

MIDDLE EAST TECHNICAL UNIVERSITY  
DEPARTMENT OF ELECTRICAL AND ELECTRONICS  
ENGINEERING



KUARTIS

**EE400 SUMMER PRACTICE II**  
**SUMMER PRACTICE REPORT**  
**UNSUPERVISED ANOMALY DETECTION**

**GÜRAY ÖZGÜR**  
**2167054**

**SP COMPANY:** KUARTIS TECHNOLOGY AND CONSULTING

**COMPANY DIVISION:** ICT, SOFTWARE, DEFENCE

**SP BEGINNING DATE:** FEB. 2, 2021

**SP ENDING DATE:** MAR. 12, 2021

**SUPERVISOR:** İLAYDA BEYRELİ, [ilayda.beyreli@kuartis.com](mailto:ilayda.beyreli@kuartis.com)

# Contents

<b>1 INTRODUCTION</b>	<b>2</b>
<b>2 DESCRIPTION OF THE COMPANY</b>	<b>3</b>
2.1 Company Name . . . . .	3
2.2 Company Location . . . . .	3
2.3 Contact Information of the Company . . . . .	3
2.4 General Description of The Company . . . . .	3
<b>3 WORK CONDUCTED AT THE SP LOCATION</b>	<b>4</b>
3.1 Understanding the Problem . . . . .	4
3.2 General Concepts . . . . .	5
3.2.1 Anomaly Detection . . . . .	5
3.2.2 Unsupervised Learning . . . . .	6
3.2.3 Clustering . . . . .	7
3.2.4 HDBSCAN . . . . .	7
3.2.5 Variational Autoencoders (VAE) . . . . .	9
3.2.6 Generative Adversarial Networks (GAN) . . . . .	10
3.2.7 Temporal Dissimilarity Model . . . . .	11
3.2.8 Optimizer and Losses . . . . .	13
3.2.9 t-SNE . . . . .	14
3.3 Explanation of Work Done Step by Step . . . . .	15
<b>4 CONCLUSION</b>	<b>28</b>
<b>5 REFERENCES</b>	<b>29</b>
<b>6 APPENDICES</b>	<b>31</b>
A HDBSCAN Implementation . . . . .	31
B t-SNE Implementation . . . . .	31
C Waterfall Image Implementation . . . . .	32
D Testing VAE . . . . .	34
E Mel-Spectrogram Generation . . . . .	34
F infoGAN Implementation . . . . .	37

## 1 INTRODUCTION

I have performed my second internship at Kuartis Technology and Consulting, which is a company founded in 2012 by Ahmet Saracoğlu and Serdar Gedik. The reason why I chose Kuartis is that I was interested in machine learning and Kuartis could give that opportunity to me at their Computer Vision and Machine Learning Department. In their own words, Kuartis is an innovative machine learning company, which I was looking for. The company can be regarded as a small company and used to be a start-up, however, having a small population should not imply that it does not have dedicated engineers and computer scientists working in collaboration and ambition and I had a chance to work with those great people for six weeks, from 02.02.2021 to 12.03.2021. During my internship period, my supervisor was İlayda Beyreli, who has a B.Sc. degree in Electrical and Electronics Engineering from METU and a M.Sc degree in Computer Science from Bilkent University. She is currently enrolled as a PhD student at Bilkent University and working at Kuartis as a full-time machine learning engineer.

My task at CVML department of Kuartis was basically to help my supervisor at the project at hand, by testing the model, analyzing, and interpreting the results of tests, also trying to develop a model, and understanding and improving the existing one. The problem can be explained as "Unsupervised Anomaly Detection" with a few words, which will be explained in detail. During my internship, I needed to use tools that are generally used in machine learning tasks in Python. Specifically, I made use of pandas, matplotlib and PIL for data analytics and visualization, and frameworks for machine learning and neural network modelling such as NumPy, scikit-learn, PyTorch and TensorFlow.

In this report, there are several sections about my internship period and my observations, which begins with some information about the company in Section 2. Then, general concepts about machine learning with their comprehensive descriptions are discussed in Section 3.2, work conducted in SP company is explained with detailed information in Section 3.3. Various figures, schematics about the task are included for clarifications.

## 2 DESCRIPTION OF THE COMPANY

### 2.1 Company Name



### 2.2 Company Location

Silicon Building, No:17/101, METU Technopolis, 06800 Cankaya/Ankara

### 2.3 Contact Information of the Company

Telephone / Mail: +90 312 241 4045 / info@kuartis.com

### 2.4 General Description of The Company

Established in 2012, Kuartis is a leading machine learning company. The company constructs both the hardware, and its supporting framework that power huge smart solutions. In this way, they develop data-driven systems to discover and understand the meaning in the data and use the data to take actions. Since establishing, Kuartis enlarged to more than 40 individuals including very experienced software and hardware engineers, which enables the company design and produce their own devices from scratch. Since the company is located in Technopolis area of METU, naturally, it is affected by the research environment of the university. That's why, R&D is the unit where most of the money is spent and earned. The company serves these end products mostly to companies in the defense and automotive industries. Furthermore, the company wants students to expand their knowledge in the field and join the company afterwards, that's why it hosts several university students who are eager to learn, and helps them to improve their knowledge about the field. As one of the students, I had a chance to see the environment in the Computer Vision and Machine Learning Department among senior engineers, shown in Figure 1.

