Detection of anomalies in up- and download speed and latency of Internet connections

Finn-Frederik Jannsen (finn-frederik.jannsen@haw-hamburg.de)

Keywords

CNN, CAE, NN, Autoencoder, Convolutional, Neural Net, Download, Upload, Latency, Quality of Service, Speedtest

Abstract

This projects' main goal is to detect Quality of Service issues in general end-user Internet connections. The application could potentially increase awareness and certainty of such issues. To achieve this a convolutional neural net (CNN) autoencoder is used to detect anomalies on three inputs. The inputs are upload speed, download speed and latency and a model is trained for each one separately. A detection is done by analyzing the prediction error of the model when feeding it with one of the metrics. To do so an error threshold is selected and data points in a certain temporal vicinity that are breaching that threshold are collected together. At a given size of such a collection it is considered to point to an anomaly. As a software foundation the Keras library, which is based on Tensorflow 2.3.1, will be used to construct and use the model. For training and testing a dataset that's been recorded over a weeks time using a DSL/LTE Hybrid connection is used. The connection is inherently unstable and is hence useful for detection purposes. For training purposes it is filtered for anomalies beforehand. The results show a degree of applicability after tuning of the model but further experimenting would be needed to create a reliable and well-rounded application.

1 Introduction

Consumer internet speeds have long been lower than advertised when entering a contract with an Internet Service Provider (ISP). While the average customer reports getting about 80% of the advertised speeds there are many people getting only 55% of what they paid for [1]. While a certain amount of reduction of the download and upload rate is to be expected after factoring in the nature of multiplexing a connection, as well as other overhead, there is a limit to how far a sold product should deviate from its advertised specifications. This problem has been going unnoticed to a high degree due to its non transparent nature. The average consumer won't notice a gap in performance unless the bandwidth is used up as much as possible. One scenario where this can happen is in a game manager where lots of files are pulled over multiple connections to saturate the bandwidth. Another case where one can notice it is by actively monitoring the internet speed metrics. Since the latter isn't normally done on a regular basis, many times such a gap between advertised and delivered speeds won't be noticed. This project is about providing a solution to this problem: As a foundation to the program three metrics (up-/download speeds and latency) are collected on a regular basis and are then analyzed by a machine learning algorithm. The algorithms purpose is finding anomalies in the collected data using a time series prediction. Found anomalies and the rest of the data are then reported back to the user. This way conclusions about the Quality of Service (QoS) can be drawn.
While there has been research about the QoS of ISPs, it is typically about establishing a QoS metric and mostly from an ISP's position [3]. While there are existing methods to establish a metric from a consumer's perspective they are consisting of much theory and aren't very practical [4]. In order to decide whether an end-user is satisfied with the product, that user needs to make an easy to understand picture for himself and this is what this project aims to enable.

2 Methods

To identify anomalies in the aforementioned metrics latency, up- and download speed, we need to collect measurements first. This is done using the speedtest-cli written by user 'sivel' on GitHub[5].
It is a Python library containing core functionalities for doing speedtests using Ookla's service hosted on 'speedtest.net'. While there are some concerns about the reliability of the reported data expressed by the author himself, these are mainly about small deviations from speedtest.net's results or more importantly the limitations of HTTP-based speedtests. Apparently the latter one becomes an issue when trying to measure speeds in the hundreds of MBit/s [2], which was proven to be correct when testing speeds on a 1GBit/s connection. Despite the limitations it was deemed reliable enough for superficial analysis on lower-end speeds (below 100MBit/s) using machine-learning techniques.
The underlying Neural Network (NN) used to determine the anomalies is a 1D Convolutional Neural Network (CNN) built using Tensorflow Version '2.3.1'. More precisely it is a reconstruction convolutional autoencoder (CAE) which means that the model is being used to predict the next value in a time series and delivers an error value. This value is expressing the degree of error between predicted and actual input. It is then used with a given error threshold to determine whether an anomaly candidate was found or not.
The CNN model is used 3 times separately, one for each metric, since each one behaves in a unique manner in both heavy and low load scenarios.

