Magnus Totland

Detection of leakages in a water distribution network using an autoencoder

Hovedoppgave i Bygg og Miljøteknikk
Veileder: Franz Tscheikner-Gratl
Medveileder: David Steffelbauer
Juni 2022

Hovedoppgave

NTNU
Norges teknisk-naturvitenskapelige universitet
Fakultet for ingeniørvitenskap
Institutt for bygg- og miljøteknikk

NTNU
Norwegian University of Science and Technology

Magnus Totland

Detection of leakages in a water distribution network using an autoencoder

Hovedoppgave i Bygg og Miljøteknikk
Veileder: Franz Tscheikner-Gratl
Medveileder: David Steffelbauer
Juni 2022

Norges teknisk-naturvitenskapelige universitet
Fakultet for ingeniørvitenskap
Institutt for bygg- og miljøteknikk

Detection of leakages in a water distribution network using an autoencoder.

Abstrakt:

Lekkasje fra ledningsnett er et problem i en rekke byer i verden og vannmangel er forventet å bli et stadig økende problem på verdensbasis grunnet klimaendringer. Ett eksempel på et ledningsnett som sliter med høye lekkasjetall er i Oslo.

I denne oppgaven testes en autoenkoder på ett datasett som inneholder trykkverdier fra sensorer plassert i et ledningsnett, med mål om å avgjøre når en lekkasje oppstår. Det testes på 14 forskjellige lekkasjescenarioer. For å håndtere dette problemet med avviksdeteksjon brukes en autoenkoder. En autoenkoder brukes på grunn av dens evne til å rekonstruere sin inngangsdata, slik at den kan brukes til avviksdeteksjon når inngang- og utgangsdata sammenlignes. Rekonstruksjonsfeilen som brukes til å utlede avvik er da forskjellen mellom inngang og utgang for hver sensor, og en topp i rekonstruksjonsfeilen antyder at det har oppstått en lekkasje.

Ett hovedspørsmål og en rekke mindre spørsmål vil bli besvart i denne oppgaven. Det første og viktigste spørsmålet er å avgjøre om autoenkoderen kan oppdage lekkasjer i det hele tatt. Etter det, ble to alternative fremgangsmåter for trening av autoenkoderen undersøkt. For det tredje, ettersom ett nevron tilsvarer en sensor, og en sensors plassering er kjent, ble det også gjort en undersøkelse om hvorvidt autoenkoderen kan utføre lekkasjelokalisering. For det fjerde, testing på varierende trenings-, validerings- og testtidsintervaller for autoenkoderen for å utlede hvor mye data som er tilstrekkelig for at autoenkoderen fortsatt lykkes med å oppdage lekkasjene. Til slutt, etter å ha funnet lekkasjene, ble det gjort et forsøk på å redusere antall nevroner i autoenkoderen, noe som vil tilsvare å ha færre sensorer i et ledningsnett, for så å undersøke om pålitelig lekkasjedeteksjon fortsatt ville være mulig.

Autoenkoderen lykkes med å oppdage alle 6 rørene som sprekker mellom en halvtime og 18 timer. For de lekkasjene som vokser kontinuerlig, lykkes autoenkoderen med å finne lekkasjen når den når omtrentlig 4.5 $\frac{m^3}{h}$. Autoenkoderen lykkes ikke med å finne de aller minste lekkasjene i ledningsnettet. Videre, autoenkoderen presterte litt verre med en reduksjon i inngangsdata og sensorer, men likevel bedre enn forventet i utgangspunktet.

Abstract:

Leakage of drinking water from distribution networks is a problem for several in the world and water scarcity which is expected to increase worldwide because of climate changes. One example of a water distribution network struggling with excessive leakage is found in Oslo, Norway.

In this thesis, an autoencoder is tested on an artificial dataset containing pressure values from a variety of different leakage scenarios from a water distribution network with the goal of detecting when those leaks occur. Finding those leaks is in practice an anomaly detection problem and an autoencoder can be used for this purpose, because of its ability to reconstruct its own input, allowing it to be used for anomaly detection when input and output is compared. The reconstruction error used to deduce anomalies is then the difference between the input and output for each given sensor, and a spike in the reconstruction error implies a leak has occurred.

One primary, and several secondary research questions relating to the usage of and how well the autoencoder performs are addressed in this thesis. The first and most important question was whether the autoencoder can indeed detect leaks at all, and to deduce which configuration of the autoencoder and its hyperparameters worked well for this task. Secondly, an investigation of different training approaches to the autoencoder was carried out. Thirdly, inspecting whether the autoencoder can perform leak localization as one neuron corresponds to one sensor with a known location. Fourth, testing on varying training, validation, and testing time intervals for the autoencoder to deduce how many samples are required until the autoencoder fails at detecting the leaks. Finally, after finding the leaks, an attempt was made to reduce the number of neurons in the autoencoder, which would be akin to have fewer sensors in the water distribution network, to investigate whether reliable leak detection would still be possible.

The dataset used in this project, is one created artificially, based on the hypothetical L-Town, from the BattLeDIM competition. It contains 14 different leaks of varying flow.

The 6 pipe bursts were all detected within about 18 hours of the pipe breaking, the shortest being within 30 minutes. The 3 of the 4 leaks with a continuous volume increase were discovered once they reached a volume of about 4.5 $\frac{m^3}{h}$. The 4 incipient leaks of a small size and slow growth remained undiscovered. The autoencoder's performance deteriorated slightly as the number of training samples and neuron-count was reduced, but at a slower rate than initially expected.

Tables:

Table 1 shows the different autoencoder architectures typically tested. 10
Table 2 Overview of different optimizers, loss and activation functions tested in this project. 10
Table 3 The various values tested for the different hyperparameters 10
Table 4 Results 15
Table 5 The autoencoder architecture and hyperparameters associated with the best performing configuration 17
Table 6 Overview of quick burst leaks 18
Table 7 Overview of slow increase leaks 19
Table 8 Overview of incipient leaks 21
Table 9 Results from 7-and 5-day time intervals, and 10- and 15-minute resampling 26
Table 10 Initial results from area A with 28 sensors, with standard deviation multiplier set to 2.5. 29
Table 11 Performance of smallest autoencoder with 5 sensors. 30
Table 12 Brief overview of some ML terminology 1
Table 13 Brief overview of the purpose of different hyperparameters 2

Figures:

Figure 1 Generic illustration of an autoencoder 3
Figure 2 Schematic of the script used in this project. 5
Figure 3 Flowchart for the threshold function used. 7
Figure 4 An illustration of a frequently used autoencoder during initial development. 9
Figure 5 Illustration of L-Town. 11
Figure 6 The different leak classifications: quick burst, slow increase, and incipient leak 12
Figure 7 The location of pipes with leaks found in L-Town 13
Figure 8 Shows all the different leaks in L-Town for the year 2018. 13
Figure 9 Red dots indicate pressure sensors in L-Town. Green dots are reservoirs, and the teal dot is a pump. 14
Figure 10 Rolling mean on a per day basis for the reconstruction error from sensors n469, n722, n726 and n752 with the leak in pipe 183. 18
Figure 11 Rolling mean on a per day basis for the reconstruction error from sensors n105, n114, n616 and n726 with the leak in pipe 461. 19
Figure 12 A comparison between sensors with a rolling mean on a per day basis for an incipient leak in pipe 810. 21
Figure 13 Accumulated and shifted training approach comparison for a burst leak in pipe 183. 22
Figure 14 Comparison between shifted and accumulated training for a slow increase pipe leak in pipe 461. 23
Figure 15 Shows all the sensors' reconstruction errors together on a rolling mean per day basis for leak in pipe 866. 24

Figure 16 Rolling mean on a per day basis for the reconstruction error from sensors n286, n469, and n740 with the leak in pipe 866. 24
Figure 17 Detected leak on a two-week training interval with 10-minute resampling in pipe 369. 26
Figure 18 Shows leak in pipe 673, produced with a 7-day run, with a one-day rolling mean. 27
Figure 19 Shows an undetected leak in pipe 369. 30
Figure 20 shows a false positive detected in pipe 538 at the first red x, the second cross shows when the leak actually does happen. 31
Figure 21 Shows a successful detection of a slow increase leak in pipe 628. 31
Figure 22 Comparison between close and far away sensors for leak in pipe 810. First cross when it starts, second when peak is reached. 33
Figure 23 Comparison between close and far away sensors for leak in pipe 654. First cross when it starts, second when peak is reached. 34

Abbreviations:

ANN.	Artificial neural network
CPU.	Central processing unit
DMA.	District metered area
GPU.	Graphics processing unit
ML.	Machine learning
MSE.	Mean square error
MNF.	Minimum night flow
WDN.	Water distribution network

1. Introduction:

When cracks appear in water distribution pipes, water starts leaking, compromising the water distribution network (WDN) in several ways. For one, treated water is being wasted during leaks. Secondly, excessive leakage will reduce the pressure within the system, increasing pumping and energy demand to transport the same amount of water to the customer. Thirdly, in low pressure zones there is a risk of contamination of the water supply if a leak has occurred. The estimated cost for water loss through leakage is about 39 billion USD annually, on a worldwide scale, not accounting for the related damage to roads and other infrastructure associated with water leakages (Liemberger & Wyatt, 2018). Worldwide, water scarcity is expected to increase, further underlining the importance of avoiding water loss due to leakages (Unicef, 2022).

To address this problem, leak detection is routinely done to search for the sources of water loss in WDNs and is currently done in a variety of ways by water utilities. For example, acoustic based leak detection methods include noise loggers that are placed in manholes, handheld listening devices such as acoustic correlators to detect and pinpoint exact location of leaks, and mobile acoustic loggers to be placed in a network for a short period of time to listen for leaks if one is expected in the area (Puust, et al., 2010). Feedback from consumers is also relied on. Despite these efforts, municipalities in Norway continue to see a worryingly high amount of water loss percentage. One example being Oslo's estimated 37 % water loss (Hult, 2022), which is in stark contrast to Denmark's 7 % (Fisher, 2016).

In the research literature regarding leak detection in WDNs, there is a distinction in approaches as to how to find the leaks more effectively. One is model based, and the other is data driven (Hu, et al., 2021) (Romero-Ben, et al., 2022). In short, model based implies that there is some model of the water distribution network that is reliable and a divergence between the expected pressures or flows and the actual measured pressures and flows are indications of a leak. A data-driven leak detection approach relies on large quantities of data. The data can be pressures, acoustics, flows, tank levels or pumping volumes and frequencies. The data is then investigated for anomalies or particular patterns indicating an abnormality of some kind, which would indicate a leak. It should also be noted that there have been studies in which the methodology utilizes a dual or mixed leak detection approach (Soldevila, et al., 2016) (Daniel, et al., 2022).

Each approach has their respective advantages and disadvantages. For one, the model-based approach requires a well-calibrated hydraulic model of a distribution system to accurately predict what reasonable values for pressures and flows are across the system. Creating such a model is challenging, and the larger the water distribution system, the more challenging it is to develop such a model. The benefit of the model-based approach is that it is a straightforward method of detecting leaks, assuming the necessary calibrated model is available. The drawback is then of course that the creation of such a calibrated model is challenging, and it would need updating whenever the WDN changes. As such, the model-based approach is believed to be better suited for smaller water distribution networks.

As for data-driven leak detection methods, one is not dependent on a well-calibrated WDN model or very intricate knowledge of the functioning of the WDN of interest. Rather, as the name implies, one is dependent on data. The data collected from sensors scattered around the network will also, in a real-world scenario, include excessive noise and sensor malfunctions. As such, the quality of the data and/or the data preprocessing is an important aspect to consider before feeding the data into any type of algorithm as is typically done in the data-driven leak detection. Overall, a rule of thumb is that the data-driven leak detection approach is better suited for larger distribution networks, as it does not require any calibration or detailed model of the WDN. Although, where to draw the boundary between small and large is unclear in this context, nevertheless the larger the network, the stronger benefit one would presumably see if choosing to adapt the data-driven approach.

Machine learning (ML) is a research field that is a central part of many data-driven leak detection methods. With the help of data-collection efforts done by water utilities, flow and pressure data can allow the utilization of various algorithms that are able to detect anomalies. An autoencoder is one such ML algorithm that can be used for this purpose. The anomaly detection occurs when new data is supplied as input to a trained algorithm and a breach of the normalcy of the system is detected, indicating that a leak has appeared in the most recently processed input. ML for anomaly detection has shown itself useful in other industries, such as finance and medicine, and there is hope that it can be implemented successfully in the detection of leakages in WDNs as well (Anandakrishnan, et al., 2017). Furthermore, ML has been used in the context of WDNs already. In 2012, a team of researchers tested an artificial neural network (ANN) for the detection of real simulated (by opening fire hydrants) pipe bursts in a city in UK (Romano, et al., 2012). They showed that such a system could be used to identify these abnormal events in a fast and reliable manner with a low false positive rate. Another case study, with an online AI data analysis component searching for anomalies at the district meter area (DMA) level, was tested for one year. 36 % of alerts were shown to correspond to repairs done by the municipality or customer alerts (Mounce & Boxall, 2010). Furthermore, researchers have succeeded in detecting small leaks in a testbed setting, with the help of an autoencoder, showing the potential promise of this ML algorithm for this purpose (Cody, et al., 2020). A review paper of data driven approaches to detection of pipe bursts by (Wu & Liu, 2016) show that a common challenge across data-driven approaches to leak detection is a high false positive rate in a real-world setting. In addition, there is a need to further improve the lowest detectable size of a leak. In the context of burst detection, detecting the leaks as soon as possible is paramount. Furthermore, as with any ML algorithm, the rule of thumb that the more data the better holds true, and as mentioned, sensor- or data transmittance failure exist as a potential pitfall for data-driven leak detection methods. In this project, these problems are largely circumvented by the use of artificial data, which is appropriate as this is a preliminary investigation as to whether an autoencoder can function as a leak detection tool.

An artificial neural network (ANN) is an integral part of an autoencoder and is often considered the most well-known type of ML algorithm. An ANN consists of a network of neurons with a web of links between them. The network has an input layer, a number of hidden layers and an output layer in the end. Colloquially one can say that the network learns, during the training stage, when its values shift through the network and depending on the degree the model's output matches the training data, the internal parameters of each neuron within the network is finetuned in order to predict the correct output more accurately. The technicalities of this learning process are beyond the scope of this thesis, and the curious reader is advised to consult the necessary literature. One recommendation is Deep Learning by Ian Goodfellow, available online for free at https://www.deeplearningbook.org/ (Goodfellow, et al., 2016).

An autoencoder consists of two such ANNs. One ANN is known as the encoder, and the other as the decoder. The defining characteristic of the autoencoder configuration is the bottle-neck layer in between the encoder and decoder. The bottle-neck layer has substantially fewer neurons than both input and output layers. Conceptually, the idea is that the bottle-neck layer functions to condense the information from the input layer, and from there, have the decoder try to reconstruct it, only working with the most important aspects of the input. In essence, the autoencoder is being trained to recreate its own input. A generic schematic of an autoencoder can be seen in figure 1.

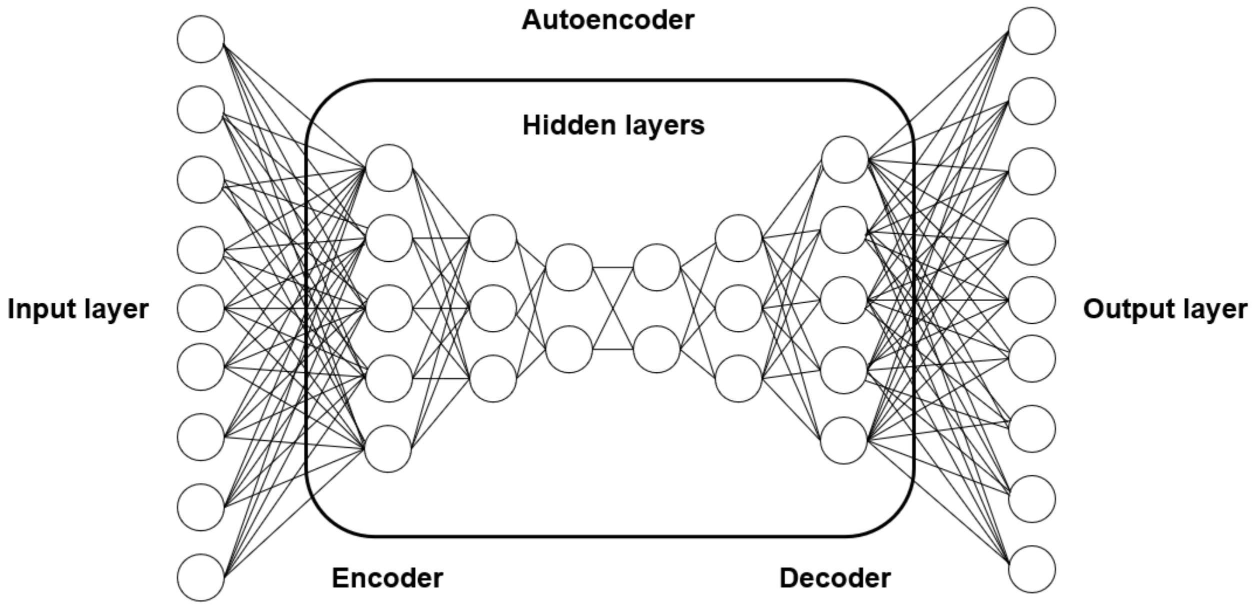

Figure 1 Generic illustration of an autoencoder

As mentioned, the ability to recreate its own input allows an autoencoder to be used for anomaly detection. Anomaly detection is the practice of utilizing various techniques and methods to detect occurrences that breach the perceived normalcy of a dataset. An anomaly can be a wide range of incidents such as abnormal values within a dataset, an image that doesn't match other images, credit card fraud or a leak in a water distribution system. The reconstruction error is then the difference, or mean square error (MSE), between a model's output and its input, after the training and validation stage is complete. If no leak had occurred, a model would predict the same as what has been given as input, resulting in a low reconstruction error. Subsequently, when a leak does occur, one would expect to see a spike in the reconstruction error which implies that an anomaly has occurred.

A benefit compared to other ML algorithms is that the autoencoder is an unsupervised ML algorithm, meaning that it works without the need of labelled data, as opposed to supervised ML algorithms. This is a necessity due to the fact that the accessibility of datasets containing leaks which are labelled for the training are often rather scarce.

Due to the fact the detection of leaks boils down to the aforementioned anomaly problem, in which the normal state of the WDN needs to be perceived, before a breach of this normalcy can occur, an autoencoder can be a fitting tool for the job. Autoencoders have been used for anomaly detection in a variety of different settings, beyond the aforementioned finance and medicine uses for ML in general. One example is the detection of abnormal radiation patterns done by (Ghawaly, et al., 2022), from the Oak Ridge National Laboratory in the US. Another example is the use of an autoencoder to detect the occurrences of seizures in patients with epilepsy (You, et al., 2021). The wide range of applications in which autoencoders could be used for anomaly detection prompted the need to also investigate their performance in the context of WDNs. Still, it was not known how much data would be necessary before any adequate detection of leaks could be done by an autoencoder. As such, there was a need to first use artificial data for three reasons. Firstly, to ensure that the data was reliable and high quality, meaning the circumstances in which the data was retrieved was known and accounted for. Second, to have some more leeway in the number of sensors to employ and the possibility of resampling to a larger timestep-size if needed. Third, as most data-driven leak detection methods struggle with a high false positive rate, the use of artificial data allows for the circumvention of data-preprocessing that would perhaps obfuscate the autoencoders performance at leak detection. Then, the aforementioned research questions followed naturally. The primary one being whether an autoencoder can indeed work for leak detection at all. The secondary research questions being to investigate how large of an influence the number of sensors employed in the autoencoder has, and to deduce the influence of training interval size and resampling at the autoencoder’s ability at leak detection.

In this thesis, after the methodology, the results of the main research question as to how well an autoencoder worked for the detection of leaks will be presented. Next, a number of smaller quirks and features of the autoencoder are laid out. First, a comparison between two different approaches to the training of the autoencoder. Second, leak localization. Third, the influence of training interval size and resampling, before ending with the results obtained when reducing the number of sensors in the autoencoder. After the results, a discussion as to different ways one might improve the results further is presented in 3.6 and 3.7. Towards the end, limitations of this project and recommendations for future work is presented.

2 Methodology:

In the following few sections, an overview of the methodology and the design of the associated threshold function used to evaluate whether a leak had indeed occurred or not is presented. In addition, the different autoencoder architectures attempted throughout are shown, as well as a description of what other hyperparameter changes were made throughout.

To do this project, PyTorch was used for the ML components, as well as other commonly used python packages used in an engineering setting (PyTorch, 2016). A simple schematic of the method is displayed in figure 2.

