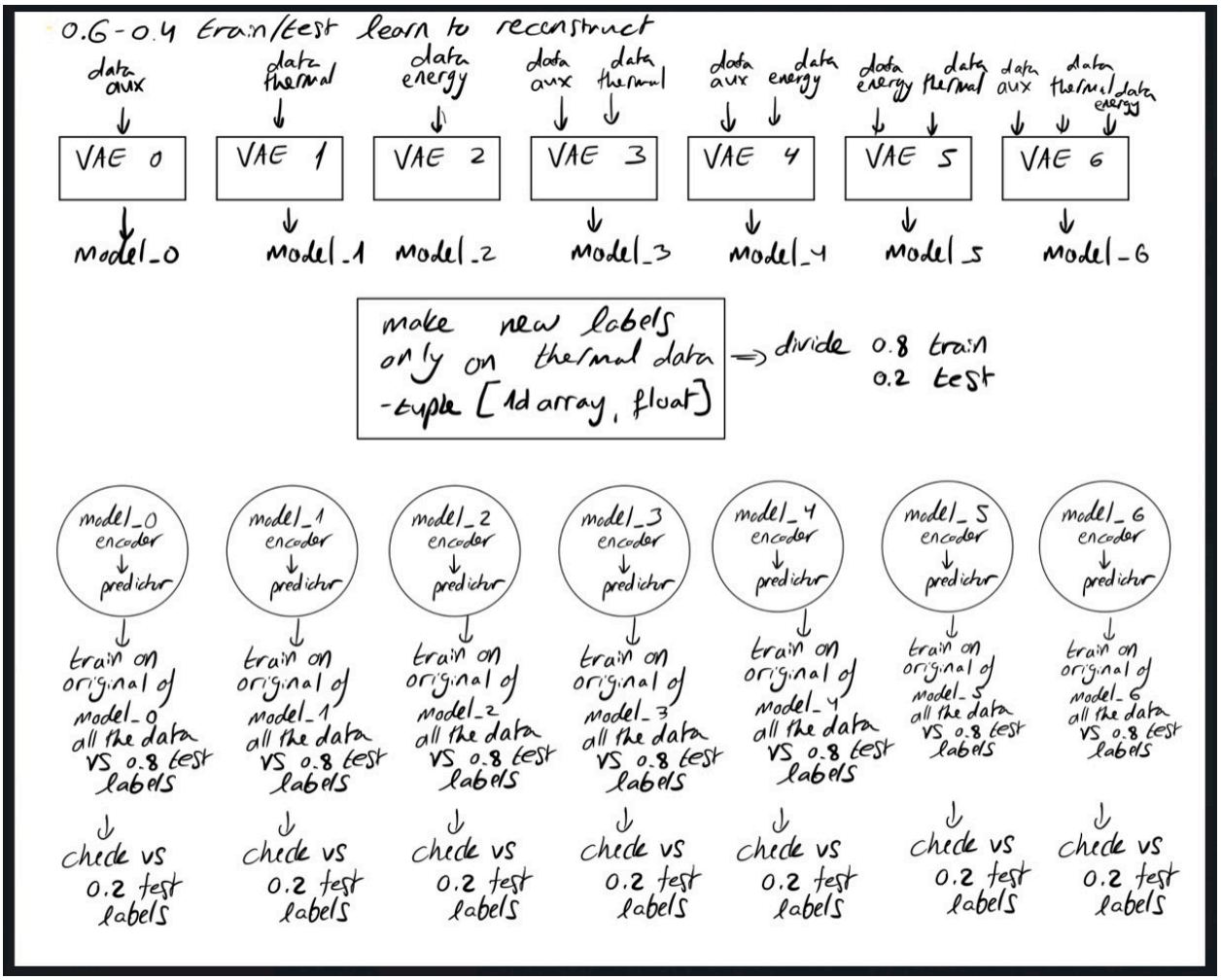
The Predictor NN would consists at its base the saved model from the initial training phase. We would append a linear NN to the flattened output of the decoder, while freezing the encoder, and not using it.
The code is as followed:

```
class Predictor(nn.Module):
    def __init__(self):
        super(Predictor, self).__init__()
        self.fc1 = nn.Linear(1306624, 64)
        self.fc2 = torch.nn.LeakyReLU(0.2, inplace=True)
        self.fc3 = nn.Linear(64, 1)
    def forward(self, x):
        x = self.fc1(x)
        x = self.fc2(x)
        x = self.fc3(x)
        return x


class Encoded_Predictor(nn.Module):
    def __init__(self, modelA, modelB):
        super(Encoded_Predictor, self).__init__()
        self.modelA = modelA
        self.modelB = modelB

    def forward(self, x1):
        x1 = torch.flatten(self.modelA.module.features_encoder(x1))
        x2 = self.modelB(x1)
        return x2
```