2.1 Model

The model is a reconstruction convolutional autoencoder as proposed in a code example for anomaly detection in time series. A summary of the layers is shown in figure 2.1. It consists of 2 Convolutional 1D layers which are downscaling the width from 16 to 4 and 2 Convolutional 1D Transpose Layers which are upscaling the width back to 16 again, all using a stride of 2 and a kernel size of 7. The number of filters for each layer is

expressed as the last value of their output shapes. The final Conv1DTranspose Layer compresses the depth of the feature map back to 1 to return the output to the original format. 2 dropout layers are used to avoid overfitting and to generalize the model. For the dropouts a rate of 0.2 is being used. Since the models purpose is a reconstruction of the input, the training data is being used both as input and target for the fitting process. This is done to achieve an output resembling the input as close as possible.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d (Conv1D)              (None, 8, 32)             256
_________________________________________________________________
dropout (Dropout)            (None, 8, 32)             0
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 4, 16)             3600
_________________________________________________________________
conv1d_transpose (Conv1DTran (None, 8, 16)             1808
_________________________________________________________________
dropout_1 (Dropout)          (None, 8, 16)             0
_________________________________________________________________
conv1d_transpose_1 (Conv1DTr (None, 16, 32)            3616
_________________________________________________________________
conv1d_transpose_2 (Conv1DTr (None, 16, 1)             225
=================================================================
Total params: 9,505
Trainable params: 9,505
Non-trainable params: 0
_________________________________________________________________
```

Figure 2.1: Model-Layers. The input is being scaled down to 1/4th the original size by using convolutional layers and is being scaled up again by convolutional transpose layers. Dropouts are placed in between to generalize the model and avoid overfitting

2.2 Autoencoder

For the purpose of understanding how to find an anomaly using this model it is important to understand the concept of an autoencoder. An autoencoder is by definition an automatic encoder, which is achieved by using neural networks for both encoding and decoding purposes. A basic schematic of an NN-based autoencoder is shown in figure 2.3.

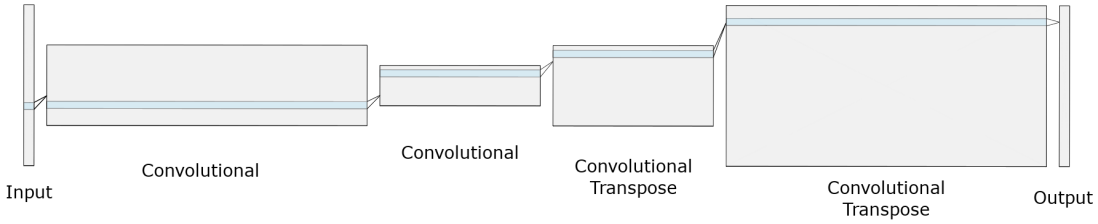

Figure 2.2: Model-Layers Visualization

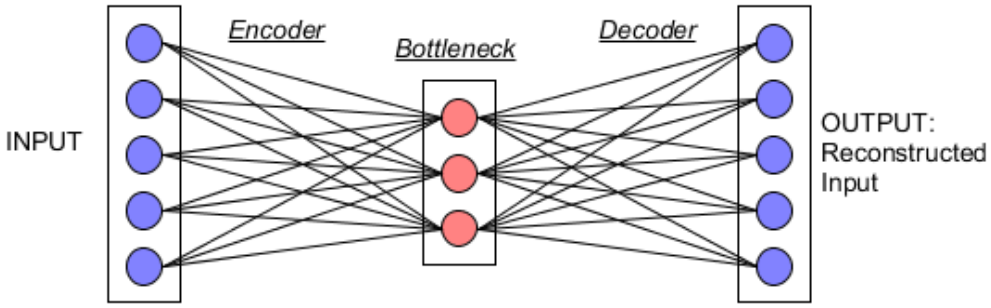

Figure 2.3: Autoencoder

In figure 2.2 one can see how the size of the data representation (height in the picture) is being compressed (encoded) first and is being decompressed (decoded) afterwards. As explained in section 2.1 the output is compared to the input to adjust the weights for reconstruction. After training the model is supposed to reconstruct it's input. However, if the output is not resembling the input at all an error value is given which can then be used to determine whether the given input data contains out of the ordinary values.
This way the inability of reconstruction of the autoencoder is in reverse used for detecting abnormal behaviour which is assumed to be an anomaly.

2.3 Data

Datasets including the metrics up-/download speed and latency for end-user internet connections are either scarce or non-existent at the time of training. This means that the data needed for training has to be recorded first. To solve this problem and to collect

data in real-time when running the finished program later on a wrapper was written for the aforementioned speedtest-cli, that allows for both direct access using callbacks as well as simple recording to a .csv file.
Roughly one week of data was recorded on a connection that uses a merged DSL/LTE connection and is rated for 50MBit/s and 10MBit/s Upload. While the DSL connection is only rated for 6MBit/s, it is the LTE connection that's delivering the rest (and most) of the bandwidth. Due to the wireless component of LTE it is further expected that a higher amount of diminishing factors can potentially impact the connection quality when compared to a wired connection [6].