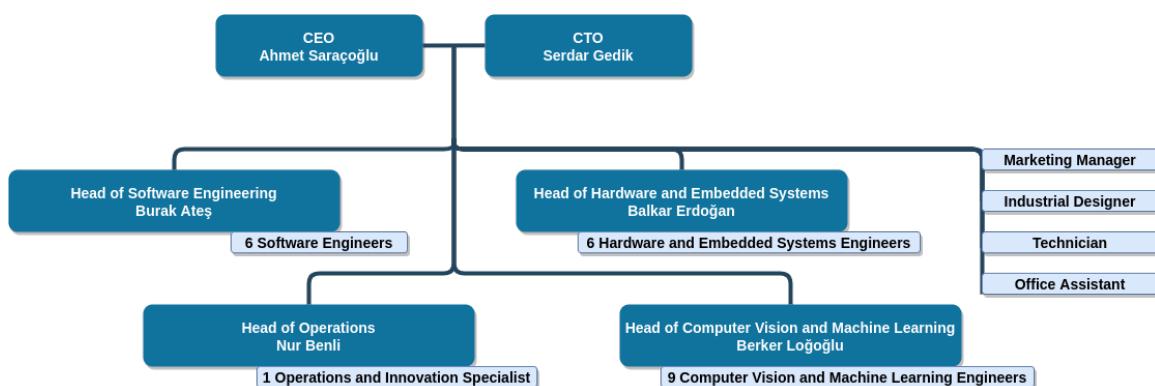


Figure 1: Administrative structure of Kuartis

### 3 WORK CONDUCTED AT THE SP LOCATION

#### 3.1 Understanding the Problem

The transportation and distribution of oil and natural gas has a crucial importance in the oil and gas industry. In addition to effectiveness of the transportation and distribution system, it is also very important to ensure maintenance and safety of the pipeline. Otherwise, companies may experience unexpected economic losses from both losses of the resources and interruption of the consumption, i.e. operational losses. The latter is also unwanted by the consumers, which may result in inconveniences in many ways. Thus, protection of the pipeline and opposing the potentially dangerous events at the right time is what needs to be done. For this reason, both researchers and companies have been more interested in research on pipeline surveillance systems and pipeline security in recent years [1–5].

In the literature, there are several ways to deal with this problem. Main methods include a Distributed Acoustic Sensing (DAS), to detect, and a Pattern Recognition System (PRS), to classify the possible threats along the pipeline, which constitutes a surveillance around the pipeline [1]. Distributed Acoustic Sensing is generally achieved by distributed acoustic sensors, an optical fiber line following the pipeline, to observe the activity in the ground and detect unusual activities threatening the integrity of the pipeline. For further information about acoustic sensing, optical time domain reflectometry (OTDR) and optical frequency domain reflectometry (OFDR) methods can be examined [2]. Pattern Recognition System is a system whose aim is to understand the system and decide whether the activity observed by DAS is a threat or not, which is a whole other research topic since employing the acoustic data and trying to find meaning, i.e. modelling, can be done in so many ways [4].

The illustration of the problem can be seen in Figure 2, which is prepared by Samm Technology, for further understanding the source of the problem [?].

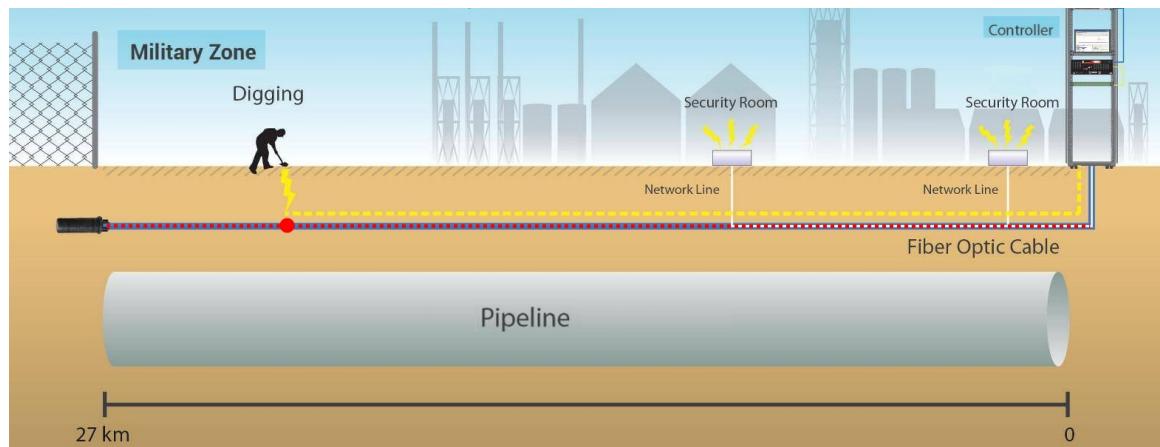


Figure 2: Illustration of pipeline security problem

Our part as Kuartis is to develop a pattern recognition system to identify the threats to prevent economic and operational losses, which will inform the authorities to intervene with the threats. Since the project was rather new, there was a need for deep research, and concrete ideas with strong background for implementing the system. To this end, it is better to explain the concepts before presenting the results.