As we can observe, the flattened “Z” of the VAE encoder, is the input to the Predictor module. We would train the following system with an MSE loss, and check our prediction vs our labels, while each time changing the inputs of the models.
This time, the inputs of the models are two different parameters.

The first parameter, is the saved model on which we trained, including all of the saved weights. The second parameter would be the data corresponding to that model, this time 100% of it. The test labels for training would be 80% of the labels we have, while the comparison would be made vs 20% of the labels.

Figure of the above explanation – next page.


0.6 - 0.4 train/test learn to reconstruct
data aux
data thermal
data energy
data aux
data thermal
data aux
data energy
data energy
data thermal
data aux
data thermal
data energy
VAE 0
VAE 1
VAE 2
VAE 3
VAE 4
VAE 5
VAE 6
model_0
model_1
model_2
model_3
model_4
model_5
model_6
make new labels only on thermal data - tuple [1d array, float]
=> divide 0.8 train
0.2 test
model_0 encoder
predictor
model_1 encoder
predictor
model_2 encoder
predictor
model_3 encoder
predictor
model_4 encoder
predictor
model_5 encoder
predictor
model_6 encoder
predictor
train on original of model_0 all the data VS 0.8 test labels
train on original of model_1 all the data VS 0.8 test labels
train on original of model_2 all the data VS 0.8 test labels
train on original of model_3 all the data VS 0.8 test labels
train on original of model_4 all the data VS 0.8 test labels
train on original of model_5 all the data VS 0.8 test labels
train on original of model_6 all the data VS 0.8 test labels
check vs 0.2 test labels
check vs 0.2 test labels
check vs 0.2 test labels
check vs 0.2 test labels
check vs 0.2 test labels
check vs 0.2 test labels
check vs 0.2 test labels

Results and conclusions –

We trained models 1,2 and 5 and checked their results, while evaluating the models' prediction of the time in the day of the peak value.

We wanted to check several things:

1. If a pre trained VAE assisted with the training of the appended model.
2. We wanted to predict the time during the day of the peak, corresponding between the thermal and the energy sensors.
3. We wanted to check if the addition of sensors improves the prediction of the model.

Therefore, we will show several graphs of the three models.

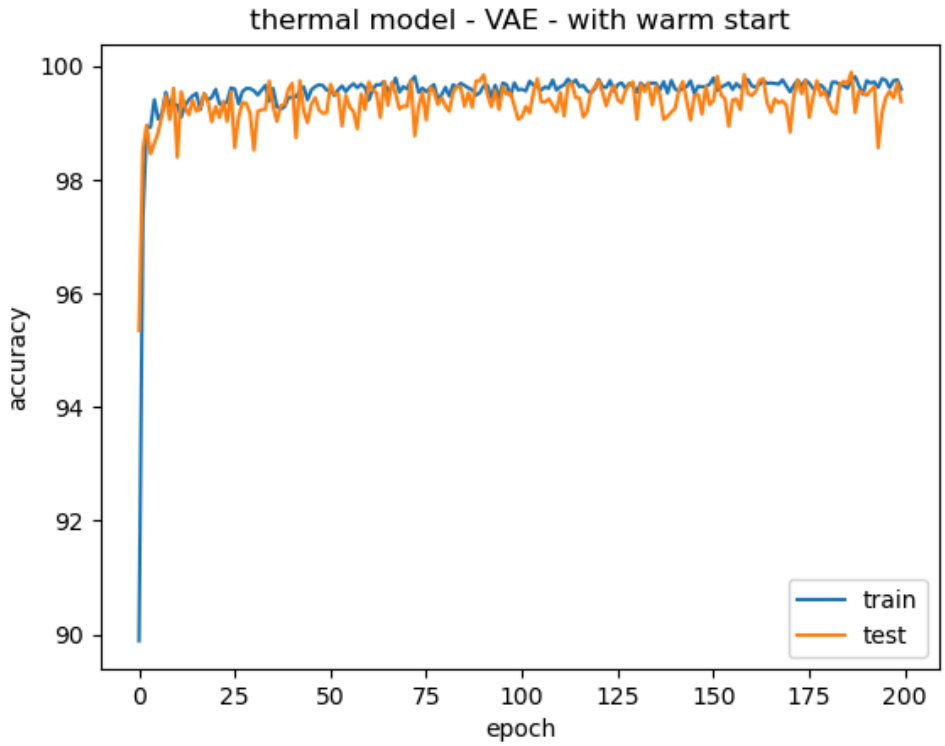
The three models:

1. Thermal sensor only.
2. Energy sensor only.
3. Energy and Thermal together.

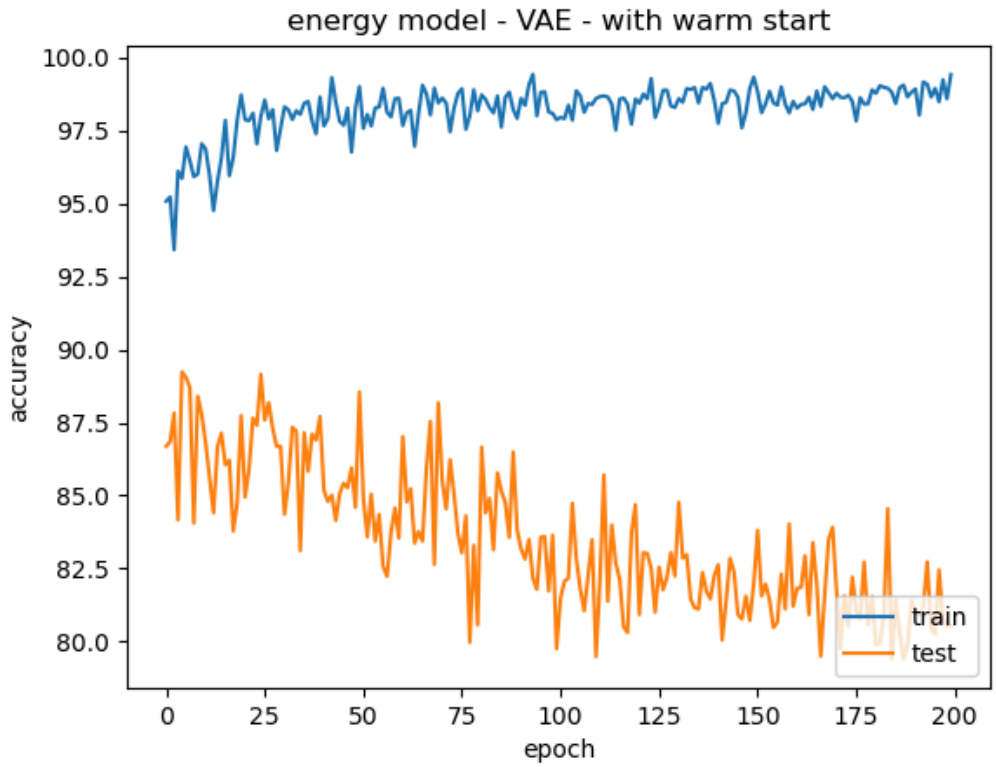
Accuracy is defined as the return value of the mean square error between the mean of a batch (4 samples) - between the prediction and the label itself. The models were trained for 200 epochs. As for example if the accuracy is 95%, it means the time is within 5% of the original label.

The first batch of graphs is the accuracy of the prediction for the three different models. Using the pre trained VAE models (what is called warm start).