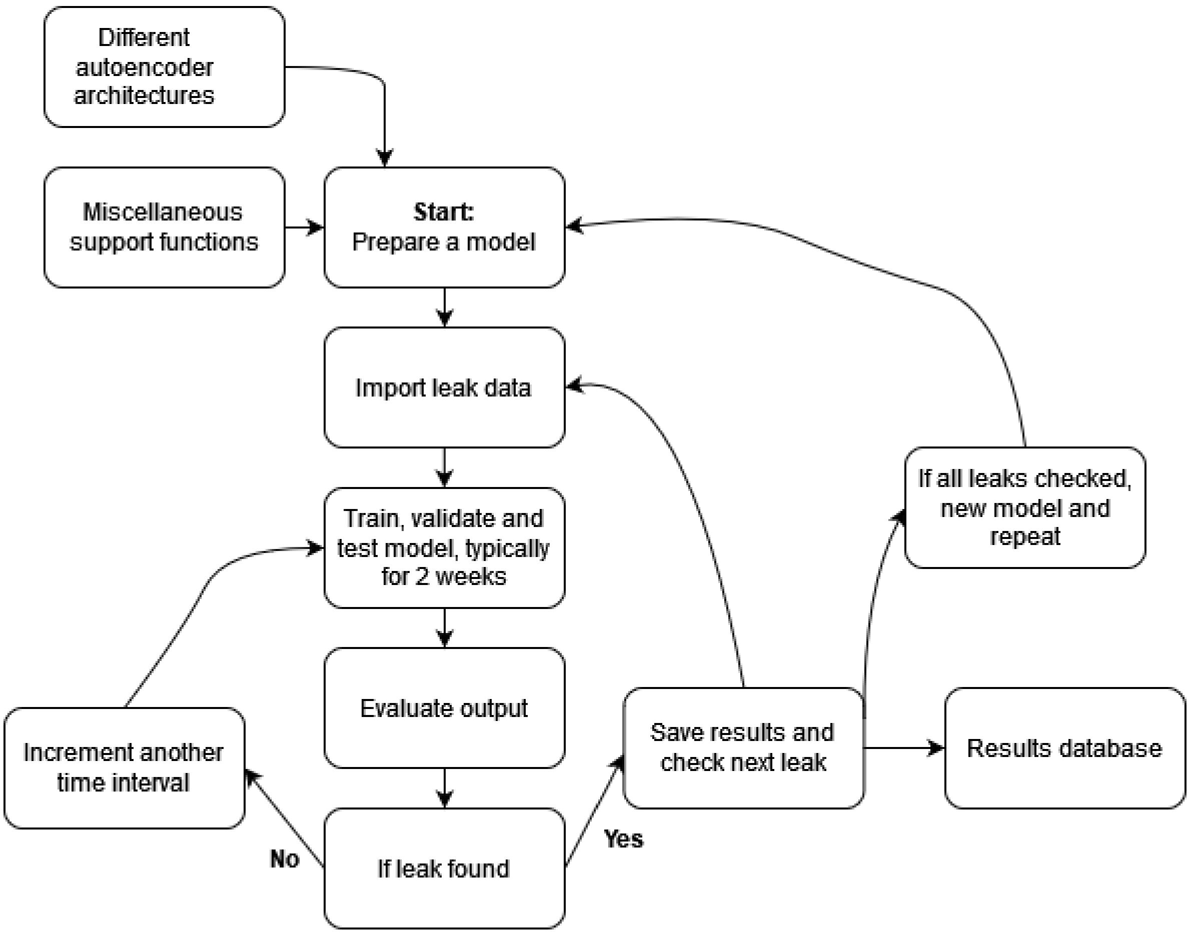
Different autoencoder architectures
Miscellaneous support functions
Start: Prepare a model
Import leak data
Train, validate and test model, typically for 2 weeks
Evaluate output
If leak found
No
Yes
Increment another time interval
Save results and check next leak
If all leaks checked, new model and repeat
Results database

Figure 2 Schematic of the script used in this project.

The general idea of the method is to iteratively feed the data through the model and evaluate whether a leak has occurred. If none is found, the time interval used for testing is added to the training-interval before the next testing-interval is evaluated. If a leak has occurred, it is stopped and the timestep in which a leak was detected is logged. Throughout development, the time interval for each of training, validation and testing was typically set to two weeks. As mentioned, occasionally changes were made, before running to see what influence a given tweak has had on the outcome. Examples of changes include the following:

- Adjusting and customizing time intervals for training, validation, and testing.

- Resampling the data to a larger timestep.
- Different autoencoder architectures.
- Different hyperparameter settings.
- Inspection of individual sensors’ reconstruction error plots.
- Finetuning of threshold function, to attempt to pinpoint leaks more accurately.
- Adjusting the number of sensors to be used in the autoencoder.

Furthermore, there is a need for a function to conclude based on the output from the autoencoder, whether a leak has occurred or not. This can be done by a simple visual inspection of the reconstruction plot, but it is necessary to create a function to automate this task.

How accurately one is able to detect the leak, meaning how far from it first occurring one first detects it, is largely dependent on the design of the threshold function in which one determines something abnormal has happened. If the criteria set for abnormality is too loose, one risks an increase in false positives, if it is too strict, one risks not the detecting the leak at all, despite it being apparent visually. Automating this aspect of the leak detection problem is necessary due to the fact it would allow a larger number of iterations with different autoencoder architectures and hyperparameter configurations to be tested without human interaction.

2.1 Threshold function

Ideally, you’d be able to detect the exact timestep in which a leak first occurs, but that would also entail a challenging design problem as to how to structure the threshold function. One attempt to do such a task has been done, but it is not unreasonable to think that different threshold function designs could potentially improve the accuracy of the leak detection done in this project.

The autoencoder has 36 neurons as input and output. One of the neurons include the tank level, one for the water demand seen in area A of L-Town, one for the pump and the remaining for pressure sensors. If the pressure sensors are in the same area of L-Town, they typically follow a somewhat similar pattern. Therefore, while all the neurons are included in the training of the autoencoder, only the pressure sensors are considered for the threshold-function. Furthermore, there was an insistence to make the threshold function rather robust as to not include unnecessary amounts of false positives if it was too loosely designed. Handling preliminary false positives and attempt to verify them before concluding whether a leak actually occurred or not, would needlessly add complexion to the task at hand. In addition, with 33 sensors, one would only need a few sensors to detect a leak at a given timestep within a certain timespan to indicate an occurrence of a leak, so there was some leeway in that regard as well.

Figure 3 shows a flowchart of how the threshold function was structured in this project. As water demand follows a specific pattern throughout the day, pressure values fluctuate accordingly, therefore the first thing that happens is that the average reconstruction error seen each hour of the day is kept track of and updated on a weekly basis. Then, a dummy variable is created that consists of the average value for a given hour, and some multiple of the standard deviation. This multiple will be referred to later on as the standard deviation multiplier and was typically in the range between 2 and 3. Each sample is then compared to this dummy variable in the preliminary check. If a simple is larger than the dummy variable, it will move to the second check, in which the next samples for a few hours ahead are investigated to see if they also exceed the reconstruction error seen at the same time of the day for a number of days prior. The purpose of this is to reduce the likelihood of letting through a fluke value the preliminary checks. Next, a check if the given sample also exceed the corresponding value for a few weeks prior. Seeing as false positives is a reoccurring danger when working with data-driven leak detection, the threshold function's perhaps excessive checks and comparisons was an attempt to reduce the number of false positives detected.

After each sensor's reconstruction error had been run through the threshold-function, the timestep in which each sensor would detect a leak would be available. These timesteps were then grouped together if they occurred within a day of each other, and it was assumed that the largest group contained the leak. If the group size exceeded or was equal to three (meaning three positive sensors were required), it was considered a valid group and a leak was detected. From there, the earliest sample of positive detection was chosen as the sample in which a leak occurred.

It should be noted and will be elaborated upon more at a later stage, that during the testing phase, the model is first trained and validated on about one to two months prior to the leak occurring, and then tested on the timeframe in which the leak is occurring, leaving little possible time in which false positives can occur. While working with this project, minor tweaks were made to various components of this overall process, but the general design has remained similar throughout.

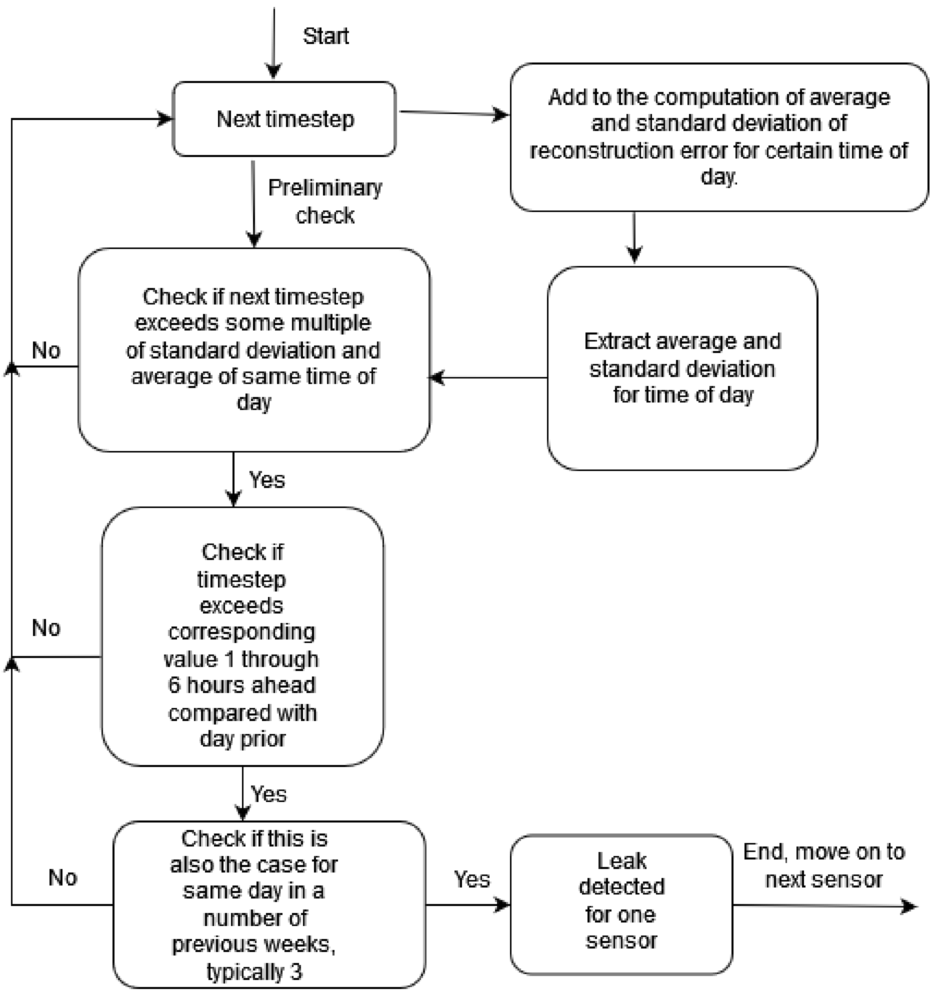
Start
Next timestep
Add to the computation of average and standard deviation of reconstruction error for certain time of day.
Preliminary check
Check if next timestep exceeds some multiple of standard deviation and average of same time of day
Extract average and standard deviation for time of day
No
Yes
Check if timestep exceeds corresponding value 1 through 6 hours ahead compared with day prior
No
Yes
Check if this is also the case for same day in a number of previous weeks, typically 3
No
Yes
Leak detected for one sensor
End, move on to next sensor

Figure 3 Flowchart for the threshold function used.

As mentioned, there are a number of different ways to design such a threshold function. In this project it functions roughly as seen in figure 3. Minor tweaks may be apparent in the final rendition of the script, which can be seen in appendix E.

Before a sample is run through the procedure displayed in figure 3, a rolling average is computed to reduce the degree the reconstruction error fluctuates. This rolling average is set to a step-size of 288 samples, which corresponds to one day with 5-minute samples. Attempts were initially made to reduce the step-size and finetune the threshold function to a larger degree to find the leaks more accurately, but it was later decided that the effort was spent more wisely working on different aspects of this overall project.

2.2 Autoencoder and hyperparameter tuning

ML algorithms such as an autoencoder, have hyperparameters and various components intended to make the algorithm more robust, less dependent on initialization and to increase reliability. Throughout conducting this thesis work, attempts have been made in order to find a more ideal configuration of these hyperparameters for the autoencoder, meaning one in which performance is found to be decent, without requiring excessive time spent during the training of the model. As the final configuration of the autoencoder will be presented later on, a brief introduction to the necessary background information is necessary.

In appendix A, ML terminology is presented. Furthermore, the purpose of the various hyperparameters is presented in appendix B. It is underlined that the following table only includes brief explanations, but the technicalities concerning the various ML concepts are beyond the scope of this thesis. The curious reader is advised to consult the aforementioned textbook or other learning resources if one wishes to see more elaborative explanations or a more general introduction to ML.

A number of different autoencoder architectures were prepared to shift through, to see if the number of layers, neurons in hidden layers or a variety of other factors had a noticeable difference in the performance. Figure 4 shows a simple illustration of an autoencoder frequently used for testing purposes during the development of this project. Technically the neurons for tank level, pump and area A's demand are also included in the training of the autoencoders, but not in the illustration.

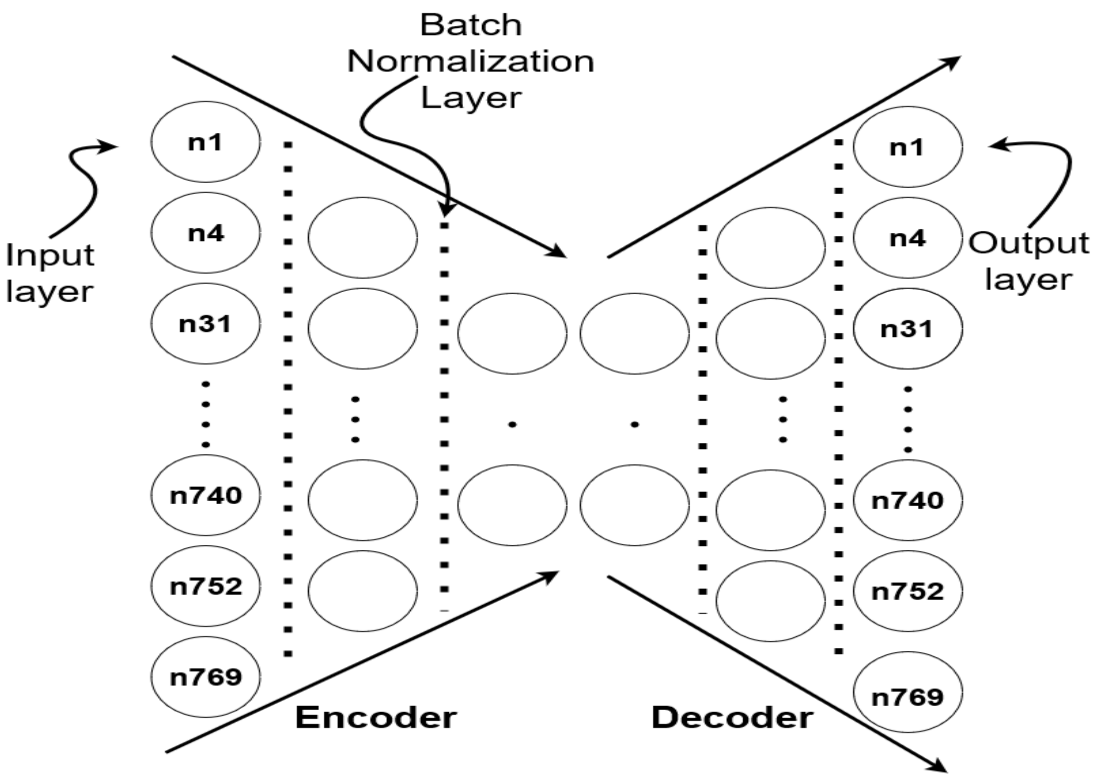

Figure 4 An illustration of a frequently used autoencoder during initial development.

When developing this project, and initially attempting to figure out a more optimal configuration of the autoencoder and hyperparameters, a shortcut was taken to increase the number of iterations of the script that could be done. The threshold function was ignored, and instead the mean square error across all the sensors was used, and the training was set for the first two weeks, and testing for the remainder of the year instead of iterating through, as previously explained. The assumption made here was that whatever configuration found to work best under these circumstances, would also perform best under the regular circumstances, when the threshold function is involved and a comparison on a per sensor basis is done. This change increased the number of configurations that could be tested drastically as there was no threshold function to evaluate the performance at this stage and there was only one training interval. The reconstruction error plots were saved and evaluated visually in the morning after running overnight.

The different autoencoder architectures consisted of 6 different alternatives, which can be seen in table 1. The autoencoders numbered between 7 and including 12 were used during the attempt at reducing the sensor count, which will be elaborated on later. Attempts with more than 5 layers showed no meaningful improvement.

Table 1 shows the different autoencoder architectures typically tested.

Autoencoder number	Layers and neurons per layer	Batch normalization	Drop out
1	$4\ layers;$ $[36, 24, 12, 6\ 6, 12, 24, 36]$	Yes, between every layer	No
2	$4\ layers;$ $[36, 20, 8, 4, 4, 8, 20, 36]$	Yes, between every layer	No
3	$4\ layers;$ $[36, 24, 12, 6, 6\ 12, 24, 36]$	Yes, but only between every other layer	No
4	$4\ layers;$ $[36, 24, 12, 6, 6, 12\ 24, 36]$	Yes, between every layer	Yes
5	$4\ layers;$ $[36, 24, 12, 6, 6, 12, 24, 36]$	None	No
6	$5\ layers;$ $[36, 26, 12, 8, 4, 4, 8, 12, 26, 36]$	Yes	No
7	$5\ layers;$ $[31, 20, 10, 6, 4, 4, 6, 10, 20, 31]$	Yes	No
8	$5\ layers;$ $[22, 12, 8, 6, 4, 4, 6, 8, 12, 22]$	Yes	No
9	$5\ layers;$ $[14, 8, 6, 4, 3\ ,3, 4, 6, 8, 14]$	Yes	No
10	$5\ layers;$ $[10, 6, 4, 3, 2, 2, 3, 4, 6\ ,10]$	Yes	No
11	$5\ layers;$ $[8, 4, 3, 3, 2, 2, 3, 3, 4, 8]$	Yes	No
12	$5\ layers;$ $[5, 4, 3, 3, 2, 2, 3, 3, 4, 5]$	Yes	No
13	$6\ layers:$ $[36, 26, 12, 8, 6, 4, 4, 6, 8, 12, 26, 36]$	Yes	No

The different optimizers, activation and loss functions tested in the aforementioned autoencoders are shown in table 2. Furthermore, the hyperparameters and their respective spans utilized during the large-scale iterations can be seen in table 3.

Table 2 Overview of different optimizers, loss and activation functions tested in this project.

Optimizers	Adam	AdamW	Nadam	adamax	SGD
Loss functions	Mean square error	Mean absolute error	-	-	-
Activation functions	Tanh	Relu	LeakyRelu	Sigmoid	-

As for learning rates, epochs, and batch sizes, they were randomly chosen among the values below each time the script iterates through the datasets with a particular autoencoder configuration.

Table 3 The various values tested for the different hyperparameters

Learning rates	[0.001, 0.0001, 0.00001, 0.005, 0.0005, 0.0025, 0.00025]
Epochs	[750, 1000, 1500, 2000, 2500, 3000, 3500, 4000]
Batch sizes	[500, 750, 1000, 1500, 2000, 2500, 3000, 3500]

Before deciding on the span of learning rates, epochs and batch sizes, a wider selection was tested, but the general ballpark displayed by the values in table 3 proved to be a consistently well-performing compromise between performance and necessary training time.

2.2 The datasets

The dataset being used is as mentioned, the dataset supplied from the organizers of the BattLeDIM competition. It is derived from an EPAnet model with the help of the python package WNTR. The EPAnet model is based on the hypothetical town called L-Town (BattLeDIM, 2020). An illustration of L-Town is shown in figure 5.

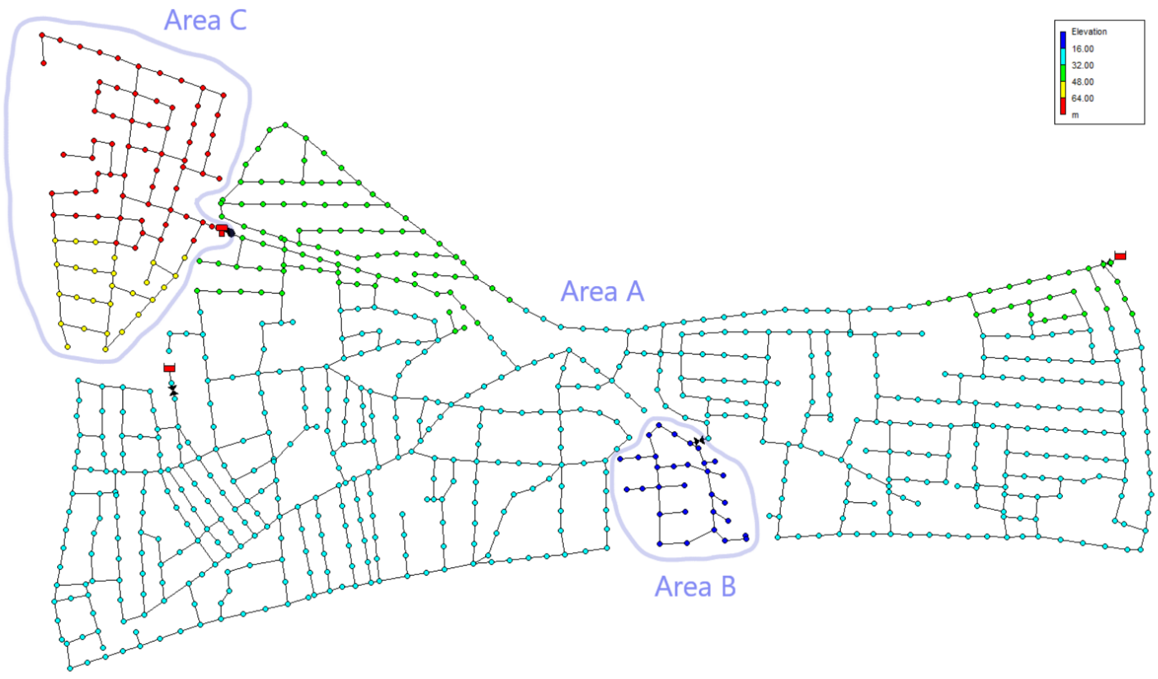

Figure 5 Illustration of L-Town.

L-Town consists of three different areas. Its population is about 10,000 people with a total length of pipes of about 42.6 kilometers. The network is supplied from two different reservoirs, and they can be seen as the small red squares in figure 5. Pressure reduction valves are installed downstream from the reservoirs to maintain a pressure head of at least 20 meters consistently for every consumer throughout the day. Another pressure reduction valve is placed between area A and B. It is installed with the goal of reducing background leakages. Between area A and area C, there is a pump and water tank installed, seeing as area C is at a higher elevation than the remaining network. The pump is set to be refilling during the night and emptying during the day to area C.