The dataset plotted in figure 2.4 shows some signs of these quality impacts:
Looking at the latency, referred to as ping, there are occasional spikes into the 100-300ms range. This is while it usually stays near a minimum of around 25ms.
While the upload rate is capping out at 11.3MBit/s it is also experiencing some spikes into the 5MBit/s range.
The download rate is the most unstable of all three metrics with a clear 24h cycle and a heavy continuous fluctuation.

General rules for inputting any data have been established during the model optimization process:

- All data is being normalized using its respective mean and standard deviation
- 5 minutes are used as the sampling interval to create a view of an adequate time-frame and dismiss very short spikes. This should also cause less traffic and CPU usage than smaller intervals which is desirable as a consumer.
- The length of a sequence that's used as input will be 16. This was chosen to allow for a mediocre startup-time of $16 * 5min = 80min$ while having enough samples for a time series.

Other rules could be chosen to alter the behaviour, e.g. to specifically watch out for micro-spikes.

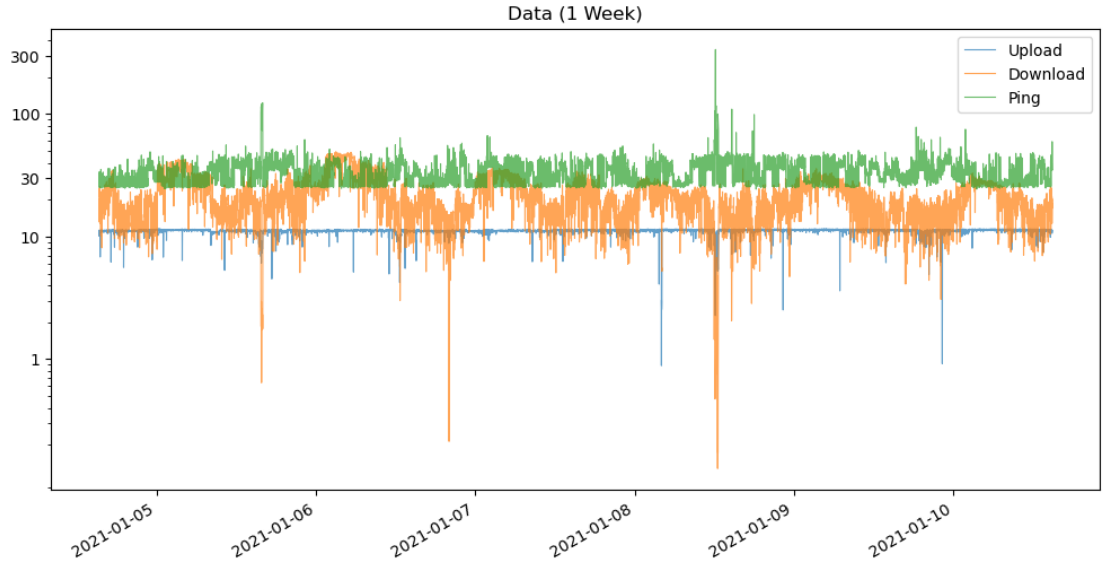

Figure 2.4: Dataset displayed using a logarithmic y-axis. Ping in ms, Up-/Downloadspeed in MBit/s

3 Training

In order to train the model for normal behaviour the dataset needs prepared first. To prepare the data outliers have been filtered from the original data. Additionally, since the download-rate is volatile, only the 3 most steady daycycles have been chosen for training. The training data for each metric can be seen in figures 3.1, 3.2 and 3.3.