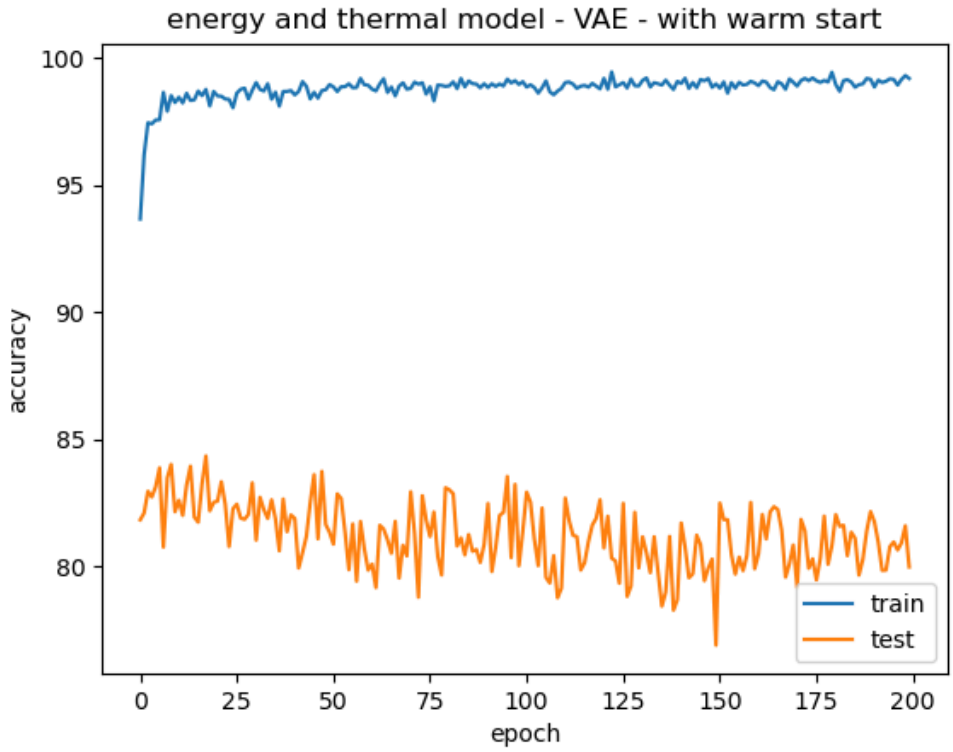
Thermal sensor –



Energy Sensor

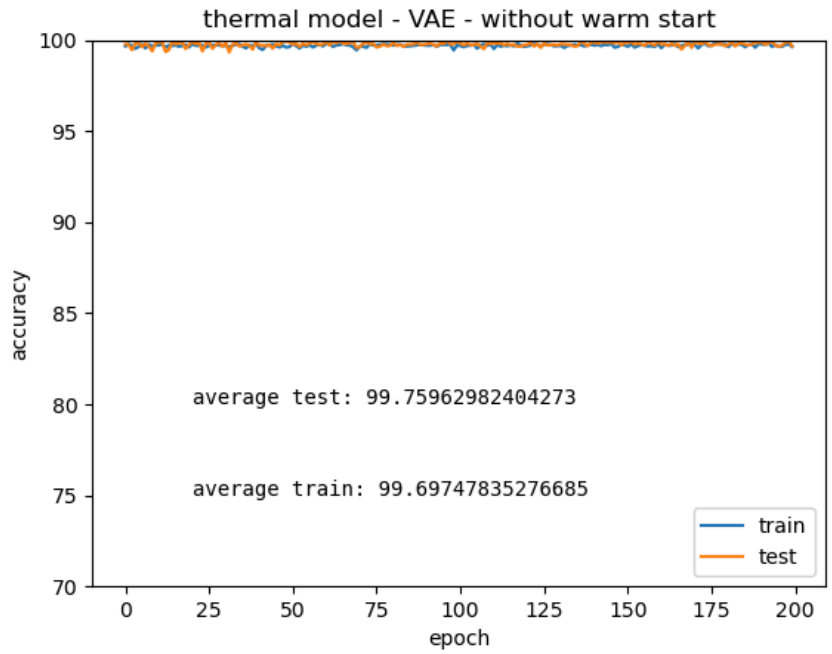


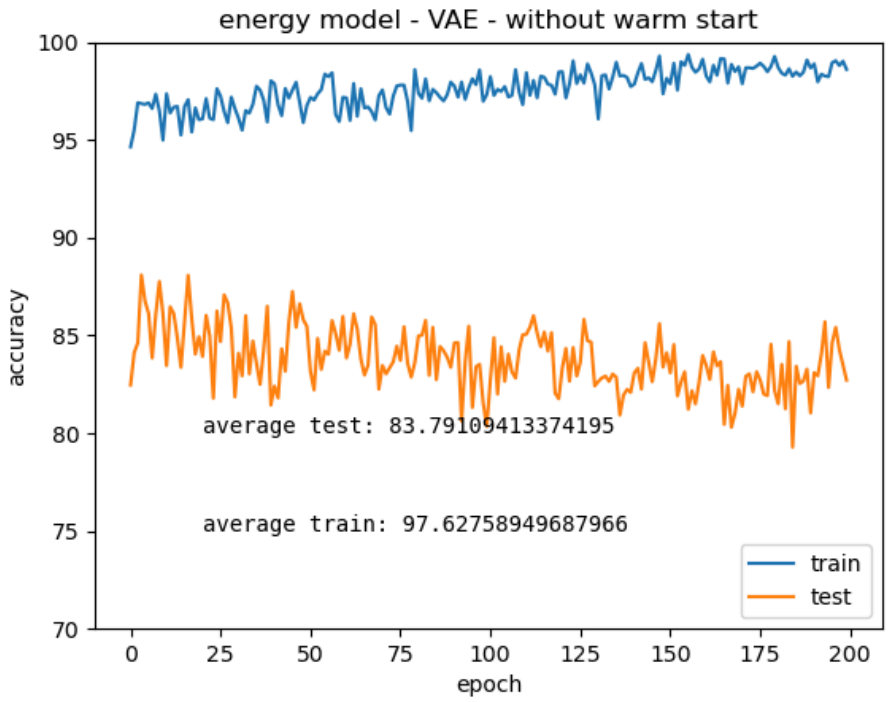
Energy and Thermal sensor

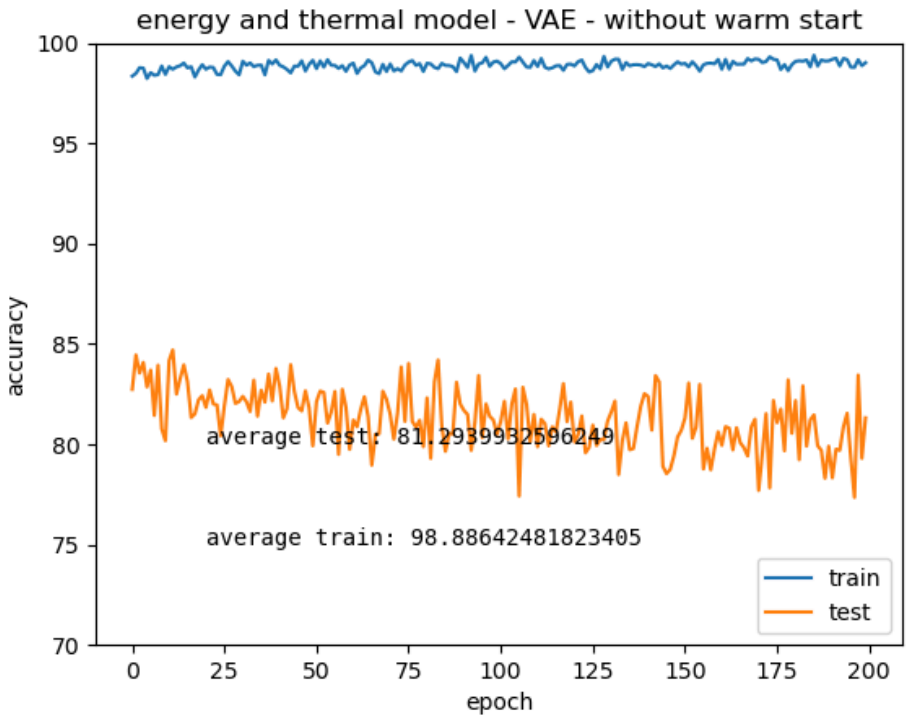


As we can observe from the graph, the correlation between the sensors is high. When training only on the thermal sensor's data, the training is at its best, but it might also be due to overfitting, since as we know we do not have a lot of data.
If we try to predict the thermal's peak from solely the energy sensor, we get 85% on the test set. This can show that there is a high correlation among those sensors. Moreover, the thermal and energy sensors, while working together, the test prediction is at its lowest, probably because it takes energy labels as well, and reduces the over fitting.

The second batch of graphs is the accuracy as well, of an untrained VAE (first initialized for training without warm start) and the appended predictor.