During weekdays the water consumption in L-town follows a regular pattern, but the weekend is more random due to increased consumption during nighttime to accommodate for the city's nightlife. There are no differences between holidays and regular days in terms of water consumption in L-Town. It is also presumed to be placed in northern Europe, meaning there is a higher water consumption in the summer months compared to the winter months.

To collect data from their network, the water utilities in L-town has installed the following types of sensors: 1 sensor for the tank's water level, 3 flow sensors and 33 pressure sensors. The values for the pressure sensors is the average on a per 5-minute basis.

The leaks found in L-town typically are of three different types, as can be seen in figure 6. The type that found in pipe 158 belongs to can be seen as a pipe bursting abruptly, which then floods a street and is quickly noticed and fixed by the water utilities. This is a type of leak that is typically easier to detect by an algorithm, but also would've likely been detected by the responsible water utilities quickly, since it may be visible at the surface level. Leak p461 sees a continuous increase in size, but due to its large size, it is found and repaired by traditional means. Leak p257 is a leak which grows gradually until it reaches its peak. It is substantially smaller in terms of volume per hour than the other leaks, but due to its small size it is presumed to be more difficult to detect, and in this dataset, this leak remains unfixed through the whole year. Incipient leaks also typically occur in pipes with smaller diameter. The challenge is then to detect these types of leaks, as quickly as possible.

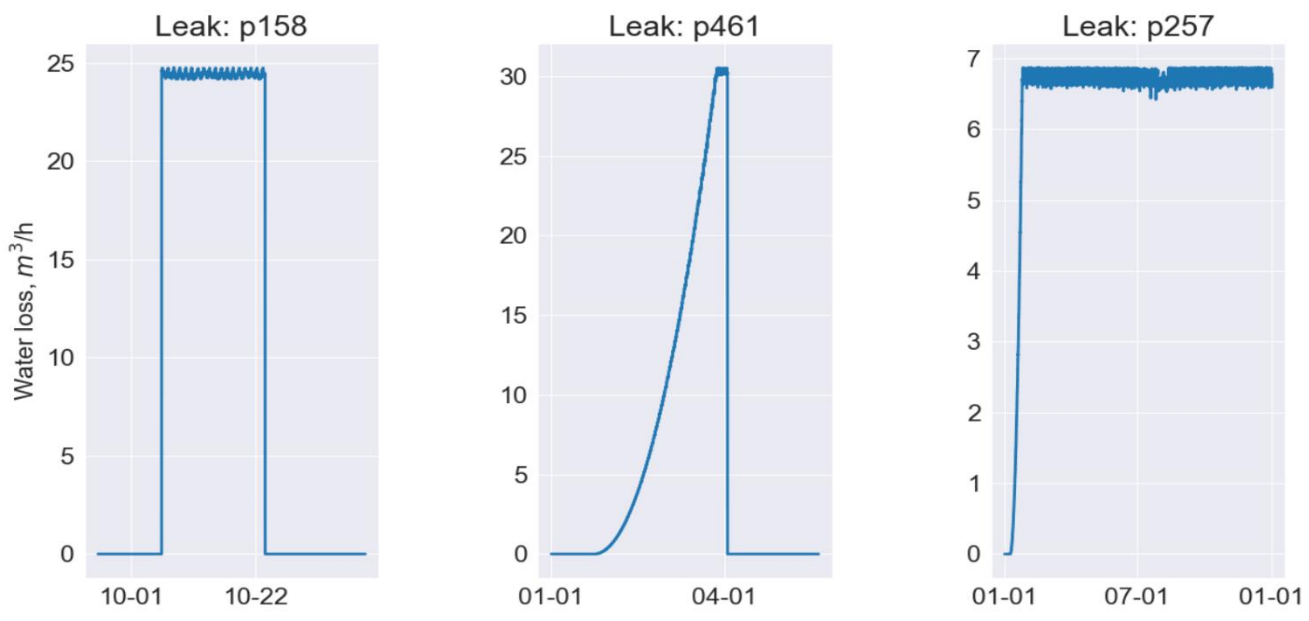

Figure 6 The different leak classifications: quick burst, slow increase, and incipient leak

Throughout this thesis, the leaks that follows the pattern seen in pipe 158 will be referred to as abrupt bursts or bursts, pipe 461 as a slowly increasing leak, and pipe 257 as an incipient leak.

In total, there are 14 leaks in L-Town occurring during 2018. There are 6 abrupt bursts, 4 slowly increasing ones, and 4 incipient ones scattered throughout the year. They're also spread across the city of L-Town, figure 7 shows the location of each of the pipes containing a leak. Figure 8 displays all the leak volumes and when they occur throughout the year.

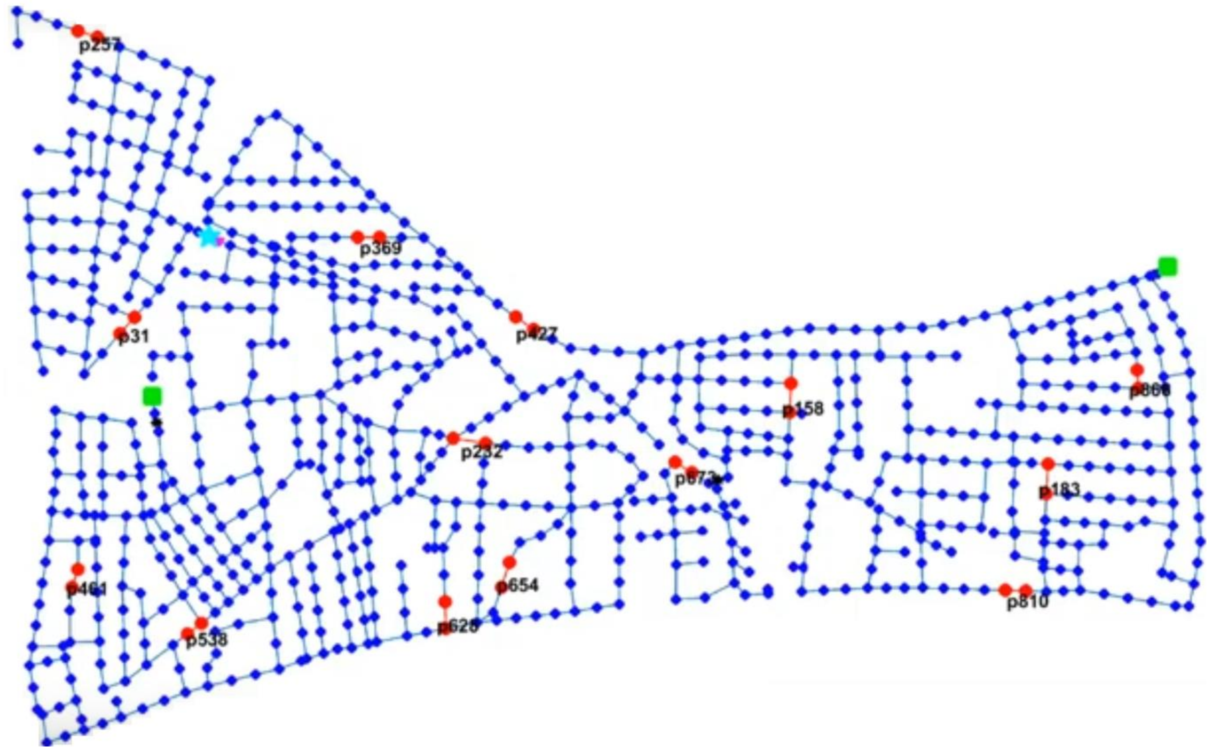

Figure 7 The location of pipes with leaks found in L-Town

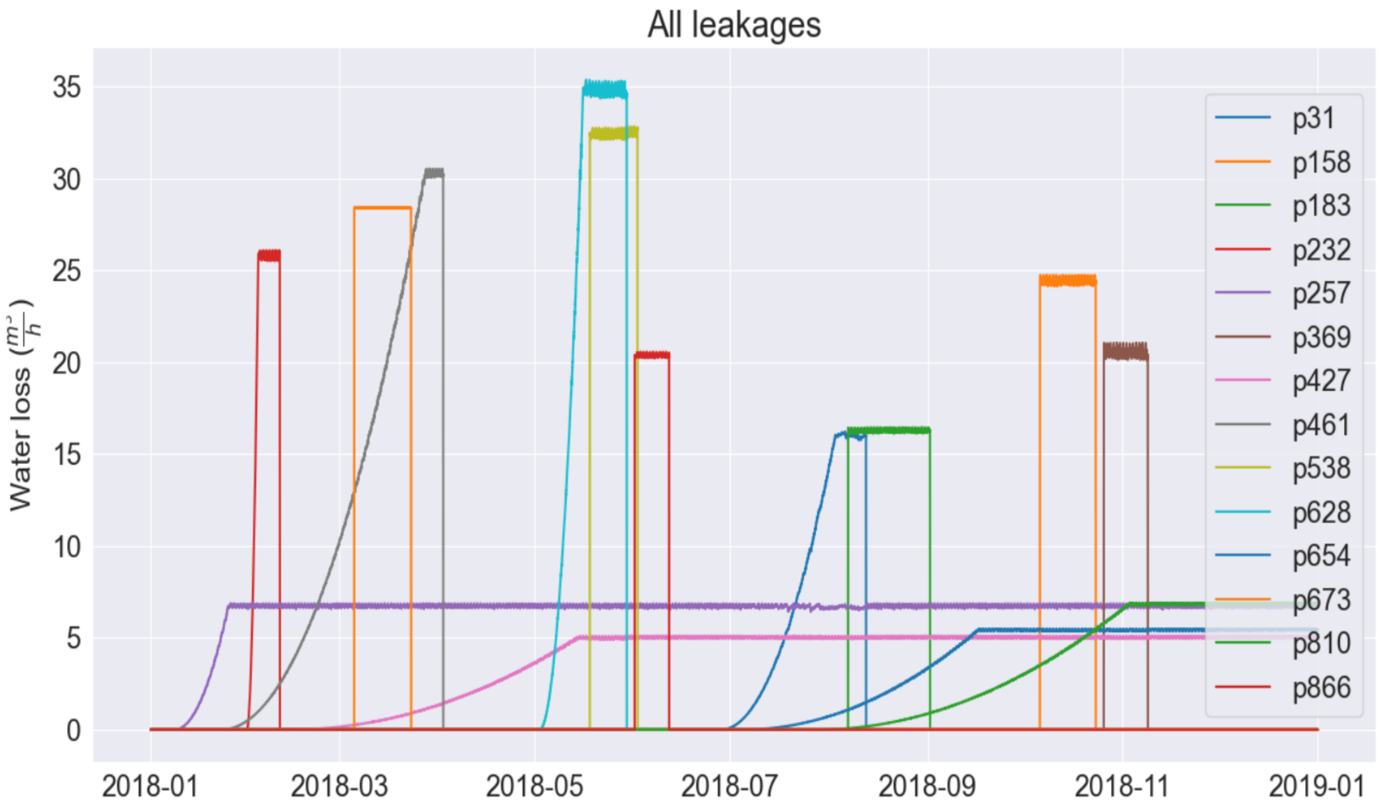

Figure 8 Shows all the different leaks in L-Town for the year 2018.

In L-town there are 33 pressure sensors. Their location is shown in figure 9. One pressure sensor will function as one neuron in the autoencoder. In addition, the demand for zone A, and the tank level, and the pump flow will also be used as neurons in the autoencoder. In total there will then be 36 features with 5-minute samples throughout the year of 2018, meaning there will be 105 120 samples in total.

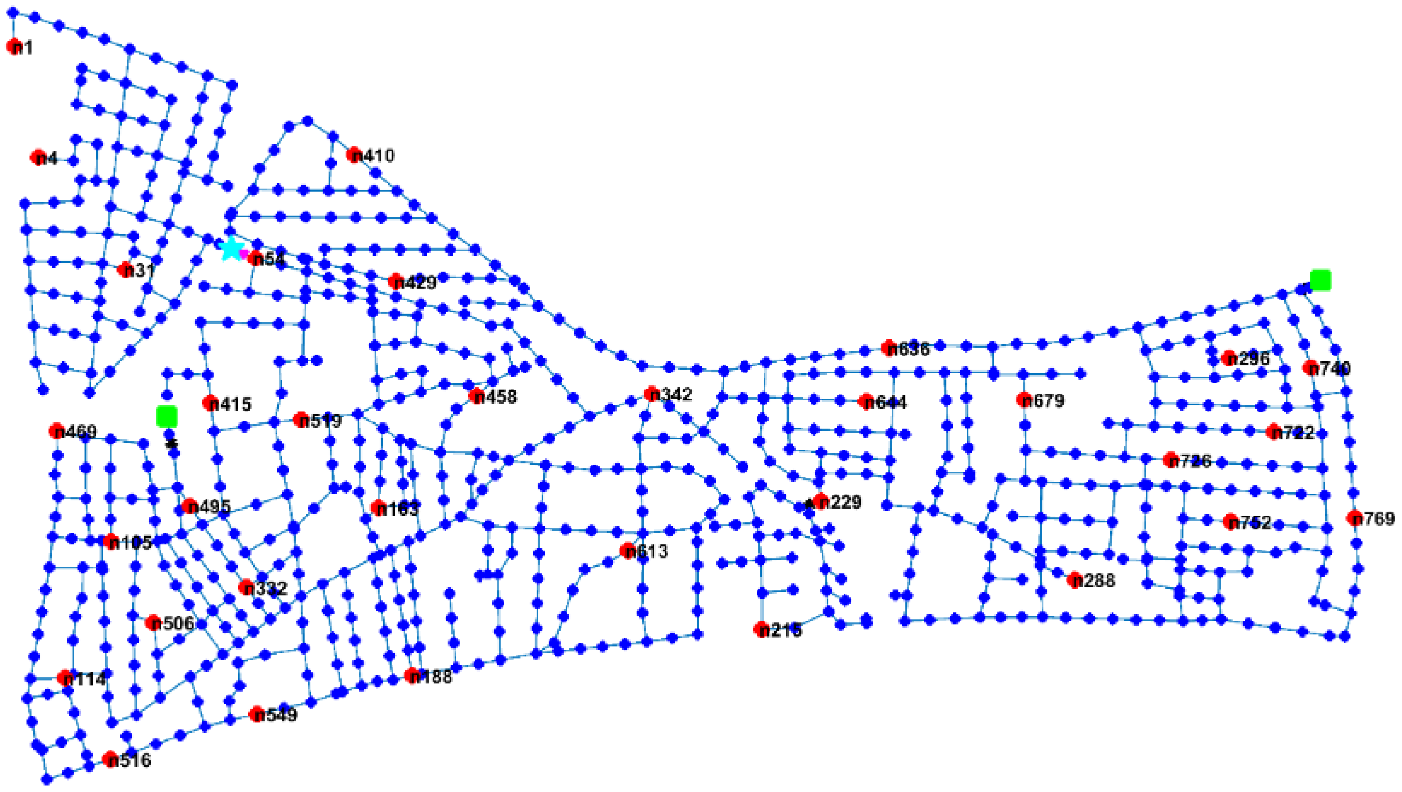

Figure 9 Red dots indicate pressure sensors in L-Town. Green dots are reservoirs, and the teal dot is a pump.

3 Results and discussion

The results can be seen in table 4. The best performance across autoencoder architecture and hyperparameter configuration is deduced based on the accumulation of time difference across datasets, assuming all the bursts and slow increase type of leaks were found. Seeing as the quick bursts were largely found faster compared to the slow increase type of leak, whichever configuration that managed to adequately detect the quick bursts while also excelling at the slow increase were more likely to be deemed the winner. This might not be an optimal metric to pick best performance by, as in a real system the consequences of having a burst unattended for some period of time is perhaps higher than having a slow increase leak grow slightly larger without sounding the alarm immediately.

One alternative could be to investigate the net volume of water lost through the leakages and rank the different autoencoder architectures and hyperparameter configurations by that but developing such a metric has not been done in this project.

Table 4 Results

Leak	Timestep leak detected	Timestep leak occurs	Timestep difference	Classification	Leak flow volume at detection
Pipe 31	56,892	51,573	5,319	Slow increase	4.48 $\frac{m^3}{h}$
Pipe 158	80,315	80,097	218	burst	24.31 $\frac{m^3}{h}$
Pipe 183	62,964	62,817	147	burst	16.21 $\frac{m^3}{h}$
Pipe 232	10,048	8,687	1,361	Slow increase	25.64 $\frac{m^3}{h}$
Pipe 257	Not found	Not found	Not found	Incipient	Not found
Pipe 369	86,039	85,851	188	burst	20.32 $\frac{m^3}{h}$
Pipe 427	Not found	Not found	Not found	Incipient	Not found
Pipe 461	13,248	6,625	6,623	Slow increase	4.28 $\frac{m^3}{h}$
Pipe 538	39,662	39,561	101	burst	32.22$\frac{m^3}{h}$
Pipe 628	36,492	35,076	1,416	Slow increase	4.82 $\frac{m^3}{h}$
Pipe 654	Not found	Not found	Not found	Incipient	Not found
Pipe 673	18,341	18,335	6	burst	28.4 $\frac{m^3}{h}$
Pipe 810	Not found	Not found	Not found	Incipient	Not found
Pipe 866	43,638	43,600	38	burst	20.35 $\frac{m^3}{h}$

As can be seen in table 4, finding incipient leaks has proven itself to be a challenge. They typically grow too slowly for the autoencoder to be able to create a reconstruction error. This will be elaborated upon more at a later stage. However, the pipe bursts are all found within less than a day. The slow increase leaks are typically found when they reach around 4.5 $\frac{m^3}{h}$, the exception being pipe 232, which is a leak that grows very rapidly in the beginning of the year, making it more challenging to detect. In addition, due to its rapid, but still gradual growth, which can be seen in figure 8, whether to classify it as a slow increase or a burst is debatable.

Furthermore, the locations of the leaks in the system have an impact in how reliably they're detected. Leaks found in area C and B are typically more challenging to detect. This is due to the fact that there is somewhat of a hydraulic disconnect when a pump or pressure reducing valve is involved in connecting the different areas together. The leak in pipe 673 is particularly sensitive and has been a source of consistent false positives throughout working with this project. It is found in area B and a closer description of this problem will be presented at a later stage.

When visually inspecting the reconstruction plots to investigate whether the threshold function did indeed sound the alarm at the time in which the leaks start to appear, it was found that there is small (a few hours perhaps) room for improvement with some of the burst leaks, but not all of them. However, loosening up the threshold function in its current state runs the risk of running into false positives at a larger rate. Perhaps a different threshold function design could allow for both.

Although, the purpose of this project is not to compare with the results from the BattLeDIM competition, as the circumstances differ between the two settings. In the BattLeDIM competition, the information about the leaks were not made available, whereas they have been throughout the development in this project. Nevertheless, an interesting quirk is that one of the winning teams managed to detect leaks between 4.8 $\frac{m^3}{h}$ and 35 $\frac{m^3}{h}$, meaning the autoencoder performed more or less just as well in that regard, due to the detection of the slow increase leaks. In the original BattLeDIM competition, the problem statement, and resources available were both larger than in this thesis, so the results aren't directly comparable to this project. Primarily the pressure values from the sensors were relied on in this project, and the EPAnet-file of L-Town that was made available during the BattLeDIM competition was not used at all in this project.

The top performing hyper parameters and autoencoder configuration found is the following seen in table 5. This configuration of the autoencoder will henceforth be referred to as standard settings and will be used in the next subsections unless otherwise stated. As mentioned, the standard deviation multiplier was set between 2.5 and 3 generally, for this final iteration it was set to 2.75. It was trained, validated, and tested with two weeks increments.

Table 5 The autoencoder architecture and hyperparameters associated with the best performing configuration.

Autoencoder architecture	Autoencoder 6
Loss function	Mean square error
Activation function	Tanh
Optimizer	Adam
Learning rate	0.025
Epochs	2500
Batch size	750
Training interval	14 days
Validation interval	14 days
Testing interval	14 days

The autoencoder reached a training loss somewhere between 0.08 and 0.2 depending on which leak dataset it was trained on. Validation error around between 0.15 and 0.3. This is after normalizing the input data.

When training the autoencoder with settings different from the ones seen in table 8, but a randomized selection of the ones seen in table 1,2 and 3, the results did not change drastically, and only minor changes were noticed, suggesting that a larger span of possible configurations would see a similar performance. Adding another layer to the autoencoder, or drastically increasing the batch size or epoch number did not create any meaningful difference. It is believed that the aforementioned configuration is a decent compromise between performance and the amount of time necessary for training.

3.1 Results for different leak classifications

To showcase visually what the different leak classifications look like when detected by the autoencoder the following plots are presented. The different classifications of leaks typically have similar looking plots, therefore one of each will be shown. The plots display the leaks from the start, until the end, when the L-Town municipality fixed them. It is used a shifted 2-week window for training, validation, and testing of the autoencoder for all the following plots, with standard settings.

As some time has been spent testing various configurations to attempt to find specific leaks, a few tables are presented with a short comment on the features associated with each leak. Some are more challenging to detect than others, and some are more prone to false positives and so on. One example being leaks found at the tail-end of summer, as the reconstruction error is compared with the larger reconstruction errors typically seen throughout summer due to increase in pressure fluctuations as the demand changes.

3.1.1 Quick burst

In figure 10, four sensors are compared to see their respective reconstruction errors. Three of the sensors chosen are the ones closest to the leak, as the size of the reconstruction error is seen to be correlated to the proximity of the pipe with the leak. The remaining sensor is the one furthest away, to show that it is indeed less affected by the pipe bursting. Although, it should be noted that it is important to compare with the sensors that are within the same area of L-town. The pump to area C function as disconnection point hydraulically and the sensors found in sector C therefore do not function as well as a baseline, despite being further away in terms of distance. For figure 10, the leak in pipe 183 peaks at $16\frac{m^3}{h}$.