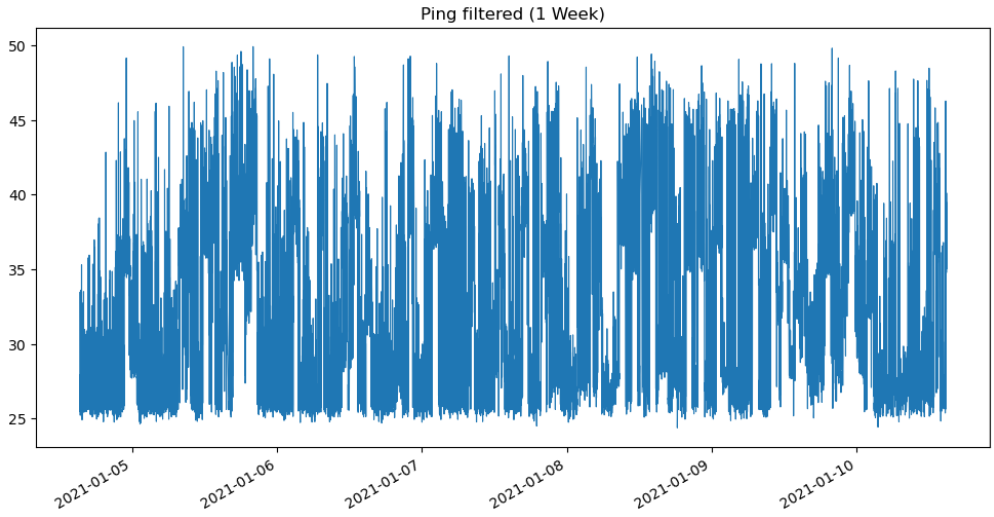

Figure 3.1: Training data for ping. Values in ms

For training the Adam optimizer was used. Furthermore a batch size of 128 and 150 epochs were used. The actual training was only running for 15-30 epochs on each metric due to usage of early stopping (with validation loss). The loss graph and MAE histogram are very similar for each metric. Hence the diagrams for download speed are chosen here to represent the general effectiveness of training in figure 3.4 and 3.5. At first glance the presented model seems to be learning the sequences well.

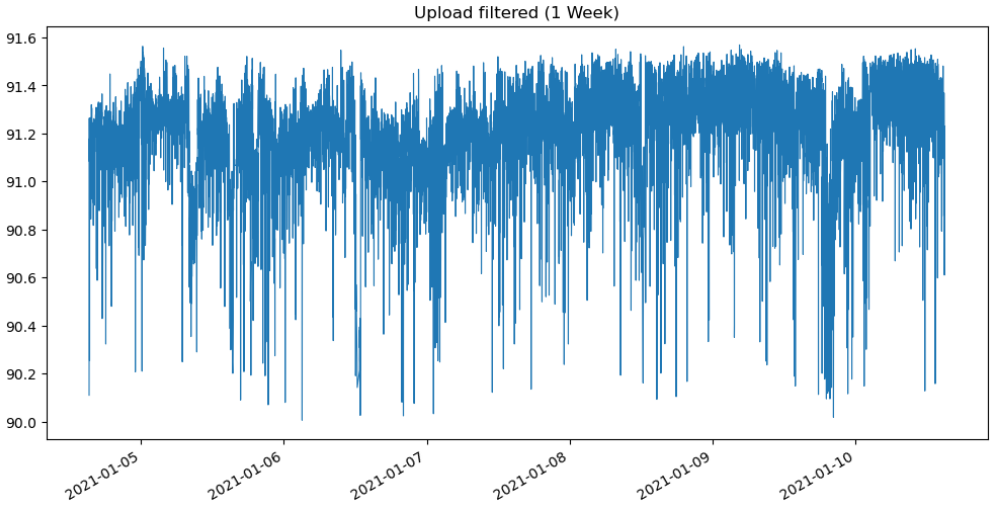

Figure 3.2: Training data for upload. Values in MBit/s

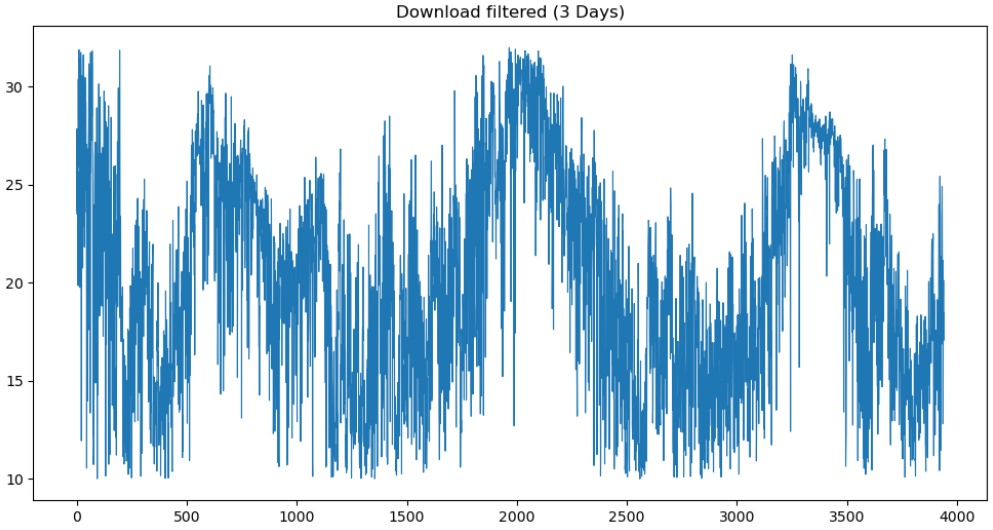

Figure 3.3: Training data for download. Values in MBit/s

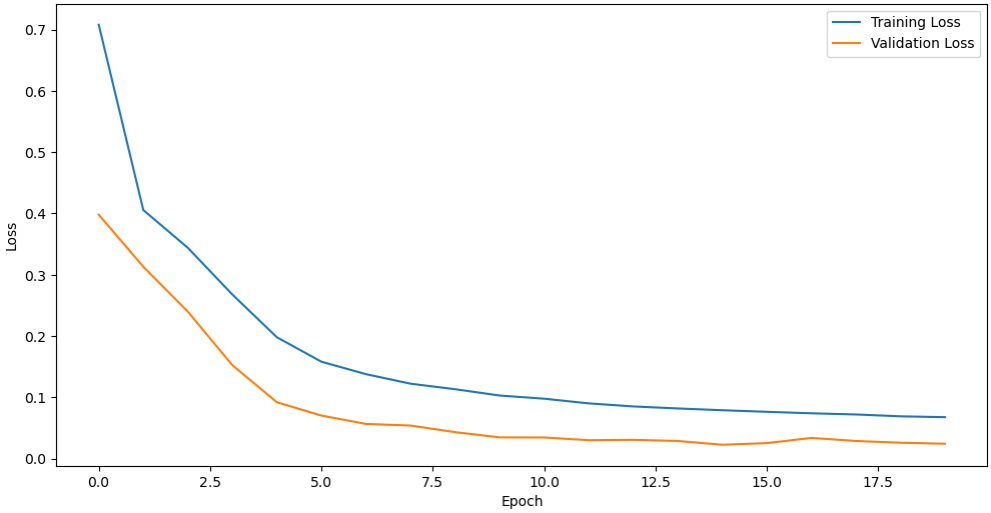

Figure 3.4: Loss graph for download speed metric

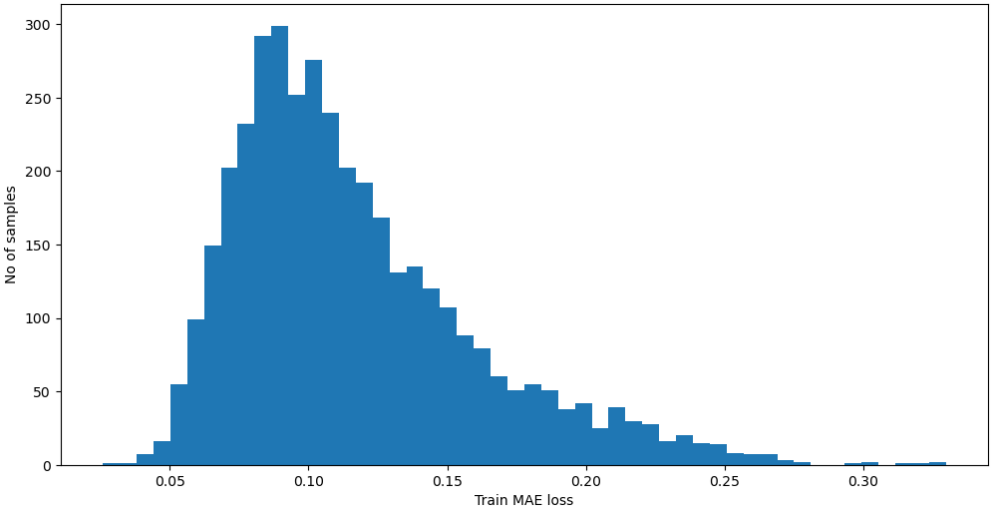

Figure 3.5: MAE histogram for download speed metric

4 Detection

To detect anomalies using the autoencoder model there are a few steps to go through. First of all rules have to be established that define what an anomaly is. Secondly anomalies need to be flagged and reported in a way that's comprehensible for the user.

4.1 Anomalies

The type of anomalies that are targeted for detection in this project are outliers and non-typical sequences.In order to to recognize an anomaly that's happened, an analysis of the prediction error of each input needs to be done. A classification using labels with this model (as done with other typical NN applications) is not possible due to the nature of it being an autoencoder. To establish rules for this analysis, experiments were done using different methods and parameters. These experiments have resulted in the following methods:

- An error threshold is chosen to determine anomaly candidates (single data points). A good choice for this is typically the 98th percentile of the training MAE loss
- An anomaly entry consists of at least 4 candidates with a maximum distance of 15 minutes of each other.