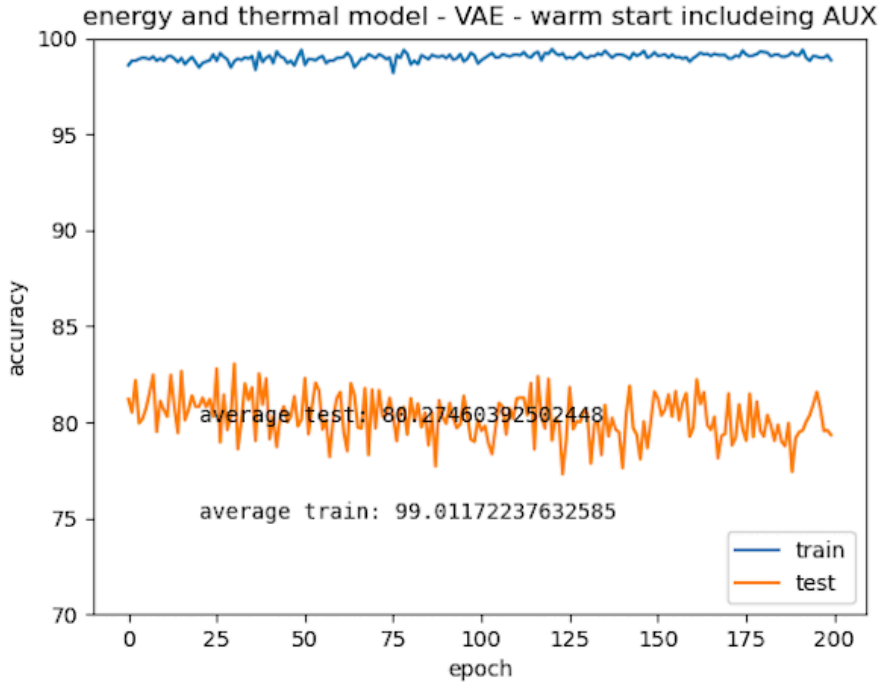






As we can observe from the graph, the correlation between the sensors is again high. When training only on the thermal sensor's data, the training is at its best but a bit lower then before, but still overfitting.
If we try to predict the peak of the thermal from solely the energy sensor, we drop to 83% on the test set. This can show that initial transfer learning improved the overall learning process. Moreover, the thermal and energy sensors, while working together, the test prediction is at its lowest, probably because it takes energy labels as well, and reduces the over fitting.

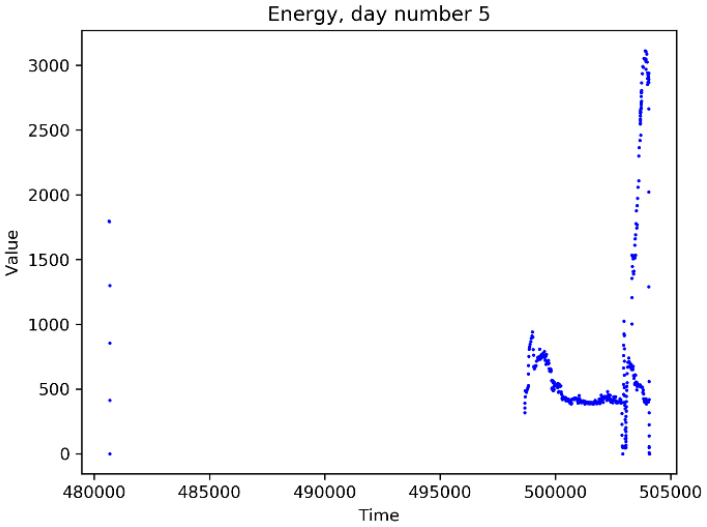
The next graph shows training on both of the datasets, the thermal and the energy, but this time, the pre-trained model, was also trained on the AUX data set, that is mainly binary.