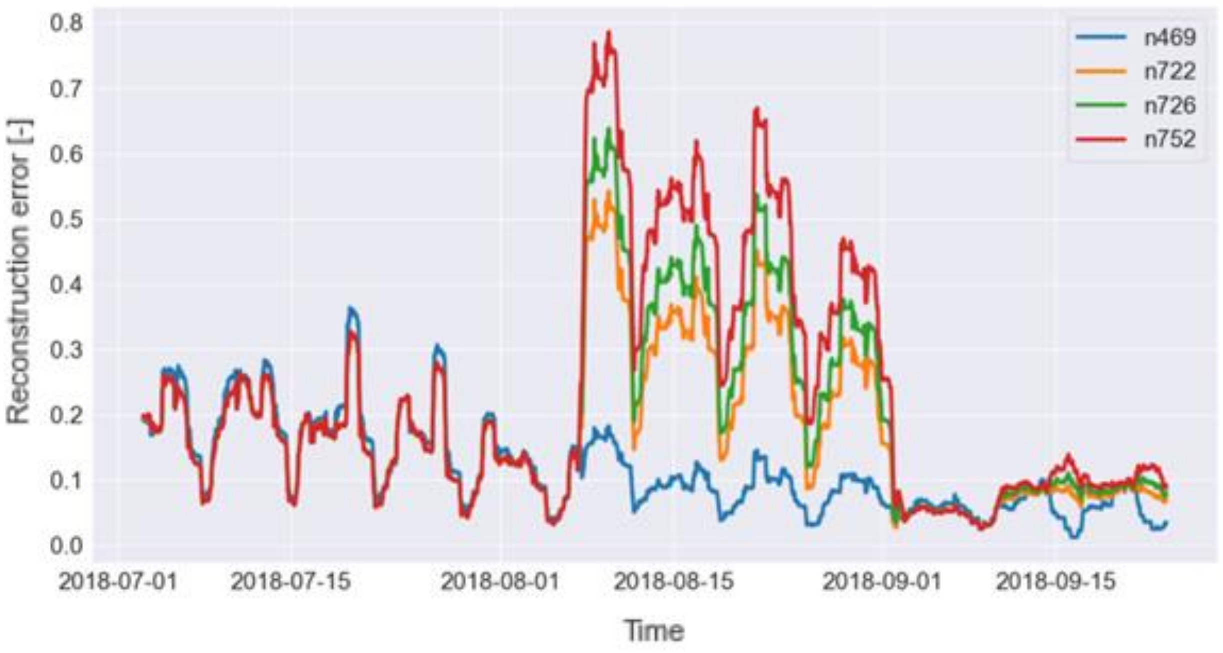

Figure 10 Rolling mean on a per day basis for the reconstruction error from sensors n469, n722, n726 and n752 with the leak in pipe 183.

Table 6 has a brief description of the quick burst leaks found in L-Town.

Table 6 Overview of quick burst leaks

Pipe Number	Area of L-Town	Comment
158	A	This leak is fairly easy to detect without any issues.
183	A	This leak is typically fairly easy to detect, without any major issues. Occasionally a false positive due to the fact that it occurs in late summer and the reconstruction error typically increases due to higher water demand in summer, and larger pressure fluctuations leading to larger reconstruction error fluctuations.
369	A	This leak is fairly easy to detect without any issues.
538	A	This leak is fairly easy to detect without any major issues. Occasionally a false positive, but not very frequently.
673	B	This leak is an outlier. Generally, roughly every sensor detects this leak, and it is not uncommon that false positives occur. The reconstruction error seen is typically abnormally large, especially for sensors also found in area B. Figure 18 shows the detection of a leak in pipe 673 and a larger elaboration as to why the detection of this leak is more unreliable than others is included.
866	A	This leak is fairly easy to detect without any issues.

3.1.2 Slow increase

On figure 11 a leak of type of the slow increase classification is shown. It can be seen that the reconstruction error is increasing more slowly compared to the quick burst variety. Furthermore, it can be seen to be a bit noisier, and likely slightly more challenging to trigger a threshold-function reliably at an early stage of leak development. For the record, in a generic run of the model, the leak in p461 were detected when it reached a flow of 4.28 $\frac{m^3}{h}$ at the date 15.02.2018, and the leak started on 24.01.2018.

As with the previous plot, the sensors shown are the ones in closest proximity and one further away to display the baseline reconstruction error of the system in comparison. For figure 11, the leak in pipe 461 peaked at 30 $\frac{m^3}{h}$. Typically, it is seen that larger leaks create larger reconstruction errors as one might expect. Table 7 has a brief description of the different leaks of the slow increase classification.

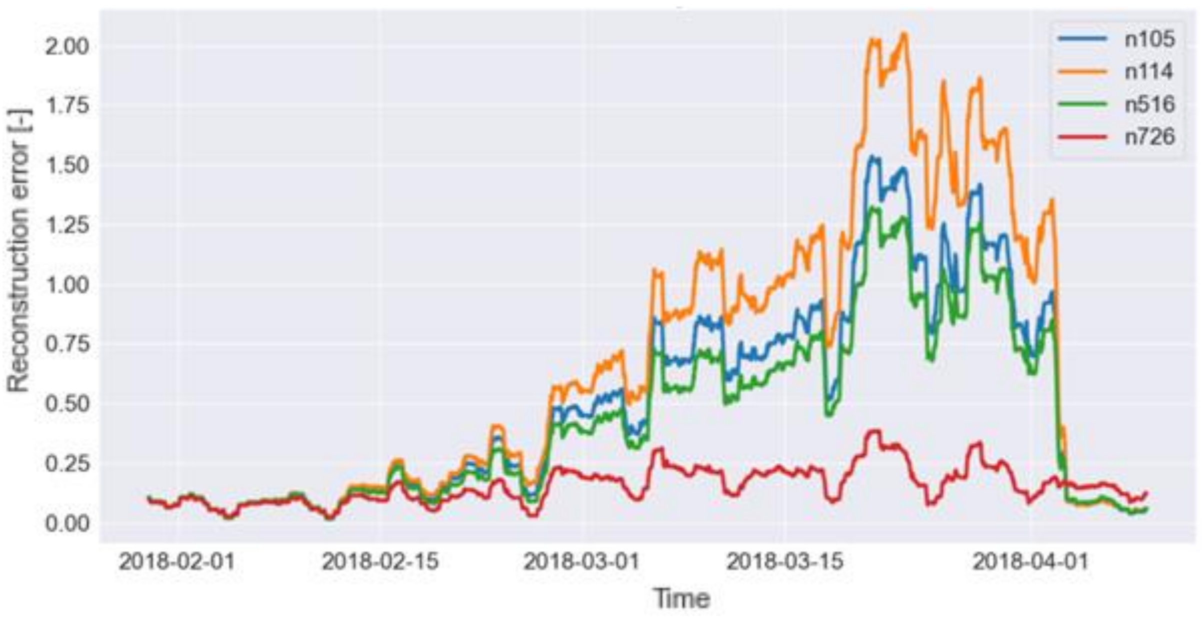

Figure 11 Rolling mean on a per day basis for the reconstruction error from sensors n105, n114, n616 and n726 with the leak in pipe 461.

Table 7 Overview of slow increase leaks

Pipe Number	Area of L-Town	Comment
232	A	This leak occurs early in the year, making it challenging to detect reliably with a 2-week training interval which was the norm for the majority of this project. It is therefore often only detected when it reaches its peak volume. It also grows significantly faster than the other slow increase leaks, as seen in figure 8. As can be seen in table 9, a shorter time interval outperforms the final results seen in table 4.
461	A	This leak is occasionally not detected at all, likely due to the fact that it occurs so early in the year, similar to the p232. This makes it more challenging to pick up on with a two-week training interval. If training data for the year prior existed, both of these leaks would likely not pose the same challenge.
628	A	This leak is typically found without any issues.
31	C	This leak is typically found without any issues, as there exists 3 sensors in area C, and the threshold function requires 3 positive sensors within a certain timespan. It can also turn sensors outside of area C positive, but typically with a delay compared to ones found within area C.

3.1.3 Incipient

As for the incipient type of leak, the autoencoder is unable to detect the occurrence presumably due to its low volume and the slow growth of the leak. When the leak's growth is occurring over multiple training and test cycles, the leak loses its prominence among the other datapoints and fails to create a sufficiently large reconstruction error for it to be detected visually in the plot or by the threshold function.

There are a number of data processing methods that can be utilized in an attempt to pinpoint the leak, such as looking at middle night flow (MNF) only, removing seasonality, noise, or a rolling 25th- percentile minimum or maximum window, subtracting a leak-free year's reconstruction error (should such a thing be available or estimated) or something else entirely. These implementations may prove to be somewhat more successful than what is shown here, and the problem will be discussed further at a later stage. Despite of attempting a number of different ways to make the incipient leak more prominent, they have all been largely unsuccessful.

In figure 12 can be seen that the leak is not apparent in the reconstruction error at the start, and still not later on as it reaches its peak flow of 6.9 $\frac{m^3}{h}$. It is not known what exactly causes the divergence between the sensors at the end of the year. One possible explanation is that water demand and subsequent pressure fluctuations typically decline in the wintertime for cities placed on the northern hemisphere, and one would therefore expect the magnitude of the reconstruction error to decline as well or stabilize (as can be seen in the beginning of the plot, at the tail-end of the summer), as it does with n469, which is farthest away. However, the closest sensors to the leak do not follow that pattern. This is a type of reconstruction error that can perhaps be detected by a threshold-function, as such the leak is detectable, but nowhere near in time of which when the leak first occurs. However, it should be pointed out that the incipient type of leaks were never found by the L-Town municipalities, and as it is with most things, it is always better to find your leaks late than never. In addition, due to the lower flow from this leak, the necessity to find it quickly is not as essential as for the larger flow quick burst leaks. Still, the confidence in figure 12 is not complete and could be a coincidence, as the similar pattern is not seen with the different incipient leaks, at least not as prominently.

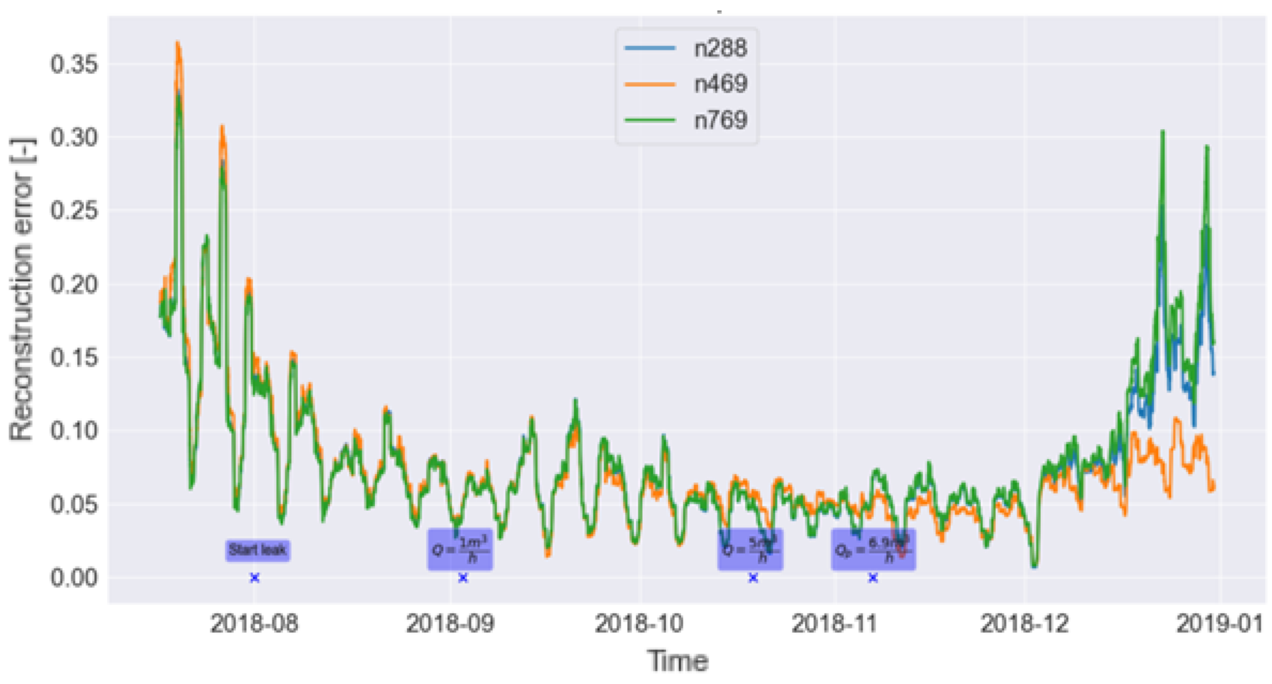

Figure 12 A comparison between sensors with a rolling mean on a per day basis for an incipient leak in pipe 810.

Table 8 has a brief description of the incipient leaks found in L-Town.

Table 8 Overview of incipient leaks

Pipe Number	Area of L-Town	Comment
257	C	Can sometimes be detected but may also simply be a false positive as no apparent leak is seen when visually inspecting the reconstruction plot. The leak occurs very early in the year, so there is loads of time available to detect false positives, meaning there is low confidence in the reliability of the results the few times it is detected. Occurs early in the year, which may make detection more challenging.
427	A	Not detected. Occurs early in the year, which may make detection more challenging.
654	A	Not detected, although interesting pattern seen, will be elaborated more at a later stage.
810	A	Not detected, although interesting pattern seen, will be elaborated more at a later stage.

3.2 Comparison between an accumulated- and shifted training phase.

One possible distinction in the use of the autoencoder is whether to choose to accumulate the training phase or have it shifted along through the year. Accumulated training phase means that the data that the autoencoder processed without a leak detection is added to the training data for the next iteration, meaning the training period continuously increases. This follows the ML mantra of “the more training data, the better the outcome”. The drawback is that it requires more computational power to train the model on a larger amount of data. The alternative being the shifted training phase, in which one continuously trains on a say, 2-week basis, for a couple of iterations, then test the autoencoder, which would not be as computationally demanding, and perhaps reduce the increased error due to the higher summer water demand and subsequent pressure fluctuations which could potentially overshadow a leak. No matter the approach, the detection of the incipient leaks could not be done, and as such, figures of an incipient leaks are not included among the following figures.

To illustrate how large of a difference these two types of approaches make, the following plots were produced. They're based on the same standard autoencoder architecture with the same hyper parameter configuration. The only difference is the accumulated and shifted training periods. Standard settings were used for all attempts.

As for sample-count when training, there is about 4000 samples of 5-minute timesteps per 2 weeks, meaning that the shifted training autoencoder trains on the 4000 samples. The number of samples for the accumulated training autoencoder is roughly 4000 per 2 weeks, so if the autoencoder is trained until it reaches a leak in late summer, as it is in figure 13 for the quick burst instance, then it will be trained on about $4{,}000 \cdot 14 = 56{,}000$ samples, as it starts around week 28.

3.2.1 For quick burst:

In figure 13, a comparison of between accumulated and shifted training period is done. It seen that the two alternatives follow each other closely, but the accumulated training shown in red appears to be slightly more stable during the period in which the leak is active. Although, it should be noted that the difference is marginal. Furthermore, the initial spike appears to be quite similar which is what the threshold-function is aiming to detect, meaning it is suspected to not be a noticeable difference between the two for the quick burst type of leak.

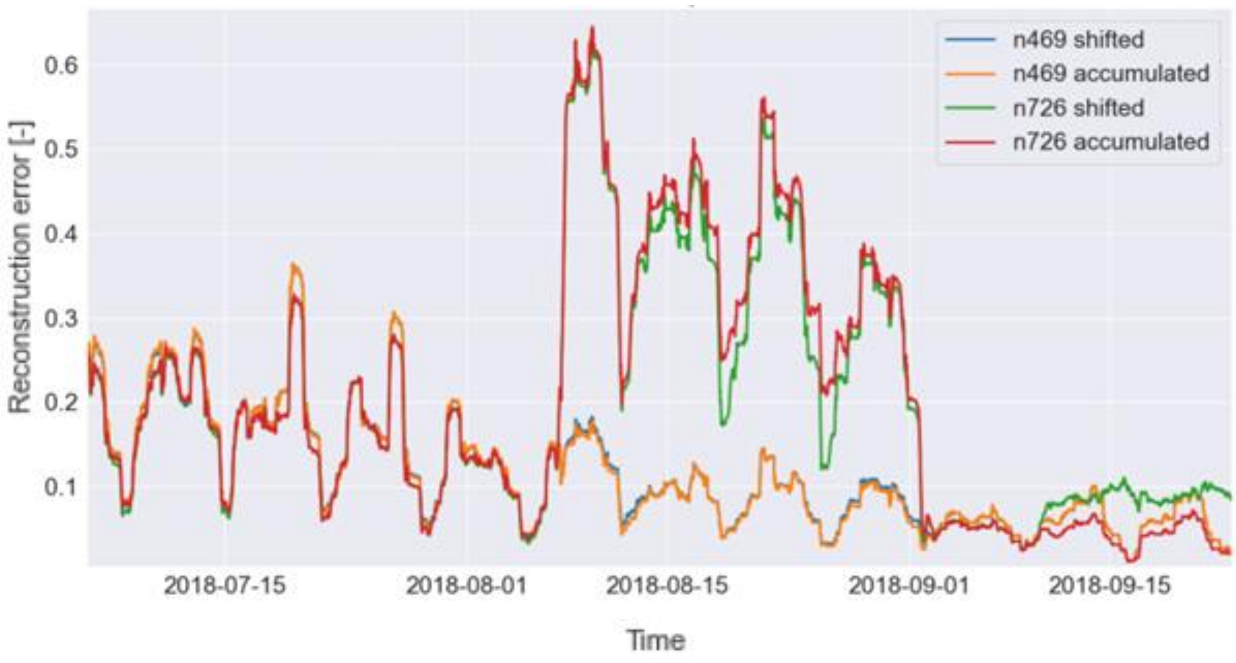

Figure 13 Accumulated and shifted training approach comparison for a burst leak in pipe 183.

3.2.2 For slow increase:

As with the prior plot, an accumulated training phase didn't create a meaningful difference between the two options. In figure 14 It is seen that the peak of the accumulated plot is more apparent, but the shifted starts diverging from the baseline n769 sensor at a slightly earlier stage. This divergence is the more influential part of the plot, as that's what would trigger whatever type of threshold algorithm is implemented in a system.

As for incipient leaks, a lack of difference between the two approaches were found and there is no obvious leak, as such a figure is not included.

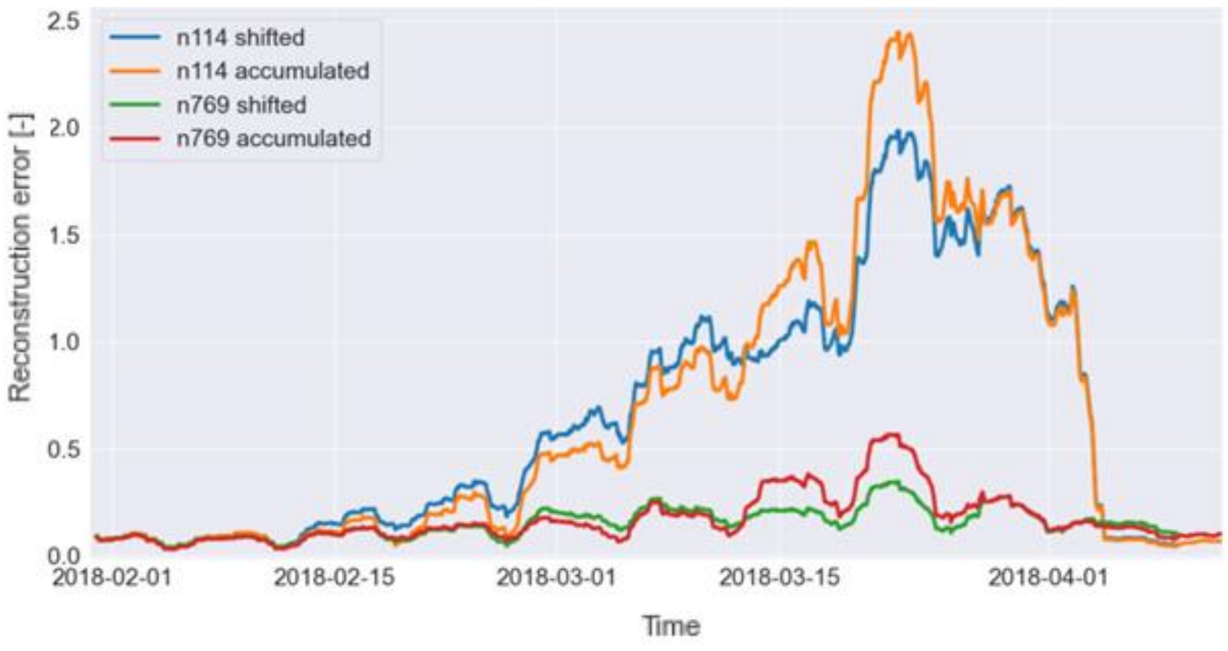

Figure 14 Comparison between shifted and accumulated training for a slow increase pipe leak in pipe 461.

The impact of this is that it could potentially reduce computational demand with a simple shifted training approach instead, as the results do not improve in a very meaningful way, despite the significantly longer training time needed. However, it is important to underline that this is a very simple preliminary test, and as such, one should be wary in justifying any choice made in an operational real-time leak detection system based on this alone. In such a case, more research would be needed.