- The amplitude of candidates with the highest/lowest value needs to fulfill the following rule: $amp > 2 * stddev$ with stddev being the standard deviation of the corresponding candidates

Using these rules to determine an anomaly allows discarding most false positives while keeping the more heavily expressed ones.
Currently there are two ways to report an anomaly to the user that have been implemented. First is automatic generation of a graph displaying anomalies in the logged data by making the affected section red. Second is a Windows notification displaying

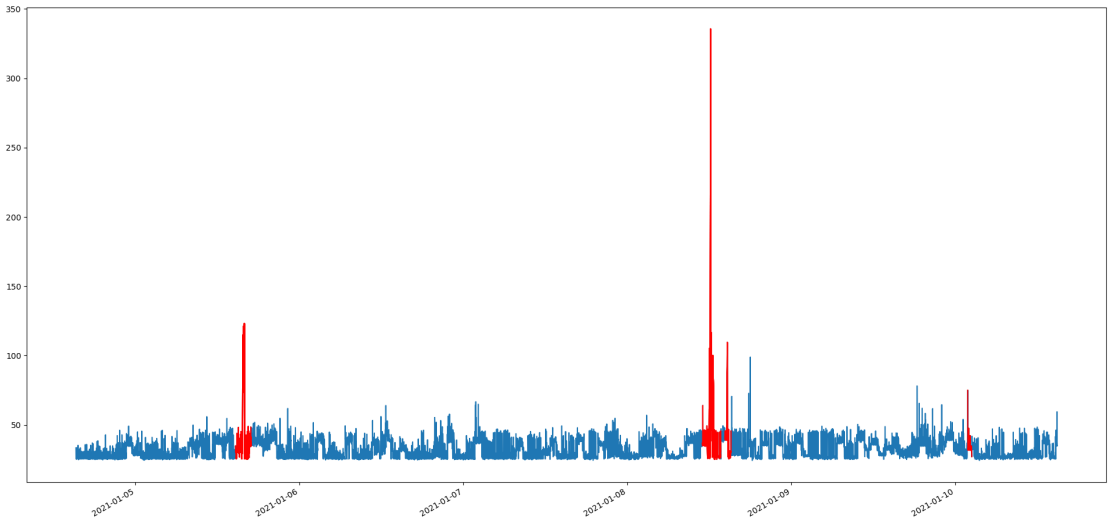

Figure 4.1: Anomalies in 1 week of ping data. Anomalies are flagged in red colour

the latest anomaly in a short text form. A user-interface was planned to allow a quick display of said graph and increased ease of use.

4.2 Testing

For testing purposes the data recorded for 1 week using a hybrid DSL/LTE connection is used (see figure 2.4). The expectation is that a high proportion of the QoS impacts discussed in section 2.3 will be recognized and thereby be flagged as anomalies. To make the testing resemble a real world application of the program, the sequence of 16 data points will be input as a sliding window over the last 16 recorded values. During testing this is achieved by sequentially feeding the recorded data into the live detection mode of the program. The results of the tests for each metric can be seen in figure 4.1, 4.2 and 4.3.

4.3 Results

In this section the test results of anomaly detection for each metric will be explained. When looking at the flagged anomalies for ping data (figure 4.1), one can see that most of the outliers are correctly detected as anomalies. Other than that the data is showing

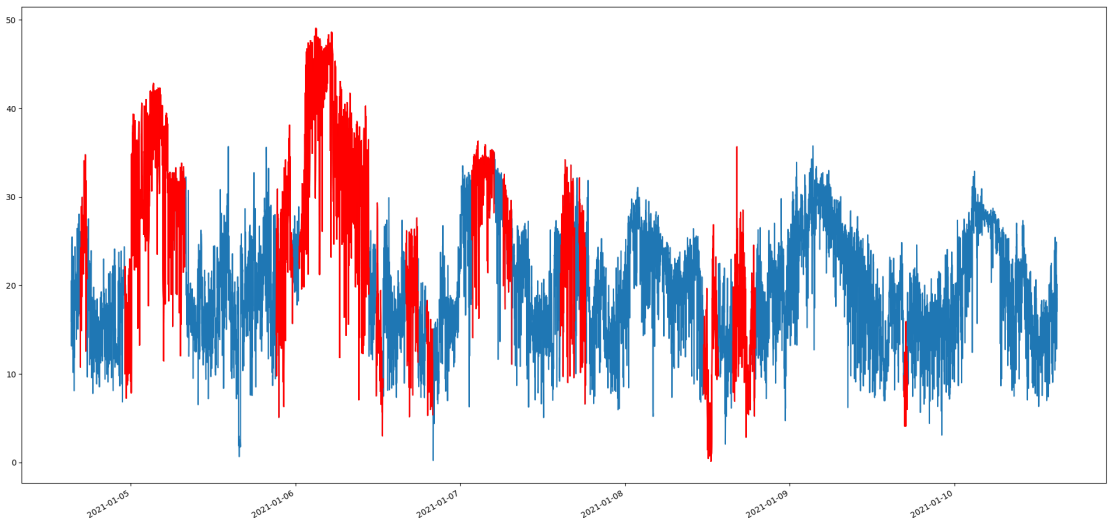

Figure 4.2: Anomalies in 1 week of download speed data. Anomalies are flagged in red colour

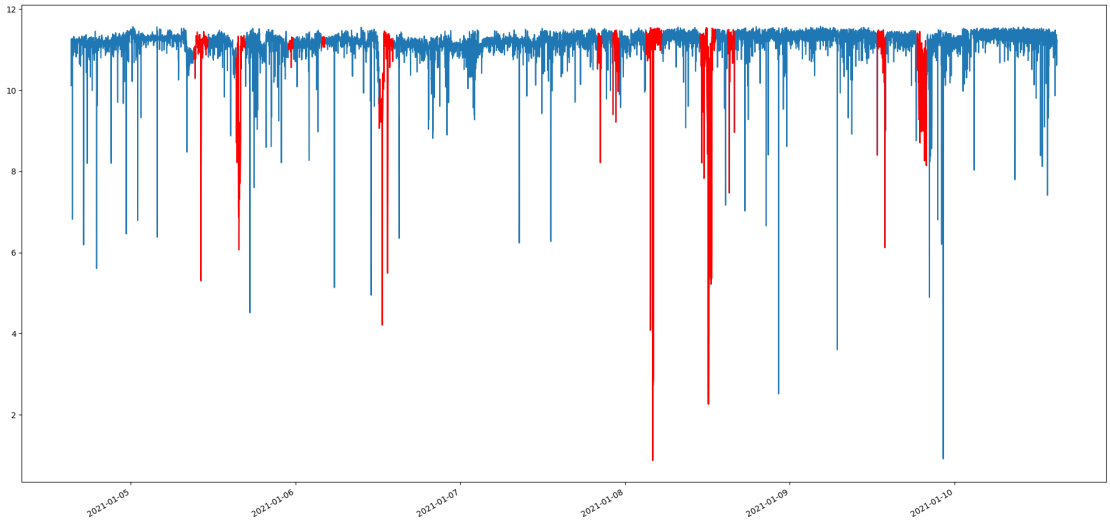

Figure 4.3: Anomalies in 1 week of upload speed data. Anomalies are flagged in red colour

no clear pattern that's occurring in a cyclic manner, so it is not detecting any violation of such patterns. There are some spikes above 60ms in the series that are not flagged as an anomaly around the 9th and 10th of January.