### 3.2 General Concepts

#### 3.2.1 Anomaly Detection

Anomaly detection is a procedure of searching and finding the data points which are different from the majority of the data points, normal of the dataset. These data points are called anomalies or outliers. Anomalies are unusual data points, normally not expected in the dataset. However, existence of anomalies is not meaningless, it may lead to different interpretations or indicate problems in the system. In our case, anomalies are unexpected events, or threats to our pipeline. In other words, anomaly detection is threat identification in our problem. Detection of these anomalies is the main problem. There are three types of anomalies which are shown in Figure 3. These are namely point anomalies, collective anomalies and contextual anomalies [6].

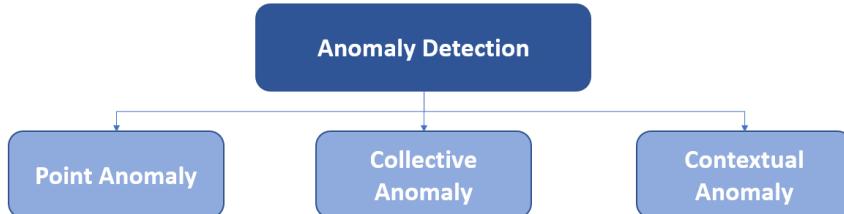


Figure 3: Types of anomalies

Point anomalies lies outside of the dataset, that's why they can be separated easily from the dataset. An explosion near the pipeline can be an example of a point anomaly. Collective anomalies arise when a collection of normal activities indicates an abnormal activity. Walking around pipeline may not be abnormal, however, walking activity in every day at the same time can be a sign of a threat. Contextual anomalies depend not only measurement conditions but also the context. Construction equipment can be example of a contextual anomaly. Construction of a subway station knowing there is a pipeline nearby may not be an anomaly or threat. As can be seen, in our pipeline security problem, we have all types of anomalies, especially point and contextual anomalies.

### 3.2.2 Unsupervised Learning

Anomaly detection can be done in several ways with machine learning, which is shown in Figure 4. It highly depends on the structure of the dataset, especially whether the dataset includes labels or not. Normally, it is usually preferred to have labels and work on supervised learning tasks. Supervised learning is done by training the model with both data points and their labels. However, if labels can not be provided for reasons such as unavailability, cost of labelling, or even confidentiality, semi-supervised or unsupervised learning can be a choice. Semi-supervised learning in anomaly detection uses only normal of the dataset while learning, separation of the abnormal points is done in a pre-processing step. Both supervised and semi-supervised methods give a result at the end of the procedure, which is abnormal or normal. On the other hand, in unsupervised learning, which does not use the labels at all, the similar points are collected together, or clustered, in a way by comparing the points with each other, and at the end, a probability is given for the data point indicating its abnormality. Then, the data point, whose probability of being an abnormal (outlier score) is available, can be classified with a properly selected threshold value [7].

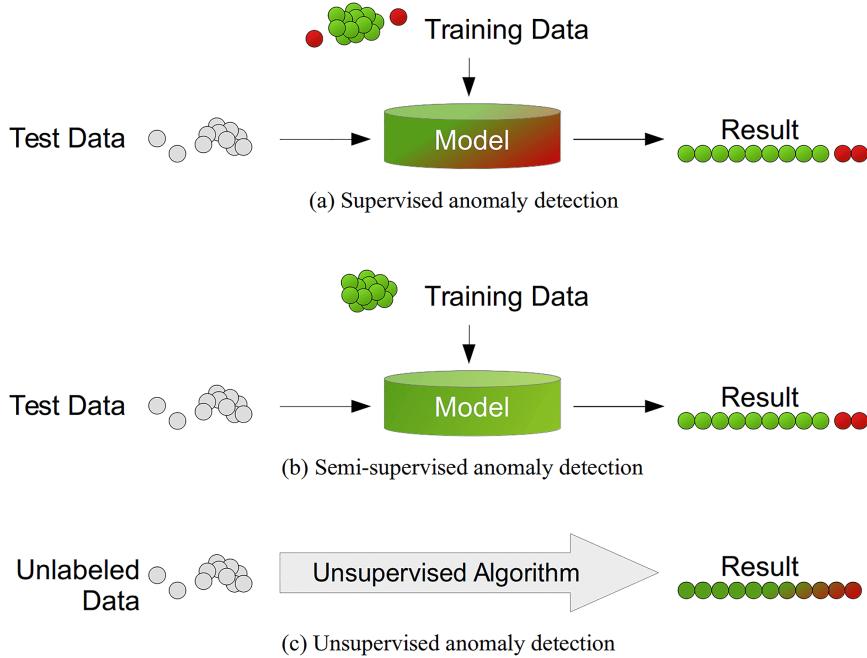


Figure 4: Different anomaly detection methods

### 3.2.3 Clustering

Before defining clustering, the importance of the clustering can be emphasized with an example. People, even while doing simple tasks, utilize clustering algorithms in their brain, and are very good at it. We can distinguish and separate different items without explicitly thinking about it. For example, even a very small child know the difference between a red apple and a green apple without knowing the colors [8]. To people who are willing to try and appreciate clustering more, apples are shown in Figure 5.