3.3 Leak localization

The autoencoder's ability of being able to create a reconstruction error on a per sensor basis allows it to do some rudimentary leak localization analysis as well. In the literature regarding leaks in WDNs, leak localization and leak detection are separate problems, and as this project's focus is on the leak detection, it largely remains outside the scope of this project. Nevertheless, it is an interesting feature of the autoencoder, so it will be mentioned briefly. Figure 15 shows the different reconstruction errors for all the sensors for the leak in pipe 866, and most follow a similar trend before the leak occurs. The few that don't belong to the sensors found in area B and C

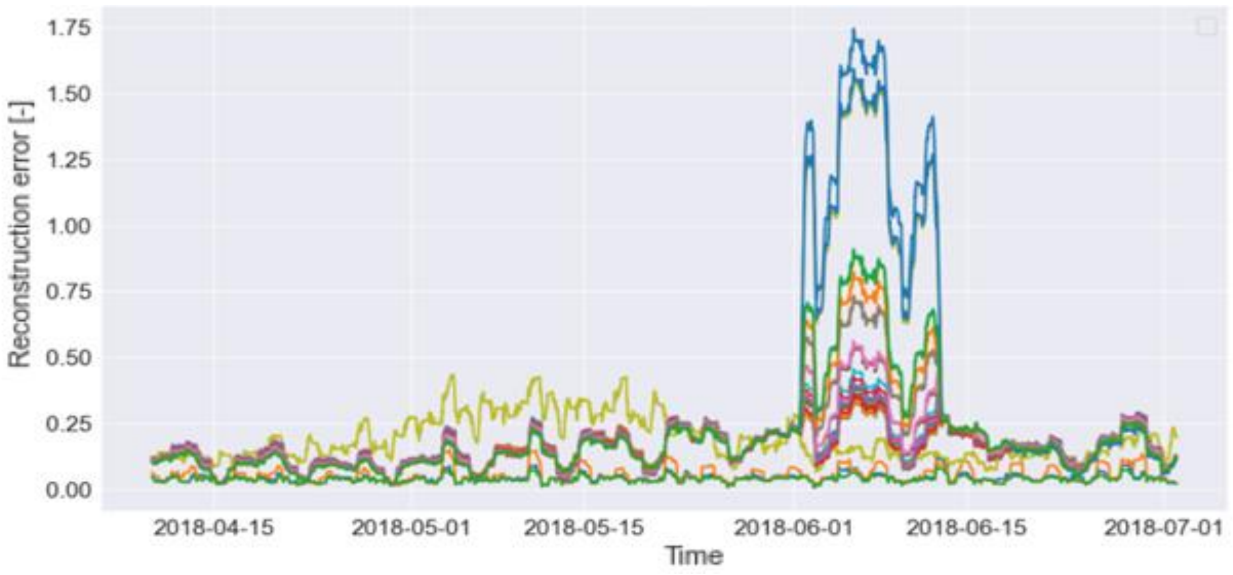

Figure 15 Shows all the sensors' reconstruction errors together on a rolling mean per day basis for leak in pipe 866.

In figure 16, it is seen that the sensors n286 and n740, which are the ones closest to pipe 866 has the largest reconstruction errors. As for the aforementioned plots, the sensor n469 is included for comparison and is the one furthest away from the pipe 866 leak and does not show any noticeable deviation from the norm when the leak first occurs. The aforementioned figure 7 and figure 9 show the pipe and sensor locations in L-Town respectively.

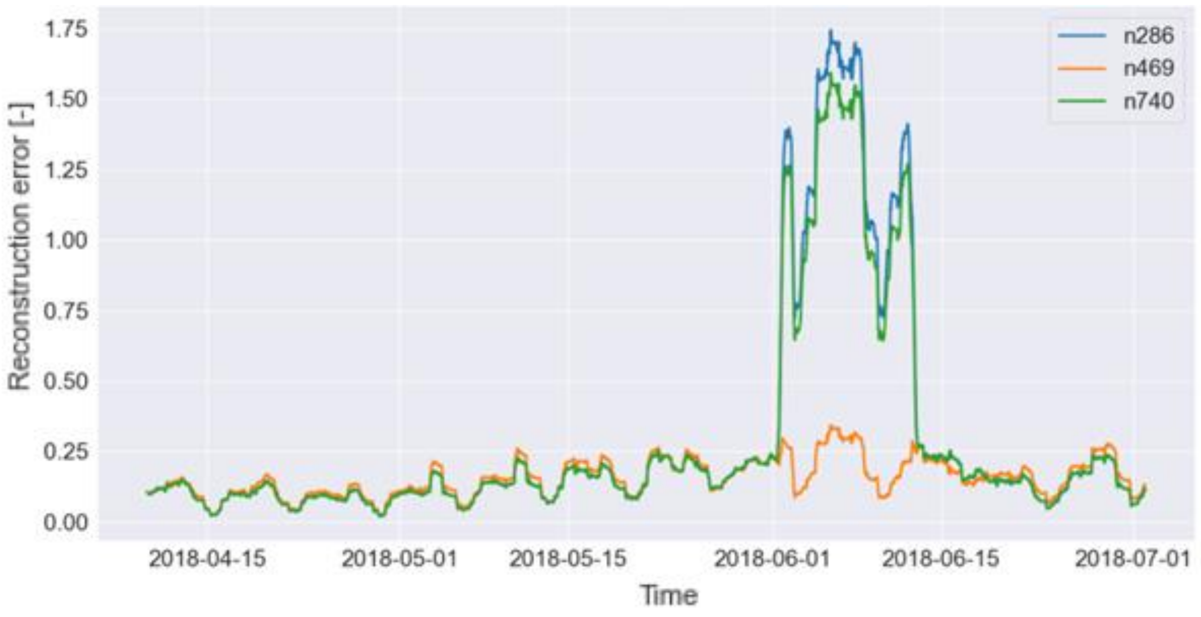

Figure 16 Rolling mean on a per day basis for the reconstruction error from sensors n286, n469, and n740 with the leak in pipe 866.

This rudimentary leak localization is shown to illustrate that if a WDN were to implement a real system, the localization benefits would be increasingly more accurate as the number of sensors in the system increased. This would presumably also reduce the manual labor cost and time spent searching associated with any subsequent in-person leak localization to a larger extent. It could perhaps also function as a starting point for some more advanced leak localization methodology.

3.4 Influence of training interval and resampling on the performance of the autoencoder

Due to the fact that it was not known how much of an influence the size of the time interval had on the training of the autoencoder, a few attempts were made in which this was adjusted. The first choice of the two-week time interval was largely chosen at random, so there was an interest in seeing whether this adjustment would make a meaningful difference. The shorter the time interval, the fewer samples the autoencoder will be trained on during the training stage. This may pose a problem on the quality of the training. The incipient leaks were excluded from this test due to the fact it was presumed they'd be too difficult to detect and including them would increase script's running time significantly.

Similarly, increasing the resampling of the from 5 minutes, which it has been originally, to 10, 30 or 1 hour, will also decrease the samples for the autoencoder to train on, assuming the overall time interval is kept the same. Ideally, one would be able to reach the same quality of training with a high resampling as the sensor's battery life is increased when the data is transmitted more infrequently. However, the old ML mantra of "the more training data the better" still holds true, so the question remains of short can you reduce the time interval for training or how high can you resample the data, and still be able to make reconstruction errors detectable for the threshold function? Is there a limit of how many samples the model needs until it is not able to create prominent reconstruction errors when leaks do occur?

For this test, the autoencoder architecture and hyper parameter settings were kept the same throughout, with the standard settings, only the time interval for training and resampling of the data was changed. All 33 sensors were being used, as well as the tank level, demand in area A, and pump flows, similar to previous tests. As in most other tests, the batch size was set to 750 and the epochs to 2500.

Initially, with two-week training intervals and timestamps on a 5-minute interval, the model is fed $12 \cdot 24 \cdot 14 = 4032$ samples per training cycle. Then another 4032 samples for validation and 4032 samples for testing. When the training time is halved or the timesteps doubled with a per 10 minutes resampling, the number of samples for the autoencoder to process is also halved. For both of these changes it is seen that the model's training and validation losses stabilizes at a higher level compared the original configuration with standard settings. This implies that the baseline reconstruction error will be higher with more variance and subsequently a messier plot in which the occurrence of a leak will be less prominent.

It was hoped that due to the fact that say, for a 5-day training interval, the autoencoder would go through almost 3 training cycles, compared to 1 training cycle for the 14-day interval autoencoder, that the extra training cycles would compensate for the initially higher training loss seen. Unfortunately, that is not the case and despite several more training cycles, the training and validation losses stabilize at a higher value compared to the original configuration. As one would expect, a similar trend is seen when the resampling is doubled from the standard 5-minute timesteps to 10 minutes.

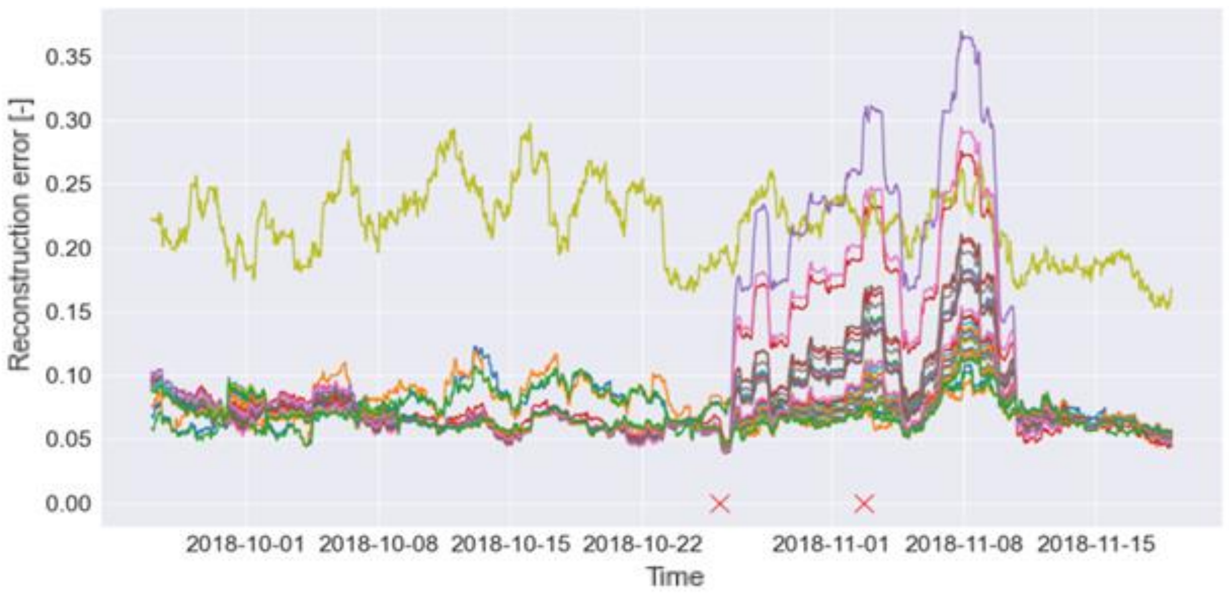

Figure 17 Detected leak on a two-week training interval with 10-minute resampling in pipe 369.

Figure 17 shows an example of leak detected with a two-week training interval, but a resampling to 10 minutes samples. The first red x is when the leak occurs, and the second is when it is detected. It is detected at a later stage than normal, and the reconstruction error is not as large as better trained autoencoders. The outlying light green line is from sensor n215, which is largely independent from the others as it belongs to the corner of area B in L-Town. The smaller reconstruction error makes it more difficult to detect a leak under these circumstances and increases the likelihood of encountering false positives.

The experiment was stopped at a 5-day training interval, as anything below saw too high reconstruction errors and false positives were more frequent. It was therefore not considered a useful endeavor to attempt to improve the results beyond this by adjusting the threshold function. The results from a 5-day, 7-day, and 10-minute resampling and 15-minute resampling can be seen in table 9. The 5-day and 15-minute resampling have roughly the same number of samples, and the 7-day and 10-minute resampling should have the same.

Table 9 Results from 7-and 5-day time intervals, and 10- and 15-minute resampling

Pipe	7-day	5-day	10-minute	15-minute	Timestep leak occurs	Leak classification
Pipe 31	57,448	57,454	57,454	60,120	51,573	Slow increase
Pipe 158	80,314	76,637	80,470	81,684	80,097	Burst
Pipe 183	57,467	57,472	63,188	63,324	62,817	Burst
Pipe 232	10,021	10,031	-9,999	-9,999	8,687	Slow increase
Pipe 369	86,037	86,047	82,316	88,128	85,851	Burst
Pipe 461	-9,999	10,931	18,676	15,144	6,625	Slow increase
Pipe 538	39,634	39,647	39,676	39,027	39,561	Burst
Pipe 628	37,310	37,139	36,572	36,798	35,076	Slow increase
Pipe 673	18,285	18,338	16,368	18,141	18,335	Burst
Pipe 866	43,646	29,147	43,660	38,880	43,600	Burst
Correct leaks	7	7	7	6	-	-
False positives	2	3	2	3	-	-
Missed leaks	1	0	1	1	-	-

It can be seen that the results are not optimal, but also not completely unreliable. For the 15-minute resampling, even the correctly detected leaks are detected so late, and the training loss was continuously larger, that any further testing was not deemed as a useful endeavor. Similar results were seen in a 4-day run too. Unfortunately, a normal sampling rate seen the UK is set to 15-minutes, but the problem of the higher sampling rate can perhaps be compensated for by increasing the number of sensors (Mounce & Boxall, 2010).

However, it is possible that more accurately finetuning of the threshold function could avoid some of the false positives, and thereby improve the results seen in table 9. An example of a leak that is often declared as a false positive is the one found in pipe 673. The reconstruction error at the time of the leak can be seen in figure 18, the red x indicates when the pipe bursts. Throughout the testing of various configurations of the overall script, it has been considered a peculiar leak, as it is sometimes detected as a false positive. It is an abrupt burst that happens in area B. There are two nearby sensors in the same area, both of which show abnormally large reconstruction errors, larger than seen anywhere else while working with this project. The light green line going vertically ends at a remarkable reconstruction error, with a rolling mean on a one-day basis, of about 250 which belongs to the sensor n215. The light blue vertical line at a slight angle belongs to sensor n229 and peaks at a reconstruction error of 8. For reference, other reconstruction errors from sensors that do detect leaks are typically between 0.8 and 2.

Another important feature seen in figure 18 are the three lines: orange, blue and green, which starts to increase before the leak actually does occur. Why this increase occurs is not known, it can perhaps be a random fluctuation. These lines belong to the sensors n1, n4 and n31, all of which are found in area C, meaning they tend to behave more independent compared to the sensors found in area A. The fact that there is three of them means that it is enough for the threshold function to trigger before the leak actually does occur and a false positive will be detected. This is likely the reason why the leak found in pipe 673 sometimes is detected some hours before it actually does occur, or why it is detected only minutes after it does occur. Still, due to the fact that all the sensors are able to detect the leak, a slightly stricter finetuning of the threshold function would likely make the leak consistently detectable.

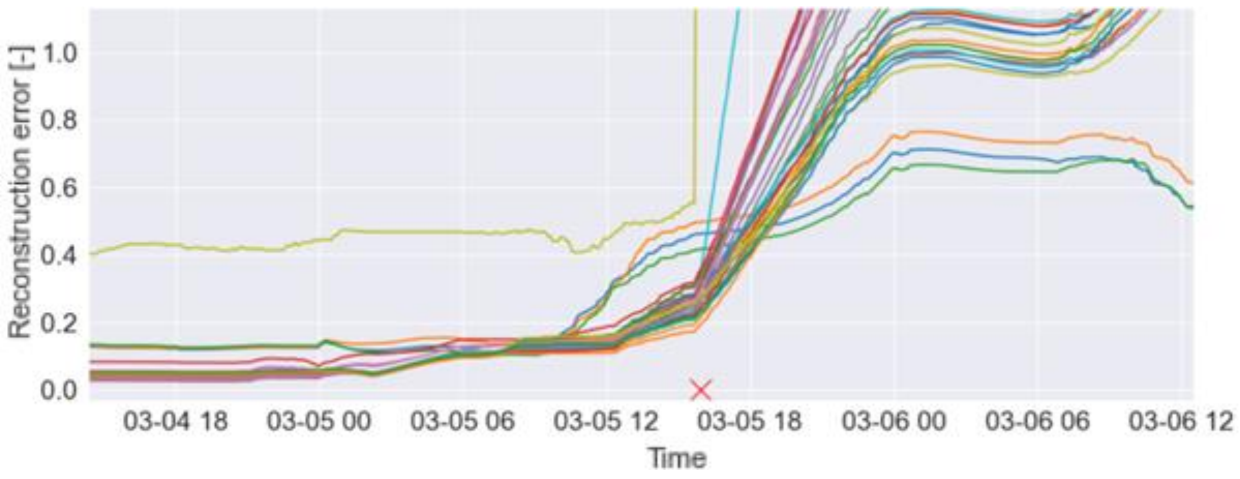

Figure 18 Shows leak in pipe 673, produced with a 7-day run, with a one-day rolling mean.

Here it is assumed that when the results deviate too far from the results seen in table 4 a line was drawn, and testing was stopped. However, it is not unreasonable to think that others would disagree as to what constitutes “too far” and would advice to keep the testing going with even smaller sample-counts. Nevertheless, this was not done here.

Thereby the boundary of sample-count needed before the autoencoder fails will be set at about $5 \cdot 12 \cdot 24 = 1440$, for a 5-day run. However, it is important to underline that anything below or above this number will not result in a binary, either functional or non-functional autoencoder for leak detection, it is a gradual change. As the training loss will likely gradually decrease with a larger sample size and subsequently the reconstruction error will increase, making any threshold function’s job easier.

The importance of this information is that it gives a rough estimation as to what is the lower end of the number of samples one would require in order to do leak detection. Although, it is important to underline more research is needed in this direction. Nevertheless, it can perhaps pinpoint the number of data samples from a leak-free WDN one would need to gather before the data can be utilized for the training of an autoencoder for later leak detection.

Still, it is important to underline that this minor test is not extensive enough to conclude anything with complete confidence. More work is needed, and it is not unreasonable to think further adjustment to the batch size and number of epochs should be done when the sample-count is changed. In this test those two parameters remained stable throughout.

3.5 Performance with reduced number of sensors.

Seeing as 33 pressure sensors for L-Town’s 10,000 inhabitant, 42.6-kilometer pipe length WDN is on the higher end, an effort was made to deduce the number of sensors required in order to still be able to have a functioning autoencoder for leak detection. Therefore, a gradual reduction of the number of sensors employed in the autoencoder was done. A number of smaller autoencoder architectures were created, with a batch normalization layer between each layer. The autoencoders were similar to the standard settings, but with a reduced size. The autoencoder architectures tested are numbered between and including 7 through 12 in table 1.

Due to the fact that the area B and C in L-Town are somewhat hydraulically separated from the larger area A, both the sensors and the leaks found in these areas were dropped first. So, in essence the focus is solely on area A of L-Town’s WDN. In the original configuration of the autoencoder, before any sensor-reduction, there are 33 sensors in L-Town. In areas B and C, there are a total of 5 sensors, leaving 28 sensors in area A. In addition, the incipient leaks were dropped from the search, as they have proven themselves to be difficult to detect regardless of circumstance. With incipient leaks and area B- and C-leaks dropped, 8 leaks remain with 5 being abrupt bursts, and 3 of the slow increase type of leak.

Beyond the first 5 sensors that were dropped from area B and C, the remainder were picked somewhat at random. It was assumed that pipes numbered close to one another were in close proximity in L-Town, and a rough attempt was made to remove sensors in such a way that the remaining ones would be evenly distributed across the WDN. No effort was made to intelligently pick the remaining sensors’ location beyond the aforementioned description. As the reconstruction error was seen to decrease across the different leak datasets, the standard deviation multiplier was reduced from 2.75 with standard settings, to 2.5 used here.

Table 10 Initial results from area A with 28 sensors, with standard deviation multiplier set to 2.5.

Leak	Timestep detected	Timestep leak occurs	Timestep difference	Classification	Flow volume at detection
Pipe 158	80,313	80,097	216	burst	$24.31\,\frac{m^3}{h}$
Pipe 183	62,959	62,817	142	burst	$16.19\,\frac{m^3}{h}$
Pipe 232	10,926	8,687	2,239	Slow increase	$25.67\,\frac{m^3}{h}$
Pipe 369	86,045	85,851	194	burst	$20.38\,\frac{m^3}{h}$
Pipe 461	13,238	6,625	6,612	Slow increase	$4.25\,\frac{m^3}{h}$
Pipe 538	39,648	39,561	87	burst	$32.26\frac{m^3}{h}$
Pipe 628	36,576	35,076	1,500	Slow increase	$5.42\,\frac{m^3}{h}$
Pipe 866	43,637	43,600	37	burst	$20.38\,\frac{m^3}{h}$

These results found in table 10 largely match the results found in the full 33 sensor-based autoencoder presented in table 4 and will be kept as a benchmark to compare the following autoencoder architectures with a lower sensor-count with.

Before conducting this small experiment, it was assumed that the autoencoder's performance would deteriorate in both its ability to detect leaks at all, and the accuracy timewise in which the leaks were first detected. However, after iteratively removing more and more sensors from the autoencoder, what was found is that the number of sensors were not as influential as initially thought in the detection of the leaks. For the leaks in area A, both the accuracy and the leak detection were found to be largely similar to the results seen in table 10, when sensors were removed. Therefore, only the final autoencoder's architecture, hyperparameters and performance will be presented. The results achieved with the smallest autoencoder can be seen in table 11.

Furthermore, the threshold function was slightly altered, with a reduction of the standard deviation multiplier from 2.5 to 2.2, and the number of sensors needed for a positive detection was reduced from 3 to 2. The specific sensors employed in the smallest autoencoder were also switched to another set of sensors roughly evenly scattered throughout the WDN in order to reduce the chance of the initial set of sensors resulting in a fluke performance. The result of this change was more or less the same, but the timesteps in which leaks were detected changed slightly, some to the model's benefit and some to the model's detriment.