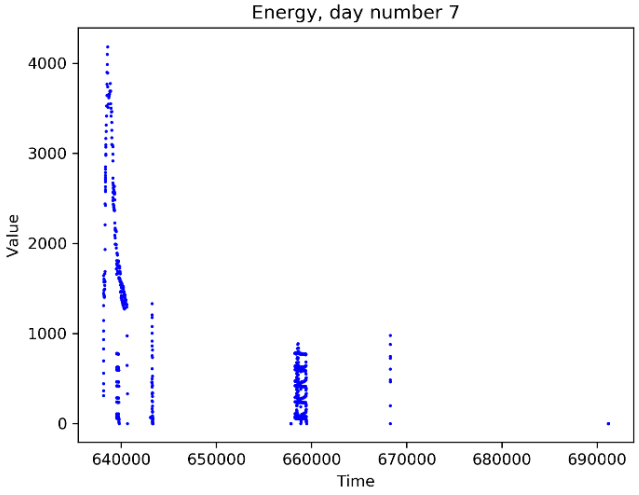


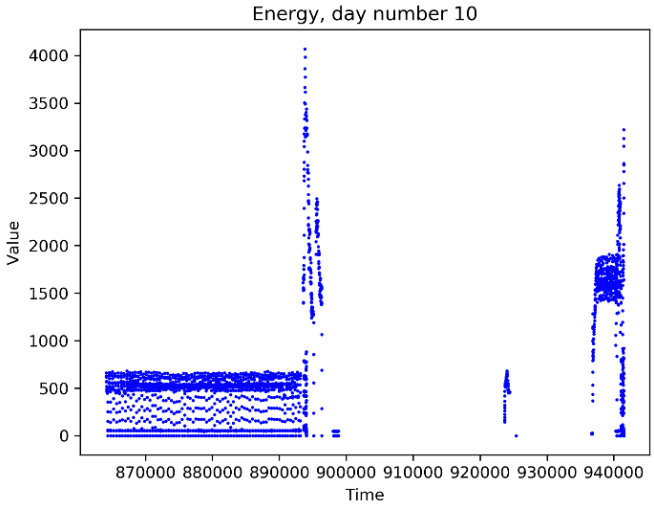
We can observe from the graph, that the more data we add, in that case the AUX data, the overfitting reduces, but still we can still predict at a high value.

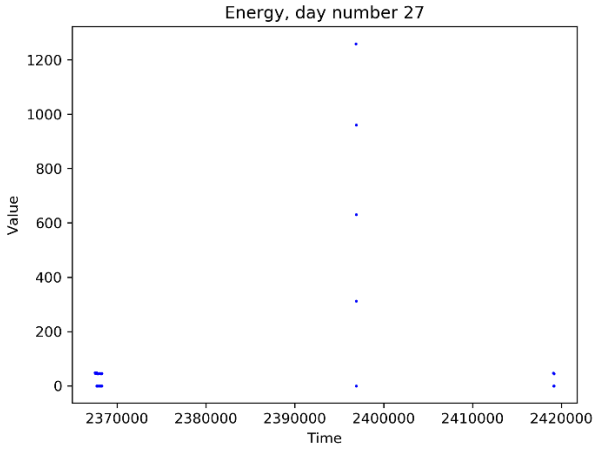
We should note that we didn't freeze the weights, so each time we used the VAE, the learning processed continued.

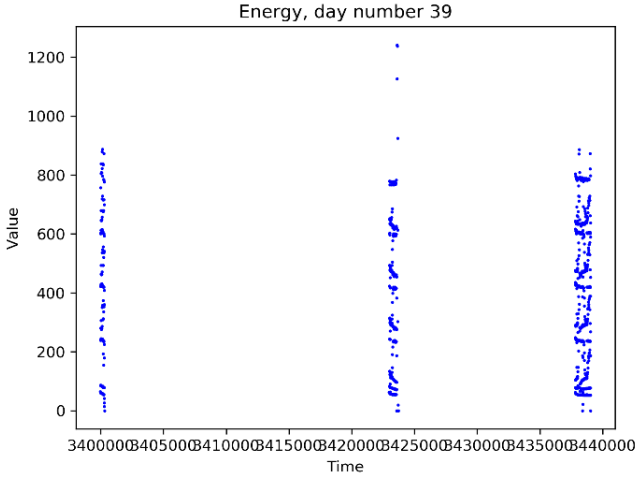
In addition we wished to make a prediction on the energy labels, but we came to a conclusion of several problems.
The data of the energy was very problematic for us to predict anything upon it, mainly due to the lack of samples, the rating samples was very different from day to day. Moreover, the sensor didn't work for all of the days, in addition to most of the days looked pretty different from each other. Presented below are examples of days of the energy, that look very differently from one another.