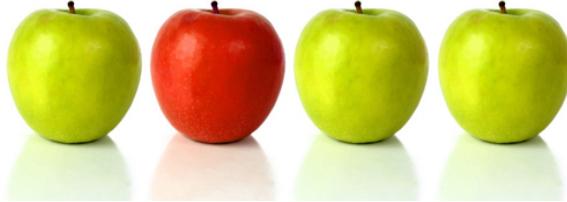


Figure 5: Clustering in apples

This task, which is a mystery to me that people do it so easily, is called clustering (grouping). Clustering is the task of understanding the data by finding similarities within data, grouping similar data points, and searching the meaning of those groups. It can be used as classification in unsupervised learning to assign the objects to clusters, and then determine which clusters corresponds to which classes. In this way, it is possible to understand of the meaning of the data. In our application, clustering of the data can mean that different clusters correspond to different activities: walking, digging with pickaxe, digging with shovel, strong wind, etc. Or clustering can result in only one cluster, which is noise, and the other activities can be seen as outliers (the points outside clusters). So, it is a very important tool in exploratory data analysis. There are different type of clusters and clustering algorithms. Cluster can be separated to five types as well-separated, center-based, contiguity-based, density-based, and conceptual clusters. K-Means, DBSCAN, and HDBSCAN, which will be explained in Section 3.2.4, can be examples of clustering algorithms. The aim of each algorithm as mentioned is to collect similar points together by using a distance function. Distance function defines the similarity of the data points. If they are close to each other, they are similar and vice versa.

### 3.2.4 HDBSCAN

Now, we want to cluster our data. Main question is that which algorithm should be chosen for clustering. By using our prior knowledge about the data, it can be concluded that there is noise and there may be possible clusters to discover for different events. Using K-Means can be an option, however, K-Means requires to know the number of the clusters since it tries to separate the data to K clusters, which we do not know. Furthermore, K-Means is robust when the clusters are equally-sized, spherical and density is homogeneous, which is

again is an unknown. The data can be in different densities, arbitrary shapes and different sizes as shown in Figure 6. HDBSCAN is a density-based algorithm developed for those challenges, and it performs way better than other clustering algorithms with an acceptable speed, shown in Figure 7. That's why, HDBSCAN is chosen by my supervisor to cluster the features extracted from a variational autoencoder, which is explained in Section 3.2.5.



Figure 6: Challenges while clustering

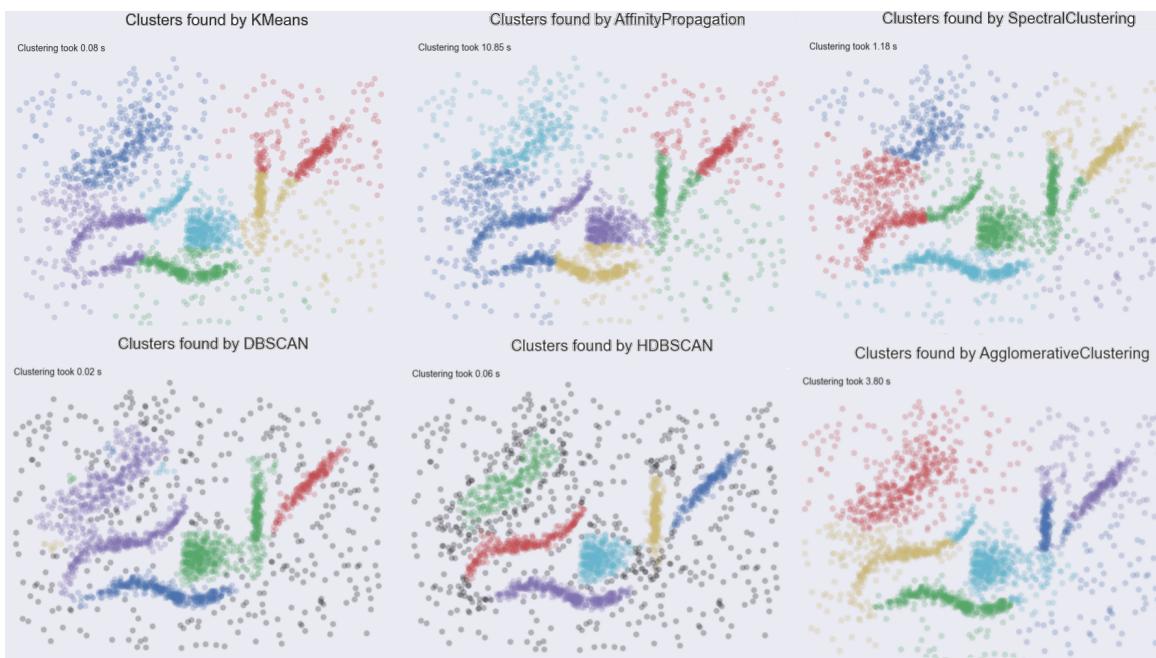


Figure 7: Comparison of different clustering algorithms

There are two main parameters to choose for HDBSCAN. Clusters can be regarded as the peak regions of the probability distribution of the data, since the data is dense at that region. Thus, to determine how much to take from that region, there is a parameter called minimum cluster size. Minimum cluster size is chosen such that more probable, i.e. more dense, areas are well-separated from the less probable, i.e. sparse, areas. There is also a parameter, which has an effect in estimation of the probability distribution function, called minimum samples. Increasing minimum samples smoothen the PDF, so there is also an optimum value for minimum samples, which will be used to cluster our data [9].