The results of the smallest autoencoder, without any tank, pump or area A's demand data utilized for training and only 5 pressure sensors can be seen in table 11. Its architecture is the autoencoder number 12, in table 1. The sensors used were n150, n332, n644, n342 and n769, and their locations can be seen in figure 9.

Table 11 Performance of smallest autoencoder with 5 sensors.

Leak	Timestep detected	Timestep leak occurs	Time difference	Classification	Flow volume at detection
Pipe 158	80,311	80,097	214	burst	24.31 $\frac{m^3}{h}$
Pipe 183	62,965	62,817	148	Burst	16.20 $\frac{m^3}{h}$
Pipe 232	9,863	8,687	1,176	Slow increase	26.02 $\frac{m^3}{h}$
Pipe 369	Not found	Not found	Not found	burst	Not found
Pipe 461	13,168	6,625	6,543	Slow increase	4.16 $\frac{m^3}{h}$
Pipe 538	35,424	39,561	-4,137	Burst	0 $\frac{m^3}{h}$
Pipe 628	36,437	35,076	1,361	Slow increase	4.49 $\frac{m^3}{h}$
Pipe 866	43,679	43,600	79	burst	20.28 $\frac{m^3}{h}$

The two leaks which proved most challenging to detect are the p369 and the p538. The p369 was not detected at all unfortunately, and the p538 had a tendency of being a false positive. However, on closer inspection it is seen that the autoencoder manages to create reconstruction errors when the leaks occur, but there is a failure of the threshold function to detect them in time.

In figure 19 and 20, the reconstruction plots for the missed leaks, p369 and p538 are shown.

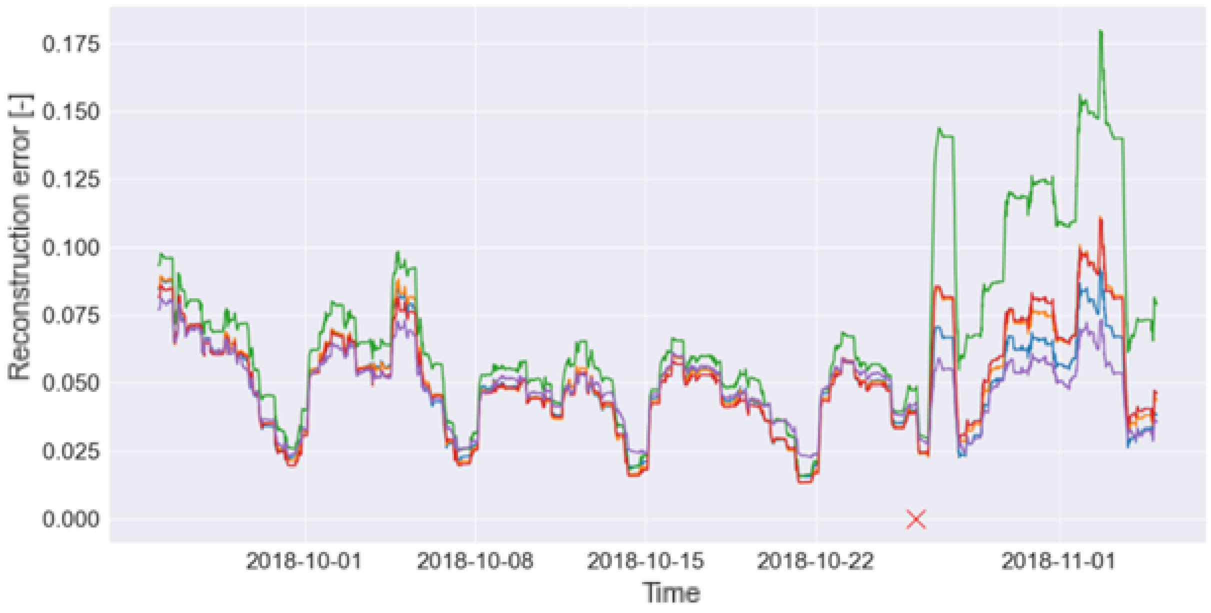

Figure 19 Shows an undetected leak in pipe 369.

In figure 19 the red x indicates when the leak in pipe p369 actually does occur, and the reconstruction plot shows that there is abnormal activity going on after this point. One sensor did indeed detect the abnormality, but due to the fact two positive sensors within a similar timespan were needed for a positive leak detection, it went by without the announcement of any leak.

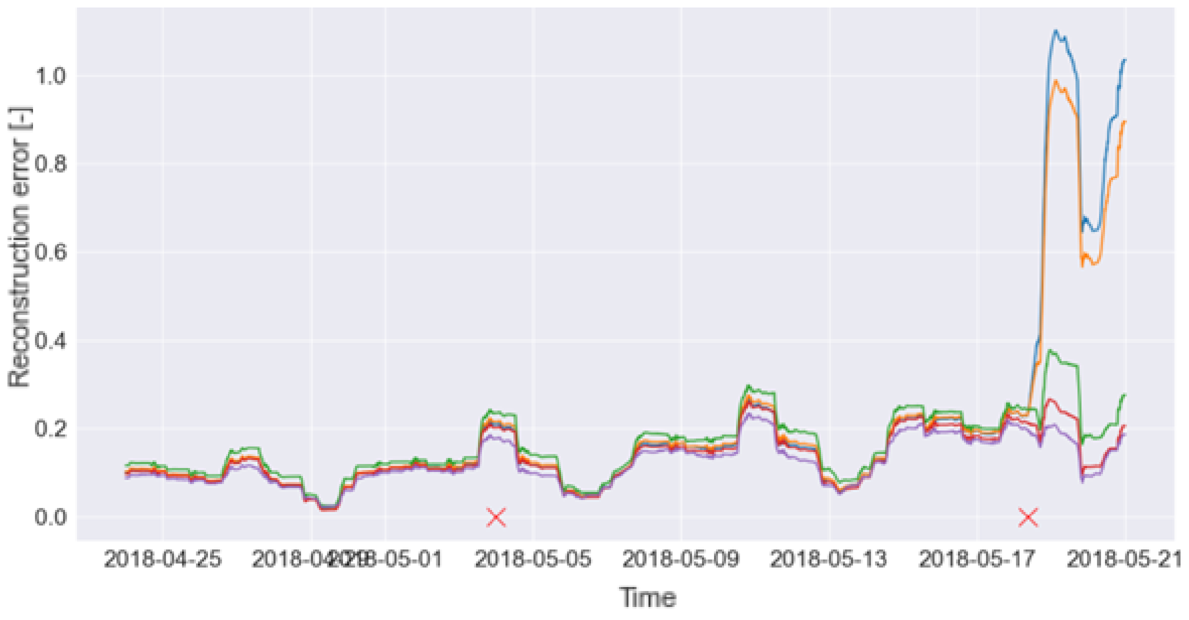

Figure 20 shows a false positive detected in pipe 538 at the first red x, the second cross shows when the leak actually does happen.

In figure 20 the reconstruction error plot of the false positive leak is shown. The first red x indicates when the false positive detection occurs, and the second red x shows when the leak actually does occur. The small peaks that repeat on a weekly basis are a consequence of L-Town's nightlife during the weekends in which water consumption was more randomly distributed throughout the day.

Seeing as both p538 and p369 are abrupt bursts, figure 21 is also included. It shows the successful detection of a slow increase leak in pipe 628. In figure 21, the first x indicates when the leak starts and the second when it is detected by the threshold function. The leak's volume is about 4.5 $\frac{m^3}{h}$ at this point.

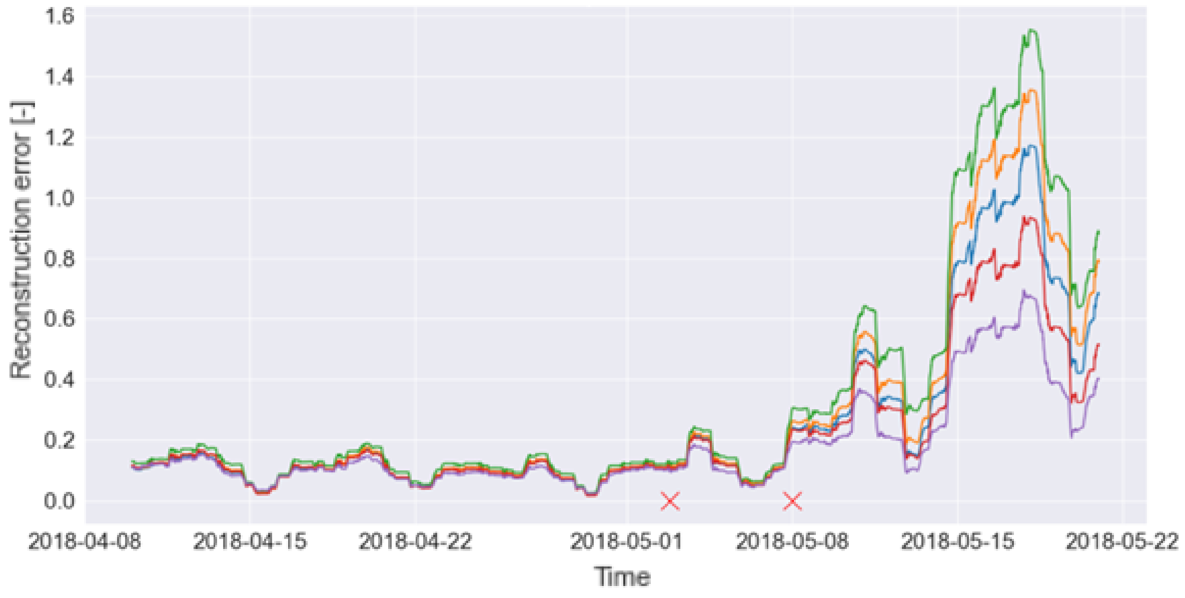

Figure 21 Shows a successful detection of a slow increase leak in pipe 628.

Figures 19 and 20 show that the failure of the smallest autoencoder was not due to the autoencoder itself, but rather the threshold function not being able to pick the leak occurrences adequately. It is possible that further finetuning of the threshold function would be able to circumvent this issue, but any further work down this path was not done due to time constraints primarily.

This is an indication that the number of sensors employed in the autoencoder may not be as important as initially expected for the detection of leaks. However, the accuracy of any rudimentary localization is reduced to a larger extent due to the fact that there are simply fewer sensors in the proximity of any broken pipe. It is reasonable to believe that the robustness of the whole system is likely to increase with more sensors, as the likelihood that the distance between a leak and a sensor is to be smaller. The sensitivity change appears to be real when implementing the aforementioned change of employing different sensors than the original 5 for the smallest autoencoder, as most leaks are detected at different timesteps.

Furthermore, as will be discussed further in 3.7, the implementation of a larger number of sensors in a WDN would allow for the creation of more elaborate threshold function designs that could perhaps reduce the likelihood of false positives compared to what is used here. One option could be to include the sensors' location and the rate of change in reconstruction error for each sensor in order to hopefully reduce the false positives and perhaps in turn allow for a rolling average on a lower time interval than the one-day basis throughout this project. It is possible that such a threshold function would reduce the detection times seen in here and its performance would likely increase as the number of sensors found in the system also increased.

3.6 On the detection of incipient leaks:

The detection of incipient leaks has turned out to be difficult with the current configuration of the model and threshold function. The incipient leaks are suspected to be too small and grow too slowly in terms of flow for the autoencoder to be able to detect the features associated with that given type of leak. That is unfortunate, and a number of attempts have been made to rectify that, all of which have been unsuccessful. At one point, it was believed that the incipient leaks would be detectable, seeing as the slow increase type is typically detected at around 4.5 $\frac{m^3}{h}$, and the incipient leaks peak in the range between 5 and 7 $\frac{m^3}{h}$.

As mentioned, a suspicion is that this is due to the fact that the leaks grow so slowly, the training data will become tainted with the leak itself, and subsequently a reconstruction error of notable size will not be created. A simple test was done to investigate this. The model was trained with standard settings, meaning the same configuration to what achieved the final results, and then trained on the leak-free start of the year and tested on a time-window later in the year, when the incipient leak had reached its peak. Nothing was detected, and upon closer inspection of the reconstruction plot, no sign of a leak was a seen, so it is not likely that the threshold function was the culprit that simply failed to detect it.

The inspection consisted of analyzing the reconstruction error and its plot in a number of ways. Oftentimes, the middle night flow (MNF), meaning the time between 02:00:00 and 04:00:00, is used for data analysis in the context of WDNs. This is due to the fact that the water consumption is lowest at this time, and whatever leaks exist, exist at all times of the day, meaning their share of the consumption is larger during the night, making them more noticeable. It is here assumed that a low water consumption during the night influence the pressure values within the WDN, which then would lead to a larger reconstruction error. Unfortunately, the reconstruction error was not sufficiently large as to be detected either by a threshold function or visually. This was the case for all the different incipient leaks.

However, and this is somewhat speculative, the assumption that a leak will create a prominent reconstruction error may hold true for bursts and slow increase leaks. Perhaps that is not the case for the incipient type. A different threshold function design could perhaps be necessary, assuming there is some feature or pattern of the reconstruction error associated with an incipient leak that could be detectable.

Although, the following can perhaps be simply due to a coincidence, or a part of the pattern typically seen in the reconstruction error towards the end of any year. It is nevertheless shown here, as the pattern is seen in two of the four incipient leaks, and the two in which the pattern is not seen both occur early on in the year, which may perhaps influence the autoencoder's ability to reach an adequate training loss before the leak starts tainting subsequent training data. In addition, for the two that do not follow the pattern, one is in area C, and the expectation that the pattern will exist there as well may not be reasonable due the hydraulic independency associated with that area.

Figures 23 and 24 both show the somewhat vague pattern of interest. Both the plots are created by using the MNF-values of the reconstruction error from an autoencoder with standard settings and the rolling mean on a half-day basis. Attempts in which seasonality and/or noise was removed obfuscated the pattern slightly, so this is not the case for the plots below. As mentioned initially, the pattern is rather vague and quite speculative, but still, it is chosen to be included as it is the closest approximation to an incipient leak found. What is seen is the following: as with other plots, it is a comparison between a sensor close to the leak, and one further away as the assumption is that the reconstruction difference between the two would be the largest. As for figure 22, the sensor n769 is closest, and for figure 23, sensors n613 are closest. Before the leaks does occur, the difference in reconstruction error is slightly smaller between the two sensors, but then gradually grows until the peak is reached at the second red x. At that point the reconstruction error stabilizes and continues with the same discrepancy between the two sensors.

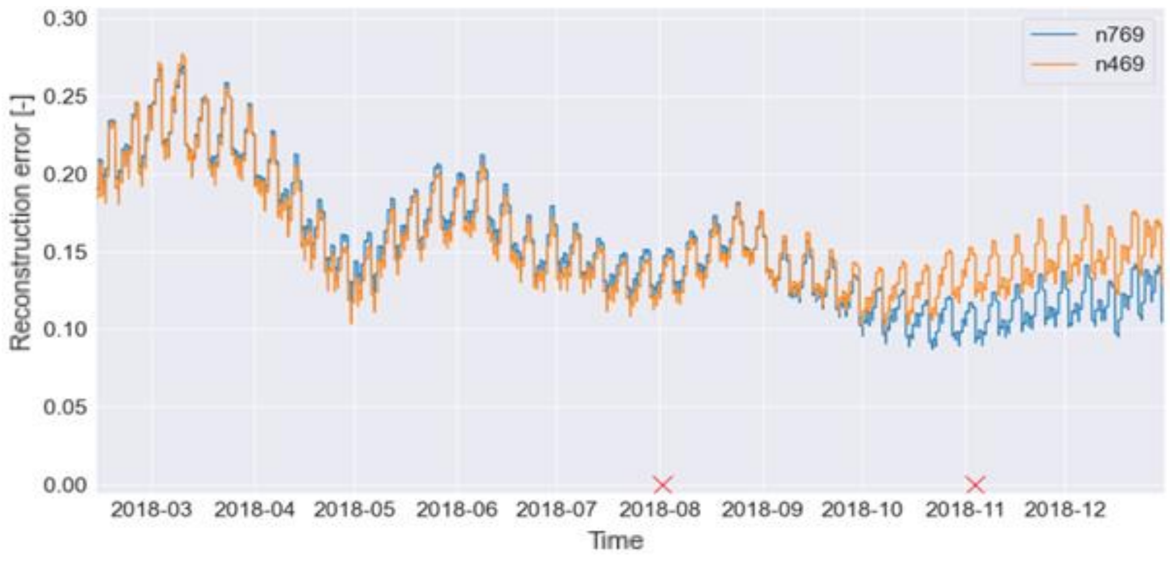

Figure 22 Comparison between close and far away sensors for leak in pipe 810. First cross when it starts, second when peak is reached.

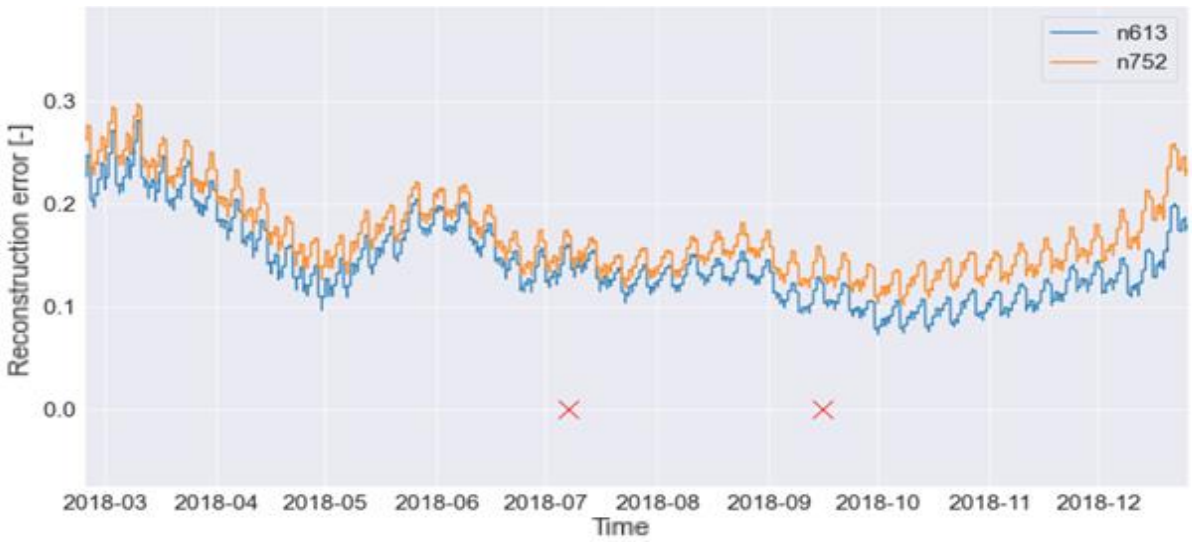

Figure 23 Comparison between close and far away sensors for leak in pipe 654. First cross when it starts, second when peak is reached.

A different threshold function design could rely on this pattern for detection. The discrepancy between all the sensors, or the rate of change in discrepancy between sensors could be used as a metric for leak detection. There likely exists other alternatives too for threshold function design, assuming this pattern is indeed an indication of a real leak. However, due to the small difference in reconstruction error it is not unlikely that such a function would also struggle with a large number of false positives. In the end, it should again be noted that this is a somewhat speculative endeavor utilizing artificial data to begin with, so in the context of a real system the likelihood of recreating and detecting the pattern is presumably rather low unfortunately.

Another strategy for the detection of the incipient leaks that unfortunately went by untested due to time constraints is to train the autoencoder specifically on the MNF-values, instead of the entire day. Although, this reduces the number of samples quite drastically, so the MNF-time interval would probably benefit from an increase to the hours between midnight and 6 in the morning perhaps, and 4 or 6 weeks rather than 2. It would then follow that the autoencoder is trained on MNF-values as well. It is believed that the autoencoder would then become more sensitive to the patterns found specifically during night-time, and perhaps a reconstruction error would become more noticeable.

3.7 On threshold function design and the number of sensors

In hindsight, a number of different possible of threshold function designs should probably have been implemented and tested in order to improve the detection times further, but the available time did not allow. There are many different threshold function design opportunities, such as a simple flat X-% increase compared to the average and a couple of standard deviations to trigger an alarm, or a design in which the change of reconstruction error per sensors is kept track of, and an alarm is triggered when a certain number of sensors deviates from some number of sensors by a large enough degree. Many more different designs could be used as well.

In order to achieve the best possible result in this project, there exists two key bottlenecks. The first being to find an autoencoder architecture and configuration that works well at creating large reconstruction errors with a short enough training time. The second, being how early the threshold function is able to detect an abnormal occurrence. In this project, a rolling average on a one-day basis have been utilized more or less throughout, unless otherwise stated. The influence of this is that fluctuations in the reconstruction error gets evened out before the threshold function does any evaluation as to whether it is able to detect a leak or not. If this was done on a shorter basis, such as half a day or six hours, the reconstruction error would fluctuate more, but if a more elaborate threshold function existed, this fluctuation could be handled, and an earlier detection time could be accomplished. An early detection time may not be that important for an incipient leak due to the low volume to begin with, but bursts and larger leaks risk compromising surrounding infrastructure if they remain undetected for a longer period of time, making an early detection time paramount.