The anomalies that are flagged in the download rate (figure 4.2) appear to be mostly of the pattern-breaking type. As discussed in section 2.3, the download rate is heavily fluctuating on a regular basis but is showing a clear 24h cycle most of the time. Since it was trained on the steady form of such cycles it is working as expected, when looking at the irregular patterns occurring especially during the first half of the week. It's also apparent that spikes into the lower download rates below 5MBit/s are not always detected as anomalies.
Concerning the upload rate shown in figure 4.3 it appears that flagged anomalies are indeed mostly abnormal behaviour. One thing to note on the contrary is that there are 2 false positives detected around the 6th of January. Other than that there are many outliers that are not detected. More than half of the spikes below 8MBit/s are going unnoticed, resulting in a false negative rate above 50%.

4.4 Evaluation

The effectiveness of the autoencoder model for this particular application is dependant on what the main goal is. With the original goal of making end-users attentive to their internet connection quality it is acceptable to catch some false positives or miss some anomalies. Nevertheless better detection rates should be targeted in general. The experiment has shown that out of all three metrics each one has a unique pattern.
It's become apparent that ping is the easiest metric to analyze using this technique since it is following a very stable pattern when seen into a time series.
After that there's upload speed, which is also expressing a clear pattern, even though it contains more noise than ping. The tests have shown that there are many downward spikes that are missed during detection. It could be possible to catch these spikes with further tuning, especially due to the one-sided nature of the outliers.
Looking at the download speed metric it's getting difficult to distinguish what would count as an anomaly. Hence it's also important to consider what patterns the model is trained for to be seen as normal behaviour.