### 3.2.5 Variational Autoencoders (VAE)

Every data has its own properties, features, sometimes the number of features are so high, and some features are basically more important than the others. It may be unnecessary to use all of the features to understand the data. For example, a blurred photo can give enough information to understand what it is. There are some reasons such as computational complexity, data storage and data visualization why feature reduction, or dimensionality reduction is needed. It can be done by simply selecting some of the features that are important, or by extracting new features from the existing ones. What autoencoders do is to reduce the number features, i.e. encode, and try to obtain the initial features from extracted features, i.e. decode with the help of a neural network architecture [10]. The goal is to represent the features as much as possible after reduction and reconstruct the same features with minimum error, which can be given as a reconstruction loss in Equation 1.

$$loss = \|x - \hat{x}\|^2 = \|x - d(z)\|^2 = \|x - d(e(x))\|^2 \quad (1)$$

In this way, after ensuring the reconstruction is well-done, features extracted after encoding can be used in clustering and classification of anomalies can be achieved. However, there are two problems with autoencoders, they tend to overfit (fit only the training dataset) and for random points from the latent space, space of all encodings, decoding does not have a meaning. To enhance the results, instead of using a simple autoencoder, a variational autoencoder can be employed, since we may observe variations of our data in real life. Thus, we take samples from the latent space to train the decoder, which is done by adding a regularisation loss, KL divergence, which will be explained in Section 3.2.8 with other loss functions. The overall loss is given in Equation 2. Illustration of variational autoencoders are shown in Figure 8.

$$loss = C \|x - \hat{x}'\|^2 + KL[N(\mu_x, \sigma_x), N(0, I)] = C \|x - f(z)\|^2 + KL[N(g(x), h(x)), N(0, I)] \quad (2)$$

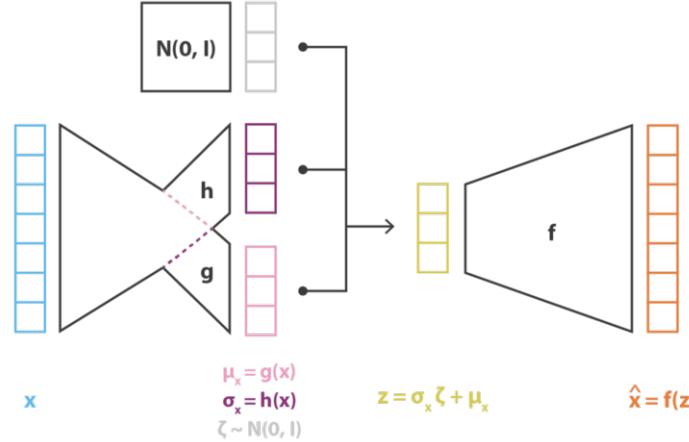


Figure 8: Illustration of Model for Variational Autoencoder

### 3.2.6 Generative Adversarial Networks (GAN)

To understand what generative adversarial networks are, it may be a good idea to start with the name. Generative implies that there is a generation, or creation, thus a generator neural network, which generates a data from the noise. Adversarial implies that there is an opposition, thus an opposing network to generator, called discriminator. We can imagine two networks playing a game with each other, we want generator to give a generated (fake) data to discriminator, and we give real data to discriminator. We want discriminator to guess which is the real data. By punishing generator and discriminator, generator learns how to generate real-like data and discriminator learns to distinguish the real data from the fake one. It is done by maximizing final classification error for generator and minimizing final classification error for discriminator [11]. At the end, it is a min-max game for the networks, which is shown in Equation 3. GAN structure is shown in Figure 9, red boxes are not a part of GAN structure but a part of infoGAN structure.

$$\text{loss} = \min_G \max_D V(D, G) = \underbrace{E_{x \sim p_{\text{data}}(x)}[\log D(x)]}_{\text{real}} + \underbrace{E_{z \sim p_z(z)}[\log(1 - D(G(z)))]}_{\text{fake}} \quad (3)$$

As in variational autoencoder case, an additional regularization term is what is needed for a better performance as indicated in Equation 4. In this case, this regularization term maximizes the mutual information between the generated output and newly added latent code. Latent code brings two advantages to GAN. One is that variations are possible as in VAE, second is that we have a control over what to generate, which was not possible with

GAN. Additional structures added to infoGAN structure is shown in red in Figure 9.

$$loss = \min_{G,Q} \max_D V_{infoGAN}(D, G, Q) = V(D, G) - \lambda L_I(G, Q) \quad (4)$$

Here, the advantage of using a GAN is to use discriminator to distinguish the normal data and abnormal data, since what discriminator essentially does is to cluster similar data points. Disadvantage might be the need of the number of clusters, which must be given beforehand. By arranging the data to normal and abnormal classes accordingly, this problem can be solved.