As noted earlier, the number of sensors also play a role in how reliable a threshold function can become. Typically, the training loss decreases the more sensors are being used, and as such the reconstruction error becomes larger when abnormal occurrences happen, which is an obvious benefit. The second benefit is that the possibility of different threshold function designs increases when there are more sensors involved as there is more leeway as to when to declare a leak has occurred. In addition, comparison of the rate of change of the reconstruction error between sensors is also something that is likely a potential feature to be utilized in the design of a better-performing threshold function. Furthermore, knowing the locations of the sensors is also valuable information. The cojoining of the change in reconstruction error per sensor metric together with physical location of the sensor would allow for a more thorough threshold function than what is utilized in this project. It is shown that sensors in close proximity to the leak do detect it sooner, and it then follows if multiple sensors, majority of which are in close proximity together, detect a leak at the same time, the likelihood of a false positive declines. When spurious increases in reconstruction error occur, it is suspected that they generally do not group together based on the location within the WDN to the same extent that actual leaks do. It then follows that increasing the number of sensors in the water distribution network would reduce the likelihood of false positives. Assuming the aforementioned benefits is seen, it would allow for a reduction in the size of the rolling average used, which would increase the accuracy of the threshold function.

The design function design problem boils down to pattern recognition and signal analysis conducted on the reconstruction error. This is a more general problem that is likely seen across industries and fields of research similar as to how anomaly detection boils down to a more general problem. As such, it is not unreasonable to believe that there exist more elaborate techniques than the aforementioned that may also be beneficial for a project like this. This small feature of the overall project is a niche problem in itself and is not something typically emphasized during standard civil engineering curriculum, so it is not unreasonable that techniques or approaches not mentioned here, but routinely utilized in other industries, also would outperform the current threshold function design.

3.8 Limitations and future work.

For one, this is a preliminary investigation on whether the autoencoder can be used for leak detection, there are many obstacles before one can implement a real system, and the purpose of this project is only to function as a preliminary test to evaluate the validity of the autoencoder for such a purpose. Due to this, a variety of choices have been made throughout this project, some of which are perhaps not that realistic in a real-world setting. The motivation for these choices has been to find the leaks with the somewhat limited available resources. In addition, initially, the motivation has been to check if the autoencoder can work at all, and as such testing on artificial data is a common first-step, to exclude inadequate data preprocessing as a potential reason as to not be able to detect any leaks.

First, the entire project has been done on a computer with a CPU and not a GPU. It was initially assumed that due to the fact that the dataset is a timeseries, rather than images or some other complex type of data, a CPU would suffice. However, the training of the model takes a while, depending on the configuration at hand, and each search for a leak typically requires multiple training stages, as the time interval tested on in which a leak is not found in, is added to the training for the next iteration. The training times slowed down the overall project occasionally, and more time would've been available for testing if this was not the case. However, a sizeable chunk of this problem was circumvented by the over-night running of the script. Towards the end, it was made apparent that online resources such as Kaggle or Google Collaboratory can supply free online GPU usage. Unfortunately, Kaggle has a time limitation, and the Google Collaboratory would require moving the script from the IDE to a notebook-format, and this was chosen against due to time constraints.

As a consequence of this, it was decided that the algorithm would start training, validating, and testing on the data about one or two months prior to the leak occurring, and not the whole year. This reduces the duration in which potential false positive leak detections can occur and is not something that would be seen in a real network. But it allowed to greatly reduce the time needed to iterate through the leak datasets with different configurations of the autoencoder in order to test a variety of different settings.

Furthermore, seeing as all aspects of the leaks were known, such as size of the leak, start date, location, growth rate etc., it was possible to adjust the threshold function in such a way that it was finetuned to detect the leaks and not false positives, rather to blindly set it a certain configuration and attempt to start detecting. By and large, this finetuning was done by adjusting the standard deviation multiplier. But seeing as one type of leak may be triggered by one configuration of the threshold function, and a different type of leak by another, it was challenging to find a middle-point to fit all types of leaks and doing so in a real system is not realistic. It is not unreasonable to believe that a different threshold function design would outcompete the current design also. Perhaps multiple different threshold functions, and an overarching one to determine if there is a match of a timespan in which a leak occurs across multiple threshold functions would be a more optimal design.

One minor shortcut implemented in the threshold function is found in the check in which a sample of interest is compared with a sample occurring a few hours ahead, and the corresponding sample found 24 hours prior. The motivation was to reduce the likelihood of false positives, but it influences the earliest timestep a leak would be able to be detected in a real life. Such a threshold design could not be implemented in a real-world setting, as whatever leak was detected would always be delayed by a few hours.

As mentioned, a number of different possible of threshold function designs should probably have been implemented and tested, but the time did not allow. Due to the ability to do such finetuning of the standard deviation multiplier, one should be wary of comparing the results of this project with others, on the numerical basis of the results alone. As they have been obtained under one set of circumstances in this project, and some other project was done under a different set of circumstances. This is a common occurrence noted in the review article by (Hu, et al., 2021), that some studies use artificial or real data, high or low sampling rate, the detection of only bursts or transients too, and a host of other differences, making comparisons between studies more difficult. This was partially the motivation for the BattLeDIM competition to begin with, to have researchers attempt to tackle the same problem under the same set of circumstances.

It is suspected that the use of artificial data is the most impactful shortcut in this project overall. By using artificial data from a model of a WDN, one is able to circumvent problems that are likely to be seen in a real system. Pressure data is notoriously quite noisy. The input data used in this project also fluctuates somewhat, but any comparison between the variance found in the input data and a real dataset is not done. Although, the datasets have been vetted by a number of researchers, so this is likely not a large problem if a problem at all. Furthermore, the seasonality in the pressure data that is utilized in this project is on a one-week basis, with an increase towards the weekend, without any holidays or special events. This might also not be completely reflective of a real WDN but is a close approximation. Some WDNs belong to cities that receive a substantial number of tourists or visitors during the summer or winter months, which would drive up water consumption, which would reduce pressure in the WDN. Different minor peculiarities such as these are also not considered in the model of L-Town and the pressure data used as input. In addition, the use of artificial data allowed the search for only one leak occurring at the same time. If several did occur, one could perhaps obfuscate the other.

In a real system, one would likely experience sensor failures of some kind, whether that be the sensor itself or a failure to reliable transmit the data. To overcome these issues the typical approach is to implement a number of different data preprocessing steps, before the training of the autoencoder. These data preprocessing steps add further uncertainty to the data, and the model's accuracy after it has subsequently been trained on this data will suffer. Potential data preprocessing steps could be to interpolate missing data, scale some known leak-free data into a missing section in the timeseries or a variety of other more complex measures. Reliable data preprocessing may also not be possible depending on the quality of the remaining sensor data that is received. Seeing as the purpose for this project was a preliminary examination of the autoencoder, it was deemed reasonable to first test on artificial data. If a shortcoming was later encountered, it would then be known that it was due to the autoencoder and not inadequate data preprocessing.

Regardless of the aforementioned shortcuts, the overall goal was never to make a direct comparison with a real system or attempt to outcompete some other study. The goal was to conduct a preliminary investigation into whether an autoencoder could be used for leak detection. Later the goal grew, and a series of minor tests where conducted, as seen noted in 3.2 through 3.5.

As mentioned introductorily, (Romano, et al., 2012), have implemented a ML algorithm in a real system. They used an ANN and were able to detect simulated pipe bursts by opening fire hydrants. Since the final results show that the detection of pipe bursts can also be done with the autoencoder, while also detecting slowly increasing leaks, it is reasonable to think that an autoencoder could be used in a real system. In addition, as previously noted, the results can perhaps be improved further by re-designing the threshold function. There are however some caveats, the most meaningful of which is aforementioned use of artificial data. Yet there still is a number of other precautions or aspects to keep in mind to take before implementation in a real system. In addition, a few recommendations of different avenues to investigate further at a later stage. Some of which will be briefly discussed in the following.

When the autoencoder is trained in this project, it is trained on data that is known to be leak-free. Due to the high-water loss seen in Norwegian cities, procuring such data that one is confident is leak-free can be a challenge. It then follows that there would be a large benefit to be seen, if testing of the autoencoder on tainted data also showed promise. To the author’s knowledge, such a test has not been done. In such a system, the autoencoder would be trained on the leaks up until a certain data, with the assumption that these leaks are endemic to the system, but any subsequent leak would still appear to be an anomaly, which can perhaps be detected. Presumably the endemic leaks would cause some sort of noise or unreliability to the overarching system, but the size of this problem is not known. As such, an investigation into finding how well an autoencoder can handle training and testing on tainted data would also be beneficial. An attempt at quantifying to what the degree the WDN training data can be tainted by other leaks, while still being able to detect subsequent leaks would be interesting to see and would have large implications to the real-world implementation of a system for leak detection based on an autoencoder.

In a real WDN changes are made to the system occasionally as the city develops or routine pipe rehabilitation is done. The WDN used to retrieve the pressure data in this project is kept stable throughout the year. In a real system, pipes are occasionally taken out of service and rehabilitated, new neighborhoods are created or some other activity is done that changes the network hydraulically. If one were to train an autoencoder across such changes, how impactful they would be on the final results is not known. As long as the pressure levels remain stable throughout, the change is presumably small, but it remains as something that one should perhaps be cautious of if false positives are detected in the proximity of on-going WDN rehabilitation. Presumably, creating WDN models in which such a change is made is at some point in a year should not be too challenging, so it would be interesting to see the outcome of such a project. Although, this problem could perhaps be circumvented to some degree with strong communicational connections within the different sections of a water utility.

Furthermore, as noted in 3.3, a sensor’s location in the WDN have had an influence on the reliability of the sensors. In this project, the sensors n1, n4, n31 for area C, and n215 and n228 for area B have been significantly more prone than others when false positives were detected. As is noted in figure 15 and 16, the sensors in area C follow each other closely. The two in area B are not as closely linked, but the reconstruction error is continuously larger for n215. Based on this, in conjunction with the results from 3.5, which showed that one could achieve fairly good results with substantially fewer sensors too, it would perhaps be reasonable to have, in a real operational network, one autoencoder for each area of the overall network or DMA that is somehow separated from one another hydraulically. Such a change would perhaps also benefit the rudimentary leak localization that could be done by the autoencoder itself.

Seeing as somewhat adequate results can be obtained with a substantially smaller sensor number, it then follows that for the municipality, the entry cost to such a system is slightly lower. However, this assumes that the sensor cost is a sizeable portion of the overall cost of such a system which may not actually be the case. Still, with a lower entry cost, it would be easier for a municipality to justify the initial investment, and the performance of the whole system would then gradually increase as more sensors are purchased at a later stage. The data analysis aspect of such a system would presumably remain somewhat similar cost-wise regardless of the number of sensors implemented in the WDN and is likely a service the municipality would have to purchase, perhaps jointly with other municipalities, from a consultancy firm of some kind that offers such a service. Close communication and record-keeping of both the data analysis and the daily use and maintenance of the WDN would be a necessity between the different actors in such an ecosystem, in order to reduce the occurrences of false positives.

4 Conclusion

To conclude, and to answer the main research question: the autoencoder can indeed work for leak detection. It has shown itself reliable at detecting bursts and leaks of the slow increase classification at an early stage. It failed at detecting incipient leaks, but more research is needed to conclude whether the detection of incipient leaks with an autoencoder is an impossibility. The two different approaches to training the autoencoder were not as impactful as initially expected and the structure of the autoencoder allows it to do some basic leak localization as well. Furthermore, it was showed that the amount of data needed for leak detection was not as large as initially expected, as most leaks were found when only five days with 5-minute samples were used. The number of sensors could also be greatly reduced, assuming they are not placed in such a way they're hydraulically disconnected. Although, it should be noted that both the reduction of sensors and the different training intervals resulted in a gradually worse performance of the autoencoder. In a real-world system, after a leak is detected, a quick visual inspection of the reconstruction plot for the sensors of interest can easily be done, circumventing this problem to some degree.

For future projects there is still some work that would be interesting to see the outcome of, which can be done with artificial data as used in this project. One of which is the aforementioned training of the autoencoder on pressure values retrieved from a WDN with a number of on-going leaks, to see if the autoencoder is still able to detect any newly formed leaks. Another challenge is to tackle the problem of incipient leaks, which remained undetectable throughout this project. Besides this, the largest limitation of this project was the use of artificial data, as such the natural next step and recommendation for future work is to implement an autoencoder for leak detection without the use of artificial data. As shown, the number of sensors necessary to detect a leak isn't particularly large before a somewhat functional system is established, then any later investments would simply further improve the system further.

5 References

Anandakrishnan, A. et al., 2017. *Anomaly Detection in Finance: Editors' Introduction.* Halifax, Nova Scotia, s.n.

BattLeDIM, 2020. *BattLeDIM.* [Online]
Available at: https://battledim.ucy.ac.cy/
[Accessed May 2022].

BattLeDIM, 2020. *BattLeDIM Problem Statement.* [Online]
Available at: https://battledim.ucy.ac.cy/wp-content/uploads/2020/02/BattLeDIM_Problem_Description_and_Rules-v1.3.1.pdf
[Accessed May 2022].

Cody, A. R., Tolson, A. B. & Orchard, J., 2020. Detecting Leaks in Water Distribution Pipes Using a Deep Autoencoder and Hydroacoustic Spectrograms. *American Socity of Civil Engineers*, May.

Daniel, I. et al., 2022. A Sequential Pressure Based Algorithm for Data-Driven Leakage Identification and Model-Based Localization in Water Distribution Networks. *Journal of water resources planning and management*, June.

Fisher, S., 2016. *Waterworld.* [Online]
Available at: https://www.waterworld.com/water-utility-management/article/16202027/addressing-the-water-leakage-challenge-in-copenhagen
[Accessed May 2022].

Ghawaly, M. J., Nicholson, D. A., Archer, E. D. & Michael, W. J., 2022. Characterization of the Autoencoder Radiation Anomaly Detection (ARAD) model. *Engineering Applications of Artifical Intelligence*, 24 February.

Goodfellow, I., Bengio, Y. & Courville, A., 2016. Deep Learning. In: *Autoencoder.* s.l.:MIT Press.

Hult, F., 2022. *Vg.no.* [Online]
Available at: https://www.vg.no/nyheter/innenriks/i/BjK9jl/naa-er-det-saa-toert-at-oslo-maa-faa-vann-fra-nabokommunene
[Accessed Mai 2022].

Hu, Z. et al., 2021. Review of model-based and data-driven approaches for leak detection and location in water distribution systems. *Water Supply*, November.

Ioffe, S. & Szegedy, C., 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *Arxiv*, February.

Lalle, Y., Fourati, M., Fourati, L. & Barraca, J., 2021. Communication technologies for smart water grid applications: Overview, opportunities, and research directions. *Computer Networks*.

Liemberger, R. & Wyatt, A., 2018. Quantifying the global non-revenue water problem. *Water Science & Technology Water supply*, July.

Mounce, S. & Boxall, J., 2010. Implementation of an on-line artificial intelligence district meter area flow meter data analysis system for abnormality detection: A case study. *Water Science and Technology Water Supply*, July.

Puust, R., Kapelan, Z., Savic, D. A. & Koppel, T., 2010. A review of methods for leak management in pipe networks. *Urban Water Journal*, 24 Febuary.

PyTorch, 2016. *PyTorch.* [Online]
Available at: PyTorch.org
[Accessed January 2022].

Romano, M., Kapelan, Z. & Savic, D., 2012. Automated Detection of Pipe Bursts and other Events in Water Distribution Systems. *Journal of Water Resources Planning and Management*, May.

Romano, M., K. Z. & Savic, D., 2012. Testing of the system for detection of pipe bursts and other events in a UK water distribution system. *14th Water Distribution Systems Analysis Conference*, September.

Romano, M. K. Z. S. D., 2010. Real-time leak detection in water distribution systems. *12th Water Distribution Systems Analysis Conference*, September.

Romero-Ben, L. et al., 2022. Leak Localization in Water Distribution Networks Using Data-Driven and Model-Based Approaches. *Journal of water resources planning and management*, May.

Soldevila, A. et al., 2016. Leak localization in water distribution networks using a mixed model-based/data-driven approach. *Control engineering practice*, October.

Srivastava, N. et al., 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research* .

Unicef, 2022. *Unicef.* [Online]
Available at: https://www.unicef.org/stories/water-and-climate-change-10-things-you-should-know
[Accessed May 2022].

Wu, Y. & Liu, S., 2016. A review of data-driven approaches for burst detection in water distribution systems. *Urban Water Journal*, December.

You, S. et al., 2021. Semi-supervised automatic seizure detection using personalized anomaly detecting variational autoencoder with behind-the-ear EEG. *Computer Methods and Programs in Biomedicine*, 14 November.

6 Appendix

Appendix number	Description
A	Table of ML terminology
B	Table of hyperparameter explanations
C	Main script
D	Autoencoder example
E	Data import and threshold function

A:

Table 12 Brief overview of some ML terminology

Term	Explanation
Number of layers	By adding multiple layers to the model, one increases the number of internal parameters to be finetuned which in turn allows for the learning of more complex functions. More colloquially, on an image dataset with handwritten numbers such as MNIST, one could expect to see the first layer's output be able to pick up the edges of the number, second pick up loops, and third put them together into a number. The number of neurons often is decided by the particular problem one is trying to solve. In addition, for an autoencoder, how steep the encoder and decoder ought to be is typically solved by trial and error.
Sample	A sample in the dataset is one row of data. If the data is a timeseries, one sample would then be a timestep, and the corresponding measurements or values associated with that given timestep.
Feature	A feature in the dataset is one column of data. If the data is a timeseries, the number of features would correspond to the number of different types of measurements. The number of neurons in an input and output layer for autoencoders generally correspond to the number of features in the dataset one intends to feed the model.
Training, validation, and testing split ratios	The split ratio concerns at which sections of the overall dataset should be used for training, validation, and testing. In the case of an anomaly detection, one can continuously increase the training and validation data if an anomaly is not found, allowing for an accumulative training phase.
Training and validation loss	When training and validating the model, it is common to keep track of training and validation loss. Ideally these numbers are rapidly decreasing in the beginning and become as low as possible. Iterations with models that have continuously larger training loss generally show smaller reconstruction errors and are more likely to detect false positives.

B:

Table 13 Brief overview of the purpose of different hyperparameters

Name	Purpose
Batch size	Batch size is the number of samples the model sees before updating the model's internal parameters such as weights and biases. When choosing the size of batch, it depends on the problem one is trying to solve. Typical values range from one to several thousand, depending on the type of problem and type of data one is working with.
Epochs	The number of epochs is the number of times the model with work through the entire training dataset. Meaning with 1000 epochs, a dataset with 100 000 values, the model will see a total of 10^8 samples. If the batch size is 100, the model's internal parameters will be updated 10^6 times. As with the batch size, the choice of number of epochs depends on the problem at hand, but typically it is a higher value than the batch size, ranging from tens to several thousands.
Learning rate	The learning rate is the increment of change for each iteration of an optimization algorithm as it attempts to minimize the loss function. Typically, a learning rate is set somewhere between 0.01 and 0.00001 in machine learning algorithms, and it is a hyperparameter that needs finetuning for the model and problem at hand.
Loss function	The purpose of the loss function is to compute the difference between the expected value and the output value during the training phase. The size of discrepancy between the expected and output value determines how heavily the weights and biases, which are internal model parameters, in the backpropagation algorithm should be shifted during the learning stage for the autoencoder. A wide variety of possible functions exist for this purpose, a well-known one is known as the Mean Square Error (MSE).
Optimizer	The purpose of an optimizer algorithm is to minimize the loss function in a neural network. There are several different algorithms to be used for this purpose. Stochastic gradient descent is a widely used option, but alternatives in which the learning rate is altered is also available. The choice of optimizer is largely dependent on what performs best through trial and error and the operator's experience (Goodfellow, et al., 2016).
Activation function	The activation function's purpose is to transform the weights and biases of a neuron to a single number, typically between 0 and 1. There are number of functions that are used for this purpose. Examples being Tanh, ReLU, Leaky-ReLU and the Sigmoid function. The output from the activation punction determines the degree in which the neuron is active for subsequent passes through the neural network.
Batch normalization	Batch normalization normalizes the internal inputs from one layer of the model to the next, resulting in a more robust model architecture. Generally, by utilizing batch normalization one can utilize higher learning rates and be less careful with the initialization, making the model more robust (Ioffe & Szegedy, 2015).
Drop out	To avoid overfitting (when the model excels at training data, but fails at testing, indicating a failure to generalize), drop out is a feature developed to reduce such troubles. It works by randomly dropping a neuron in a layer and its connections, so the others have to accommodate for it, and in doing so, making the model more robust and reduces the chance of overfitting (Srivastava, et al., 2014).