As for generalization there's been done short testing on another connection that's using fiber. The patterns each metric expresses in the training dataset are also showing up on that connection's record. It can be expected to be like that on most connection types,

but it may not be assumed as a general rule.
To further improve the reliability of the network and the surrounding algorithm there are two things to watch out for: First, the model should be trained on more datasets including different types of connections and ISPs to see if there's an improvement to generalization to be gained. Second, the way that anomalies are flagged (see rules in section 4.1) can gain from further experiments, since it is just one of many possible ways to interpret the error of prediction.

5 Discussion & Conclusion

The original question that was to be answered during this project was whether or not it's possible to recognize quality impacts of end-user internet connections using an autoencoder model as well as notifying the user about it. The tests have shown that the model can indeed analyze the metrics to find such impacts. Though for actual deployment of such a solution there is still work needed to improve the false positive and false negative rates as the scenario in which it was trained and tested is quite narrow, as it only consists of one connection type. As with many Machine-Learning experiments the lack of real-world data is the biggest hindrance in making the model work flawless in many different scenarios. Despite that there are still ways to tune the application as it is to be more reliable (see section 4.4), that mainly consist of putting more time into experimenting using different parameters or even methods.
There are a few methods that were thought of but couldn't be realized due to the limited dimension of the project:

- Application of filtering techniques for preprocessing input data. Although the usage of convolutional layers provide inherent filtering this could potentially improve detection on noisy data like the download speed rate (e.g. a Kalman filter). The downside of most filters is that they aren't stateless and thus require alternation of the dataflow or introduce lag to the analysis.

- Tunable parameters for the user. By allowing the user to set an error threshold that is a better fit to their respective connection behaviour an improvement to the detection rates could be made.

- Analysis of the prediction error as a time-series by itself. Currently an anomaly is recognized by looking at the number of prediction errors reaching a certain threshold. Due to the scalar nature of the error it could be analyzed using moving averages or filters (like mentioned above).

- Extend the model to use a combination of CNN and LSTM architecture. Research comparing bare CNN and a combination with other architectures has proven this method to improve the prediction accuracy of time series (see [7]).
- All tuning could be diversified as there are mostly general rules applied in the current state. Since the patterns for each metric show much variety, the configuration of the algorithm could be specialized for each one as well.

As a conclusion it could be said that the results are providing some certainty that an application with this goal can be realized using an autoencoder. It only requires collecting more data and tuning, as well as a full-flegded user-interface in order to become a product. Unfortunately even then it won't be able to solve the actual anomalies occurring but it could help to improve awareness of the problem. This awareness could then turn into action, like an increased amount of complaints to the Internet Service Provider (ISP), which could then ensure that improvements are made. Another consequence could also be that users that gained certainty about the bad quality are now selecting other ISPs over their own for their needs.

Glossary

NN Neural Net
MAE Mean Absolute Error
CNN Convolutional Neural Net
Ping Latency
LSTM Long short-term memory
ISP Internet Service Provider
CPU Core Processing Unit
LTE Long Term Evolution, A wireless connection standard
DSL Digital Subscriber Line, a wired connection standard
Autoencoder Automatic Encoder

6 References

[1] : *Advertised vs. actual internet speeds*. Web. Feb 2021. – URL https://www.allconnect.com/blog/advertised-vs-actual-internet-speeds

[2] : *speedtest-cli very slow although network speed is fine*. Web. Feb 2021. – URL https://superuser.com/a/1519159

[3] Huang, L. ; Zhou, M. ; Wang, W.: A Passive Mode QoS Measurer for ISP. In: *2009 WRI World Congress on Software Engineering* Bd. 1, 2009, S. 308–313

[4] Liberal, F. ; Ferro, A. ; Fajardo, J. O.: What Quality Means for Internet Users: A Guide to Selecting your ISP. In: *International conference on Networking and Services (ICNS'06)*, 2006, S. 81–81

[5] sivel: *speedtest-cli*. GithubRepository. Feb 2021. – URL https://github.com/sivel/speedtest-cli

[6] Stafecka, A. ; Lizunovs, A. ; Bobrovs, V.: Mobile LTE network signal and Quality of Service parameter evaluation from end-user premises. In: *2018 Advances in Wireless and Optical Communications (RTUWO)*, 2018, S. 209–212

[7] Yin, C. ; Zhang, S. ; Wang, J. ; Xiong, N. N.: Anomaly Detection Based on Convolutional Recurrent Autoencoder for IoT Time Series. In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* (2020), S. 1–11