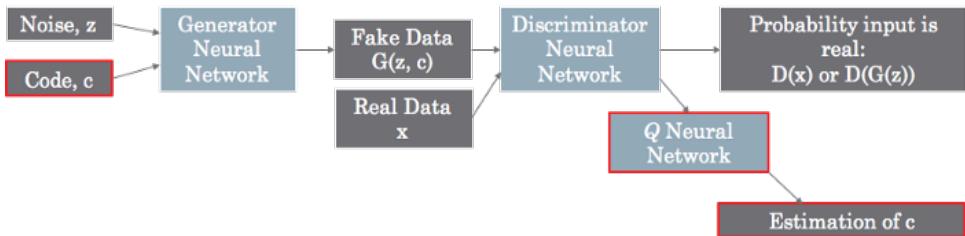


Figure 9: Illustration of Model for infoGAN

### 3.2.7 Temporal Dissimilarity Model

Now, before going into details of employing 1D data for our models, VAE, or infoGAN in this case, there is another paper I want to mention about. This paper is especially important for our problem, since it propose a solution to a similar problem. Paper is titled as "Detection of Anomaly Acoustic Scenes based on a Temporal Dissimilarity Model" authored by Tatsuya Komatsu and Reishi Kondo [12].

In this model, a one-day acoustic signal is first separated to segments and then the segments is further separated to frames. The reason why a one-day signal is examined is that there is an assumption of periodicity of one day, which is reasonable in some situations like a train station. Then a Gaussian Mixture Model is used for estimating PDF of segment by using frame-level features. Current segment and previous segment are compared by a divergence of GMMs, and a temporal dissimilarity is calculated. In this way, if we have an unexpected peak at the moment, we can be aware of the situation. However, as in our problem, every peak does not mean that it is an abnormal situations. Maybe, whistle of the first or the last train is heard in a train station just like every day. Their assumption of periodicity is activated at this stage. It also compares with the stored data from yesterday, and checks whether it deviates too much from it or not. Afterwards, the model gives an anomaly score. Figures 10, 11 and 12 are taken from the paper, and illustrates the paper [12]. As seen in Figure 12, there are blows of whistle of the first and the last train at the beginning and end of the working hours, which are normal. However, activities such as irregular cleanings, party events are decided as anomalies since they do not occur normally.

Our data is also an acoustic data collected from different fixed locations, i.e. the location of each sensor is fixed. Activity of each sensor can be assumed as periodic by noting that the difference between weekdays and weekends may be important and a periodicity of one-week can be preferred. Furthermore, by realizing that our data is not only temporal but also spatial, an additional spatial dissimilarity can be added. For this reason, this paper holds an important role for both solution and the inspiration of the solution.