C:

```python
import pandas as pd

import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import collections
import torch
import torch.nn as nn
from torch.autograd import Variable as V
from torch.utils.data import DataLoader
from random import randrange
from datetime import datetime
from statsmodels.tsa.seasonal import STL
from autoencoders_and_more import AutoEncoder_1, AutoEncoder_10,
AutoEncoder_11, AutoEncoder_12, AutoEncoder_13, AutoEncoder_14, AutoEncoder_2,
AutoEncoder_3, AutoEncoder_4, AutoEncoder_5, AutoEncoder_6, AutoEncoder_7,
AutoEncoder_8, AutoEncoder_9
from rolling_AE_functions import create_folders, import_data,
plot_reconstruction, leak_location_plot, threshold_function_v3


def train(epochs, model, model_loss):
    try:
        c = model_loss.epoch[-1]
    except:
        c = 0
    for epoch in range(epochs):
        losses = []
        dl = iter(xdl)
        for t in range(len(dl)):
            xt = next(dl)
            y_pred = model(V(xt))
            l = loss(y_pred, V(xt))
            losses.append(l)
            optimizer.zero_grad()

            l.backward()
            optimizer.step()

        val_dl = iter(tdl)
        val_scores = [score(next(val_dl)) for i in range(len(val_dl))]

        model_loss.epoch.append(c + epoch)
        model_loss.loss.append(l.item())
        model_loss.val_loss.append(np.mean(val_scores))
        if epoch%500==0 or epoch == epochs:
            print(f'Epoch: {epoch}   Loss: {l.item():.5f}    Val_Loss:
{np.mean(val_scores):.5f}')


def score(x): #score function across all sensors, only useful for evaluation
of AE's to save time.
    y_pred = model(V(x))
    x1 = V(x)
    return loss(y_pred, x1).item()


def score_v2(xt, df):
    """"
    Output from this function is a dataframe in which each cell is the MSE
between
    input cell and model's predicition for the given input cell
    """
    y_pred = model(V(xt))
    x1 = V(xt)

    n, m = np.shape(y_pred)
    for i in range (n):
        df.iloc[i] = (x1[i].detach().numpy()-y_pred[i].detach().numpy())**2/2

    return df


if __name__ == '__main__':

    time_2018 = pd.date_range(start = '2018-01-01 00:00:00', end = '2018-12-31
23:55:00', freq = '5min')
    time_2018 = {'Timesteps': time_2018}
    time_df = pd.DataFrame(data = time_2018)
    leak_flows = pd.read_csv('2018_leakages_dot.csv', delimiter = ';')

    leak_data = [{'leak':'p232', 'start': 8687,   'stop': 11634, 'h_number':
1,  'h_stop': 3},
                 {'leak': 'p461', 'start': 6625,   'stop': 26530, 'h_number':
1,  'h_stop': 4},
                 {'leak': 'p538', 'start': 39561,  'stop': 43851, 'h_number':
7,  'h_stop': 11},
                 {'leak': 'p628', 'start': 35076,  'stop': 42883, 'h_number':
6,  'h_stop': 10},
                 {'leak': 'p673', 'start': 18335,  'stop': 23455, 'h_number':
2,  'h_stop': 6},
                 {'leak': 'p866', 'start': 43600,  'stop': 46694, 'h_number':
6,  'h_stop': 10},
                {'leak': 'p158', 'start': 80097,  'stop': 85125, 'h_number':
17, 'h_stop': 20},
                {'leak': 'p183', 'start': 62817,  'stop': 70192, 'h_number':
12, 'h_stop': 16},
                {'leak': 'p369', 'start': 85851,  'stop': 89815, 'h_number':
18, 'h_stop': 21},
                {'leak':  'p31', 'start': 51573,  'stop': 64437, 'h_number':
9,  'h_stop': 20},
                {'leak': 'p654', 'start': 73715,  'stop': -9999, 'h_number':
10, 'h_stop': 24},
                {'leak': 'p257', 'start': 2312,   'stop': -9999, 'h_number':
1,  'h_stop': 24},
                {'leak': 'p810', 'start': 61254,  'stop': -9999, 'h_number':
13, 'h_stop': 24},
                {'leak': 'p427', 'start': 13301,  'stop': -9999, 'h_number':
1,  'h_stop': 24},]

    leak_df = pd.DataFrame(data = leak_data)

   #dataframe to keep the total results, the preliminary results are
concatenated for each iteration
    super_results = pd.DataFrame(index = [':)'], columns = ['Model
iteration','Dataset','Time detected leak', 'Time true leak',
'Difference','Accum. difference'], data = 0)

    all_datasets = ['p538','p183','p158', 'p369', 'p673',
'p866','p232','p461','p628','p31','p257', 'p427', 'p654', 'p810']

    models = [AutoEncoder_1, AutoEncoder_2, AutoEncoder_3, AutoEncoder_4,
AutoEncoder_5, AutoEncoder_6, AutoEncoder_7, AutoEncoder_12]
    #chosen_model = models[randrange(len(models))]

    models_container = [AutoEncoder_6 for i in range (150)]
    #models_container = [chosen_model for i in range(150)]

    for kk in range(150):
        startTime = datetime.now()

        folders = create_folders()

        loss = nn.MSELoss()

        possible_batch_sizes = [500, 750, 1000, 1500, 2000, 2500, 3000, 3500]
        batch_size = 750

        possible_epochs = [1000, 1500, 2000, 2500, 3000, 3500, 4000, 750]
        epochs = 2500
        possible_learning_rates = [0.001, 0.0001, 0.00001, 0.005, 0.0005,
0.0025, 0.00025]
        learning_rate = 0.0025

        number_of_neurons = 36 #original
        #number_of_neurons = 31 #for first iteration
        #number_of_neurons = 22 #for second iteration
        #number_of_neurons = 14 #for third iteration
        #number_of_neurons = 10 #for fourth iteration
        #number_of_neurons = 8 #fith
        #number_of_neurons = 5 #fifth, but with no tank, pump and demand,

        accumulated_diff = 0
        comparison_number_multiplier = 1
        comparison_number_multiplier_v2 = 2.75

        #time_intervals = [14, 10, 7, 5, 3, 2, 1]
        #time_intervals = [5, 1]
        time_intervals = [14] #1 day did not work, couldn't detect anything
        #time_intervals = [2]
        #h_time_multiplier = [1, 2, 2, 3, 6, 9, 14] #rough estimates
        #h_time_multiplier = [3, 12] #rough estimates
        h_time_multiplier = [1]
        for dataset_number, dataset in enumerate(all_datasets):

            model = models_container[dataset_number](number_of_neurons)
            print(model)

            adam = torch.optim.Adam(model.parameters(), lr = learning_rate)
            adamW = torch.optim.AdamW(model.parameters(), lr = learning_rate)
            nadam = torch.optim.NAdam(model.parameters(), lr = learning_rate)
            adamax = torch.optim.Adamax(model.parameters(), lr =
learning_rate)

            possible_optimizers = [adam, adamW, nadam, adamax]
            optimizer =
possible_optimizers[randrange(len(possible_optimizers))]

            res = np.array([])

            leak_found = False
            no_leak_found = False

            timesteps_in_day = 288
            num_days_tv = time_intervals[kk]
            freq_t_v = str(num_days_tv) + 'D'
            num_days = time_intervals[kk]
            freq = str(num_days)+'D'
```

```
                #timestep frequency in which the training, validation changed,
currently 2 weeks
            every_second_week = pd.bdate_range('2018-01-01','2018-12-31
23:55:00', freq = freq_t_v)

            #Empty dataframe in which to concatenate output from score-
functions later on
            Y = import_data('p232') #import any dataset, to extract columns
for dataframe-formatting
            X_per_sensor = pd.DataFrame(columns = Y.columns)

            for iii, leak in enumerate(leak_df['leak']):
                if leak == dataset:
                    dataset_nr = iii

            h = leak_df.loc[dataset_nr]['h_number']
            h = h*h_time_multiplier[kk] #for tidsjustering

            while leak_found == False:
                #updates training and validation time each iteration of the
while-loop,
                start_train_time = every_second_week[h-1]
                train_time = every_second_week[h]
                vald_time = every_second_week[h+1]

                if h >
leak_df.loc[dataset_nr]['h_stop']*h_time_multiplier[kk]:
                    leak_found = True
                    leak_timestep = -9999
                    break

                print(f'Start training time:  {start_train_time}, training
until: {train_time}, validation until: {vald_time}')

                #if end of year and leak still not found.
                if vald_time >= pd.to_datetime("2018-12-31 00:00:00"):
                    no_leak_found = True
                    break

                #separate timeframe-variable to keep track of training times,
freq is also 14 days.
                test_timeframe = pd.bdate_range(vald_time,'2018-12-31
23:55:00', freq = freq)

                time2 = test_timeframe[0]
                time3 = test_timeframe[1]
                print(f'Testing from {time2} until {time3}')

                #keep track of valid_time timestep in numerical value for
later
                vald_time_int = time_df.loc[time_df['Timesteps'] == vald_time]
                vald_time_int = int(vald_time_int.index.values)

                #import the dataframe with the given dataset, and normalize
                X = import_data(dataset)
                X = X[:'2018-12-31'] #only use year 2018
                X = (X - X[:train_time].mean())/X[:train_time].std()

                #training and validation data is extracted from X
                X_train = X[start_train_time:train_time]
                #X_train = X[:train_time]

                X_val = X[train_time:vald_time]
                xtr = torch.FloatTensor(X_train.values)
                xtv = torch.FloatTensor(X_val.values)

                #testing the AE between time2 and time3
                X_test = X[time2:time3]
                xt = torch.FloatTensor(X_test.values)

                #creating the dataloader for validation and training
                xdl = DataLoader(xtr, batch_size=batch_size)
                tdl = DataLoader(xtv, batch_size=batch_size)

                #Utilize a named tuple to keep track of scores at each epoch
                model_hist = collections.namedtuple('Model', 'epoch loss
val_loss')
                model_loss = model_hist(epoch=[], loss=[], val_loss=[])

                #training the AE
                train(model=model, epochs=epochs, model_loss=model_loss)

                #makes a copy of the X_test dataframe to have all the values
set to zero, before later filling it from score functions
                X_test_copy = X_test.copy()
                for col in X_test_copy.columns:
                    X_test_copy[col].values[:] = 0

                iter_res = []
                xt = torch.FloatTensor(X_test.values)

                for i in range(len(xt)-2):
                    b = xt[(i):(i+2)]
                    iter_res.append(score(b))
                #fills up the aforementioned dataframe with the MSE per sensor
                X_test_copy_filled = score_v2(xt, X_test_copy)
                X_per_sensor = pd.concat([X_per_sensor, X_test_copy_filled])
                res = np.concatenate([res, np.array(iter_res)])

                #X_per_sensor.to_csv(f'e{epochs}_bs{batch_size}_lr{learning_ra
te}_dataset{dataset}_.csv')

                if h > 1:
                    thres_results_df, leak_found, leak_timestep =
threshold_function_v3(X_per_sensor, dataset, vald_time_int,
comparison_number_multiplier, comparison_number_multiplier_v2)

                print(f'Leak found: {leak_found}')
                if leak_found:
                    print(f'Leak is found : -), {dataset}')
                    print(thres_results_df)
                else:
                    leak_timestep = 0
                h = h + 1

                #various model configuration information
            modeltxt = (f'Epochs: {epochs}, Learning rate: {learning_rate},
Batch_size: {batch_size}, Optimizer: {optimizer}, Loss function: {loss},
Dataset = {dataset} \n\n Model: {model}')
            modeltxt2 = (f'Training time ended at: {train_time}, validation
time ended at: {vald_time}')
            modeltxt_total = modeltxt + modeltxt2
            print(modeltxt_total)
            # txt_file = open(folders[3]+'\_'+str(kk)+'_12_04.txt', "w")
            # txt_file.write(modeltxt_total)
            # txt_file.close()

                    #creates a dataframe in which to save results
            super_results_holder = pd.DataFrame(index = [':)'], columns =
['Model iteration','Dataset','Time detected leak', 'Time true leak',
'Difference','Accum. difference'], data = 0)
            super_results_holder['Model iteration'] = kk
            super_results_holder['Dataset'] = dataset
            super_results_holder['Time detected leak'] = leak_timestep
            if leak_timestep != 0:
                super_results_holder['Volume'] =
leak_flows.iloc[leak_timestep][dataset]
            else:
                super_results_holder['Volume'] = -9999

            #super_results_holder['Threshold'] = threshold_number
            if no_leak_found:
                super_results_holder['Time detected leak'] == -9999

            super_results_holder['Time true leak'] =
leak_df['start'][dataset_nr]

            if no_leak_found == False:
                accumulated_diff = abs(leak_timestep -
leak_df['start'][dataset_nr]) + accumulated_diff

            super_results_holder['Difference'] = (abs(leak_timestep -
leak_df['start'][dataset_nr]))
            super_results_holder['Accum. difference'] = accumulated_diff
            super_results_holder['Running time 1 DS'] = str(datetime.now()-
startTime)
            super_results_holder['cmp_1'] = comparison_number_multiplier #for
averages
            super_results_holder['cmp_2'] = comparison_number_multiplier_v2
#for STDs

            super_results = pd.concat([super_results, super_results_holder])

            print(super_results)
            #super_results.to_csv(folders[2] +'\_21_may_timeinterval_testing_'
+ (str(kk)) + '.csv')
```

D:

```
import torch
import torch.nn as nn

class AutoEncoder_6(nn.Module):
    def __init__(self, length):
        super().__init__()
        self.lin1 = nn.Linear(length, 26)
        self.lin2_bn = nn.BatchNorm1d(26)
        self.lin2 = nn.Linear(26, 12)
        self.lin3_bn = nn.BatchNorm1d(12)
        self.lin3 = nn.Linear(12, 8)
        self.lin4_bn = nn.BatchNorm1d(8)
        self.lin4 = nn.Linear(8, 4)


        self.lin5 = nn.Linear(4, 8)
        self.lin6_bn = nn.BatchNorm1d(8)
        self.lin6 = nn.Linear(8, 12)
        self.lin7_bn = nn.BatchNorm1d(12)
        self.lin7 = nn.Linear(12, 26)
        self.lin8_bn = nn.BatchNorm1d(26)
        self.lin8 = nn.Linear(26, length)

    def forward(self, data):

        x = torch.tanh(self.lin1(data))
        x = torch.tanh(self.lin2(self.lin2_bn(x)))
        x = torch.tanh(self.lin3(self.lin3_bn(x)))
        x = torch.tanh(self.lin4(self.lin4_bn(x)))

        x = torch.tanh(self.lin5(x))
        x = torch.tanh(self.lin6(self.lin6_bn(x)))
        x = torch.tanh(self.lin7(self.lin7_bn(x)))
        x = torch.tanh(self.lin8(self.lin8_bn(x)))

        return x
```

E:

```
import pandas as pd
import numpy as np
from datetime import datetime

import matplotlib as mpl
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
import seaborn as sns

import os
import os.path

import math
from statsmodels.tsa.seasonal import STL

def import_data(dataset):

    file_path = f'data/{dataset}/'
    print(dataset)
    sensor_columns = ['n1', 'n4', 'n31', 'n54', 'n469', 'n415', 'n519',
'n495', 'n105', 'n114', 'n516', 'n549',
                 'n332', 'n506', 'n188', 'n410', 'n429', 'n458', 'n613',
'n342', 'n163', 'n215', 'n229',
                'n644', 'n636', 'n679', 'n288', 'n726', 'n296', 'n722',
'n752', 'n769', 'n740']

    X = pd.read_csv(file_path + 'Levels.csv', index_col=0, parse_dates=[0])
    pressure = pd.read_csv(file_path +'Pressures.csv', index_col=0,
parse_dates=[0], usecols=sensor_columns)
```

```
    pump_flows = pd.read_csv(file_path + 'Flows.csv', index_col=0,
parse_dates=[0]).squeeze()

    X = X.reset_index()
    pressure = pressure.reset_index()
    X = pd.concat([X, pressure], axis = 1)
    X = X.set_index('index')

    #X = X.drop('T1', axis = 1)
    X['PUMP_1'] = pump_flows['PUMP_1']
    X['Demand'] = pump_flows['p227']+pump_flows['p235']-pump_flows['PUMP_1']

    #X = X.resample('15T').sum()
    #X = X.between_time('00:00:00','06:00:00')
    return X

def cluster(data, maxgap):
    data.sort()
    groups = [[data[0]]]
    for x in data[1:]:
        if abs(x - groups[-1][-1]) <= maxgap:
            groups[-1].append(x)
        else:
            groups.append([x])
    return groups

def threshold_function_v3(X_per_sensor, dataset, vald_time_int,
comparison_number_multiplier, comparison_number_multiplier_v2):
    time_2018 = pd.date_range(start = '2018-01-01 00:00:00', end = '2018-12-31
23:55:00', freq = '5min')
    time_df = pd.DataFrame(columns = ['Timesteps'] ,data = time_2018)

    hour_time = pd.timedelta_range(start = '00:00:00', end='23:59:00',freq =
'1H')

    leak_data_modified = [{'leak': 'p654', 'start': 20690,  'stop': 45000,
'leak time': time_df.iloc[20690], 'leak time numeric': 20690
                          {'leak': 'p628', 'start': 35076,  'stop': 42883,
'leak time': time_df.iloc[35076], 'leak time numeric': 35076},
                          {'leak': 'p866', 'start': 43600,  'stop': 46694,
'leak time': time_df.iloc[43600], 'leak time numeric': 43600},
                          {'leak': 'p183', 'start': 62817,  'stop': 70192,
'leak time': time_df.iloc[62817], 'leak time numeric': 62817},
                          {'leak': 'p369', 'start': 80000,  'stop': 89815,
'leak time': time_df.iloc[85851], 'leak time numeric': 85851},
                          {'leak': 'p538', 'start': 39561,  'stop': 43851,
'leak time': time_df.iloc[39561], 'leak time numeric': 39561},
                           {'leak': 'p158', 'start': 80097,  'stop': 85125,
'leak time': time_df.iloc[80097], 'leak time numeric': 80097},
                           {'leak': 'p232', 'start': 8687,   'stop': 11634,
'leak time': time_df.iloc[8687],  'leak time numeric': 8687},
                           {'leak': 'p461', 'start': 6625,   'stop': 26530,
'leak time': time_df.iloc[6625],  'leak time numeric': 6625},
                           {'leak': 'p31' , 'start': 51573,  'stop': 64437,
'leak time': time_df.iloc[51573], 'leak time numeric': 51573},
                           {'leak': 'p673', 'start': 18335,  'stop': 23455,
'leak time': time_df.iloc[18335], 'leak time numeric': 18335},
                           {'leak': 'p257', 'start': 2312,   'stop': 105120,
'leak time': time_df.iloc[2312], 'leak time numeric': 2312},
                           {'leak': 'p427', 'start': 13301,  'stop': 105120,
'leak time': time_df.iloc[13301], 'leak time numeric': 13301},
                           {'leak': 'p654', 'start': 53968,  'stop': 105120,
'leak time': time_df.iloc[53968], 'leak time numeric': 53968},
                           {'leak': 'p810', 'start': 60706,  'stop': 105120,
'leak time': time_df.iloc[60706], 'leak time numeric': 60706}]

    leak_df_modified = pd.DataFrame(data = leak_data_modified)
    leak_df_modified = leak_df_modified.set_index('leak')
    leak_timestep = 0

    numb_end_sensors = 34

    prelim_results_dataframe = pd.DataFrame(index  = ['Detected', 'Timestep'],
columns = [X_per_sensor.columns[1:numb_end_sensors]], data = 0)

    threshold_time = datetime.now()
    number = float()
    for col in X_per_sensor.columns[1:numb_end_sensors]:
        print(f'{col}, {datetime.now()-threshold_time}')
        res_df = X_per_sensor[col].copy()
        res_df = res_df[~res_df.index.duplicated()]
        time_rolling = 288
        res_df = res_df.rolling(time_rolling).mean()
        stds, avgs = np.zeros(shape = (len(hour_time)+1, 1)), np.zeros(shape =
(len(hour_time)+1, 1))
        number_of_days = 5
        for j, ind in enumerate(res_df.index[time_rolling:]):
            possible_leak_detected = False
            number = res_df.loc[ind].copy()

            if (j % (288*number_of_days) == 0 and j != 0):

                h = [res_df[(j-
(288*number_of_days)):j].between_time(str(hour_time[i-1])[7:],
str(hour_time[i])[7:]) for i in range (1, len(hour_time))]
```

```
                stds, avgs = list(), list()
                for hh in h:
                    stds.append(hh.std(axis=0))
                    avgs.append(hh.mean(axis=0))

                stds.append(res_df[(j-
(288*number_of_days)):j].between_time(str(hour_time[-1])[7:],
str(hour_time[0])[7:]).std(axis=0))
                avgs.append(res_df[(j-
(288*number_of_days)):j].between_time(str(hour_time[-1])[7:],
str(hour_time[0])[7:]).mean(axis=0))

            comparison_number =
(comparison_number_multiplier*float(stds[ind.hour])+comparison_number_multipli
er_v2*float(avgs[ind.hour]))

            if float(number) >= float(comparison_number) and
float(comparison_number) != 0:

                for k in range (3):
                    try:
                        next_check_number_ts = ind + pd.DateOffset(hours = k)
                        back_check_num_ts = ind - pd.DateOffset(hours =
abs(24-k))

                        next_check_number = res_df.loc[next_check_number_ts]
                        back_check_num = res_df.loc[back_check_num_ts]

                    except:
                        pass

                    if float(next_check_number) > float(back_check_num) and
[float(next_check_number) > float(res_df.loc[ind-pd.DateOffset(days = 2*i)])
for i in range(2)] and float(back_check_num) != 0.0:
                        prelim_results_dataframe.loc['Detected'][col] = 1
                        leak_timestep_int =
int(np.where([time_df['Timesteps']==ind])[1])
                        prelim_results_dataframe.loc['Timestep'][col] =
leak_timestep_int
                        print(prelim_results_dataframe)

                        possible_leak_detected = True
                        break
                if possible_leak_detected:
                    print(f'possible leak detected, {datetime.now()-
threshold_time}')
                    break
```

```
        ts_values =
prelim_results_dataframe.loc['Timestep'].copy()
        clusters = cluster(ts_values.values, maxgap = 288*2)

        empty_sensors = np.zeros([1, numb_end_sensors-1])

        try:
            if len(clusters)==0:
                leak_found = False
                return prelim_results_dataframe, leak_found, leak_timestep
        except:
            pass

        if (len(np.array(clusters[0])) > 0) and
(np.array_equal(np.array(clusters[0]), empty_sensors[0]) == False):
            max_length_cluster = max([len(clusters[i]) for i in
range(len(clusters))])

            for jj in range (len(clusters)):
                if len(clusters[jj]) == max_length_cluster:
                    max_length_cluster_nr = jj

            if (clusters[max_length_cluster_nr][0]==0):
                if len(clusters) >=1:
                    clusters.pop(max_length_cluster_nr)

                max_length_cluster = max([len(clusters[ii]) for ii in
range(len(clusters))])
                for hh in range (len(clusters)):
                    if len(clusters[hh]) == max_length_cluster:
                        max_length_cluster_nr = hh

            biggest_cluster = clusters[max_length_cluster_nr]
            leak_timestep = np.min(biggest_cluster)

       if math.isclose(leak_timestep, leak_df_modified.loc[dataset]['leak time
numeric'], abs_tol=105120) and len(biggest_cluster) >= 3:
                leak_found = True
       else:
                leak_found = False

    return prelim_results_dataframe, leak_found, leak_timestep
```