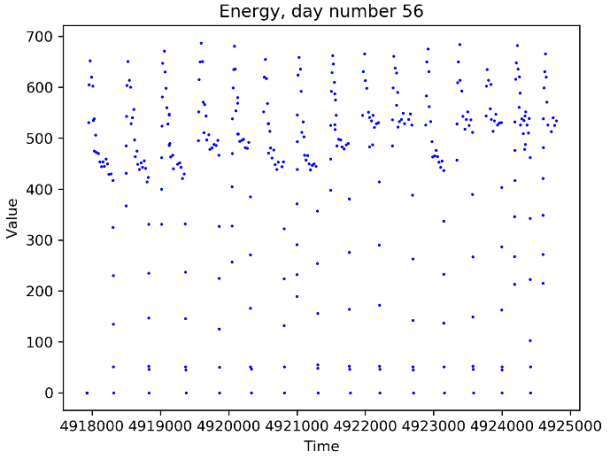


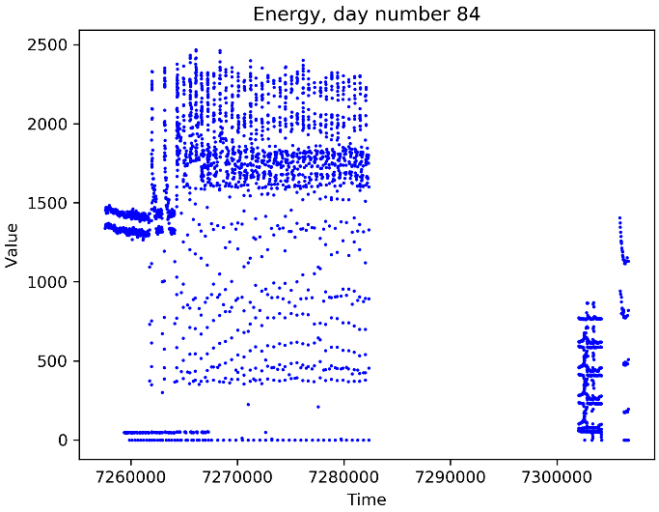


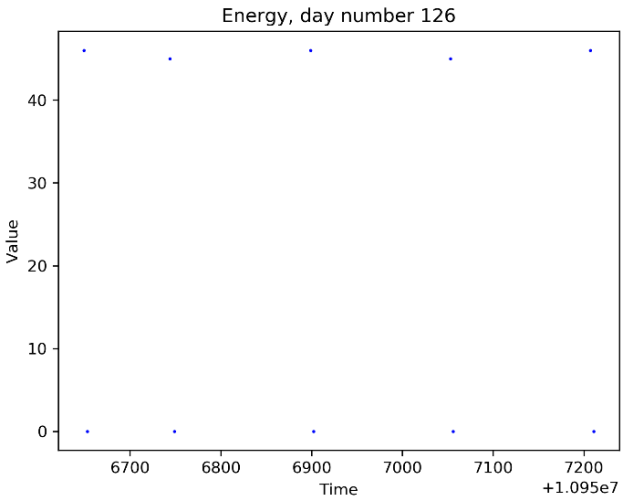












As we can see from the graphs above, the data is not sampled at the same rate, some of the days don't even have samples at all. In addition, it is very hard to predict any value based on so little unbalanced data, therefore we came to a conclusion, that given the amount of data it is not possible at the moment.
In the future, if we receive more data, it might be achievable to predict based on the energy.

Future related research -

If we had a larger set of data, we would have been able to define and predict much more useful information. In addition, it might be worth to check more direction of pre-processing phase for future analysis of the data.
Moreover, feature engineering can be used to produce more features based on the features currently sampled from the sensors. Those features could be proved to be useful.
The main setback of the prediction is over fitting of the data, since there is not much data to be dealt with, although the main goal of the project was to successfully do the whole process, from making a dataset to transforming it into a working data loader. Using that data loader, designing a system from scratch that can learn and predict something that can be useful and valuable.

We came to a prediction of the thermal's peak during the day, but we could have also predicted other metrics, which is left for future research.

References

[1] Multidimensional Scaling –
https://ncss-wpengine.netdna-ssl.com/wp-content/themes/ncss/pdf/Procedures/NCSS/Multidimensional_Scaling.pdf

[2] Learn to Combine Modalities in Multimodal Deep Learning –
https://arxiv.org/pdf/1805.11730.pdf

[3] Tutorial on Variational Autoencoders https://arxiv.org/abs/1606.05908

[4] VAE - https://arxiv.org/pdf/1312.6114.pdf

[5] Event-based Asynchronous Sparse Convolutional Network
http://rpg.ifi.uzh.ch/docs/ECCV20_Messikommer.pdf

[6] End-to-End Learning of Representations for Asynchronous Event-Based

Datahttps://arxiv.org/abs/1904.08245

[7] How transferable are features in deep neural networks? https://arxiv.org/abs/1411.1792

[8] A survey on transfer learning

https://www.cse.ust.hk/~qyang/Docs/2009/tkde_transfer_learning.pdf