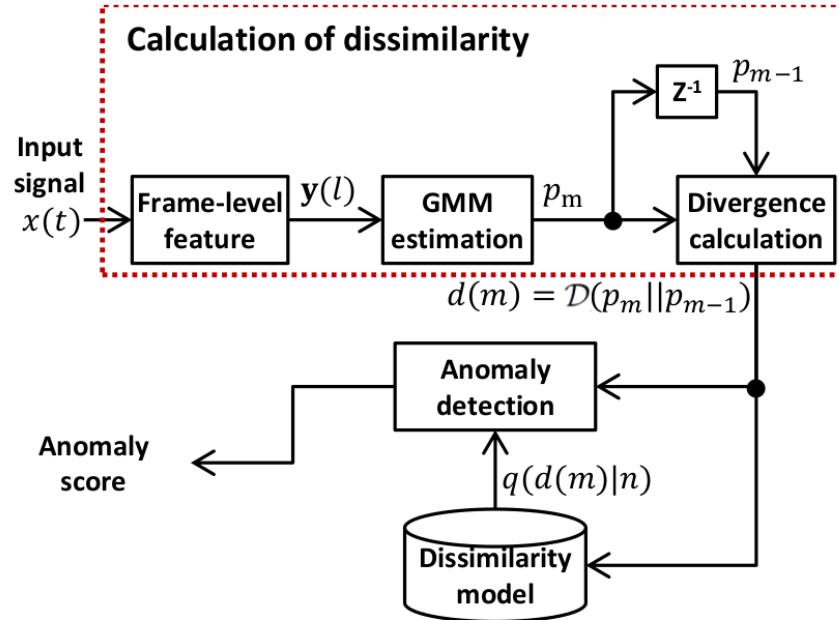


Figure 10: Block diagram of their proposal

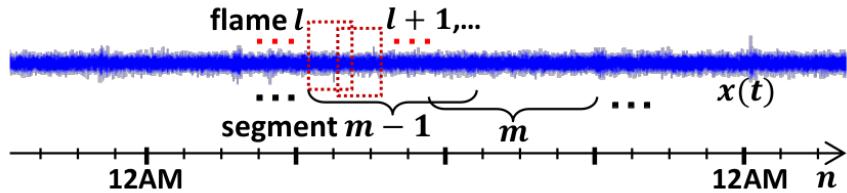


Figure 11: Relationships among temporal indices  $l, m, n$

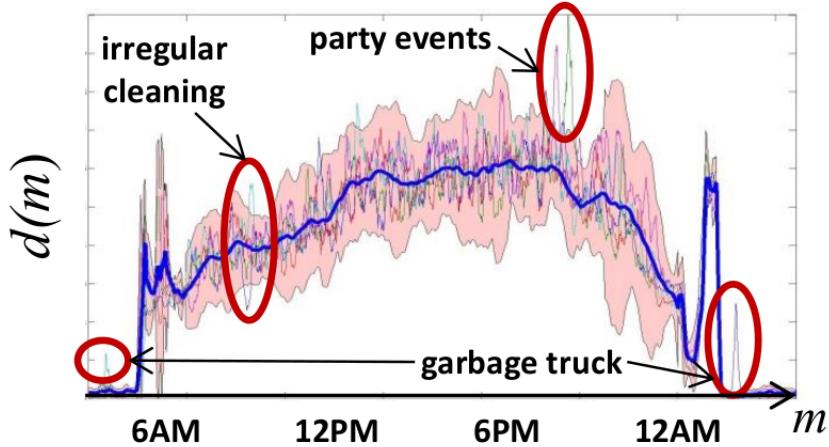


Figure 12: Temporal variations in their findings

### 3.2.8 Optimizer and Losses

#### Optimizer

- **Adam Optimization**

Adam is a method for stochastic optimization proposed by Diederik P. Kingma, and Jimmy Ba in 2014 [13]. It is very fast, however, it is computationally costly.

#### Losses

- **Mean Square Error Loss**

MSE Loss measures the squared L2 norm between the input and the output for each element. Squaring results in higher penalisation on predictions which deviates a lot from actual values [14]. Its equation is given in Equation 5.

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2 \quad (5)$$

- **Cross-Entropy Loss**

Cross-Entropy Loss is used in classification problems for known number of classes. If confident predictions are wrong, they are penalized heavily [14]. Its equation is given in Equation 6.

$$\text{CE} = -(y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)) \quad (6)$$

- **Binary Cross-Entropy Loss**

It is used when the error of a reconstruction is needed to measure as in autoencoders.

The name comes from the output being between 0 and 1 [14]. Its equation is given in Equation 7.

$$\text{BCE} = -\frac{1}{N} \sum_{i=1}^N (y_i \log(p(y_i)) + (1 - y_i) \log(1 - p(y_i))) \quad (7)$$

- **Kullback-Leibler Divergence Loss**

Kullback-Leibler Divergence Loss, in a way, measures the difference between two probability distributions, hence it is also called relative entropy [14]. Its equation is given in Equation 8.

$$\text{KL} = \sum_{c=1}^C q(y_c) (\log(q(y_c)) - \log(p(y_c))) \quad (8)$$

While infoGAN uses MSE and Cross-Entropy, VAE uses Binary Cross-Entropy Loss and KL Divergence. Temporal dissimilarity also uses KL Divergence for divergence calculation. All optimizations are done with Adam Optimization.

### 3.2.9 t-SNE

t-SNE is an exploration tool for high-dimensional data, it is a dimensionality reduction algorithm to reduce the dimension that we can visualize, 2D and 3D. It essentially maps a point in higher dimensions to 2 dimensions or 3 dimensions. The way of mapping tries to preserve distances in higher dimensions while reducing the dimensionality. Distant points are tried to be mapped distant, and closer points are tried to be mapped closer by preserving clusters in a way. The comparison with other algorithms reducing dimensions are presented in Figure 13, which shows the superiority of t-SNE clearly [15].

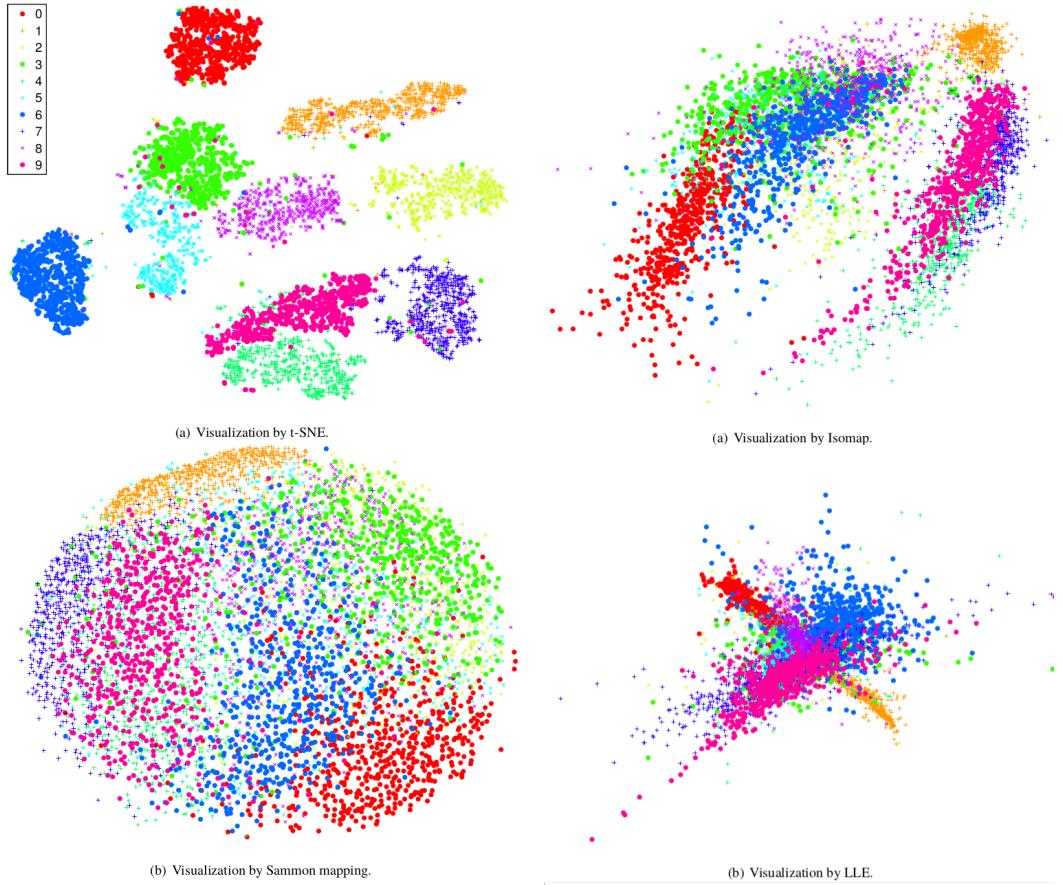


Figure 13: Different dimension-reduction algorithms for MNIST dataset

### 3.3 Explanation of Work Done Step by Step

To understand the concepts, I was given papers and book chapters to read, which I used in the project and summarized the concepts in Section 3.2.

First and foremost, since I was using my own laptop, I had to install several libraries to run the codes. For this, I have installed libraries such as matplotlib, NumPy, PyTorch and TensorFlow to my anaconda environment. Moreover, I have installed NVIDIA-cuda to use my GPU while training, to speed up the process. After installing necessary libraries considering the version not to conflict, and setting up the environment, variational autoencoder model was handed to me, and I was expected to cluster the extracted features from the encoder of VAE. I read the document about HDBSCAN, which I have explained in Section 3.2.4, and implemented the clustering.

It is a hard task to measure the performance when there is no ground truth. I have searched for measures of clustering performance.

- **Silhouette Coefficient:** It is between -1 and 1. Higher means better. It divides the mean of intra-cluster distance to the mean of inter-cluster distance, where intra-cluster distance is the distance of elements of a cluster and inter-cluster distance is the distance between different clusters. A score of 1 means dense clusters are separated from each other.
- **Calinski-Harabasz Index:** Higher means better. It divides a number representing separation between different clusters to a number representing separation of elements of clusters. It is not bounded, so a very high number is expected for well-clustering.
- **Davies-Bouldin Index:** It is non-negative a lower value indicate a better performance. It measures the similarity between clusters, and for well performance no clusters should be similar to each other. To illustrate, for two clusters, this is done by dividing the addition of cluster diameters to distance between the centroids of clusters. For more than two clusters, averaging is used.

In this stage, approximately a Silhouette Coefficient of 0.8, a Calinski-Harabasz Index of 1500, and Davies-Bouldin Index of 0.1 are obtained, which indicated that clustering was well-done.

After establishing the cluster performance, the question is that can the model find anomalies or outliers. Analysis showed that a huge cluster and outliers are detected. Here, the problem is that visualization of the clustering is impossible by considering the dimensions of the extracted features. A visualization is needed to better understand the what is happening on clusters. This visualization can be achieved by a dimension reduction algorithm such as t-SNE. By marking the outliers and showing by using a different color in t-SNE is a good way to understand what is happening, which is done and shown in Figures 15 and 16. Since HDBSCAN gives a probability of being anomaly, or outlier score, there is a need for a threshold to classify as an outlier. For this, distribution of the data is obtained in histogram plot, a maximum likelihood Gaussian distribution is fitted and a threshold is decided accordingly, as seen in Figure 14.

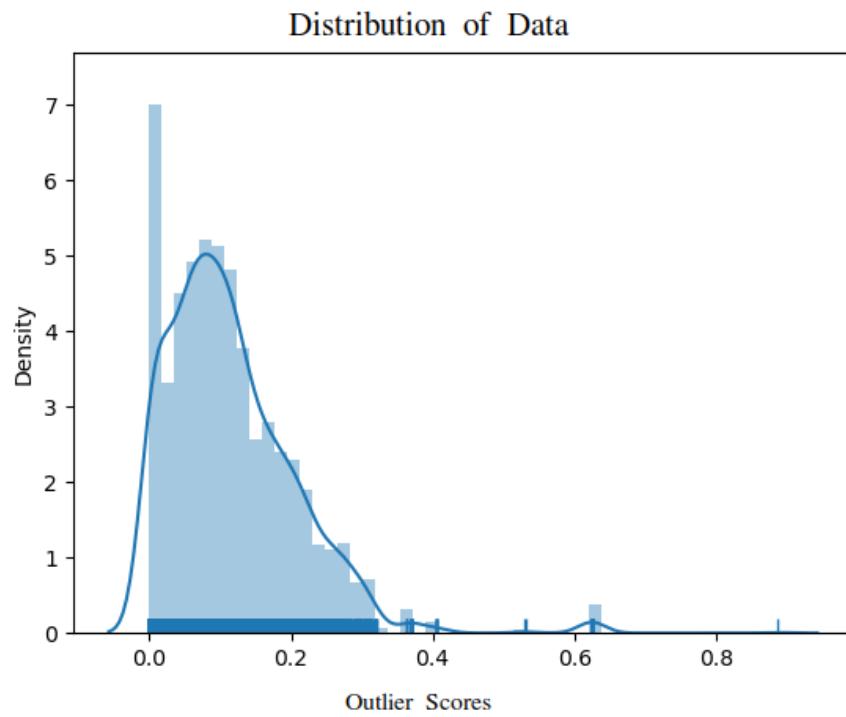


Figure 14: Distribution with a histogram and maximum likelihood Gaussian fit of the data

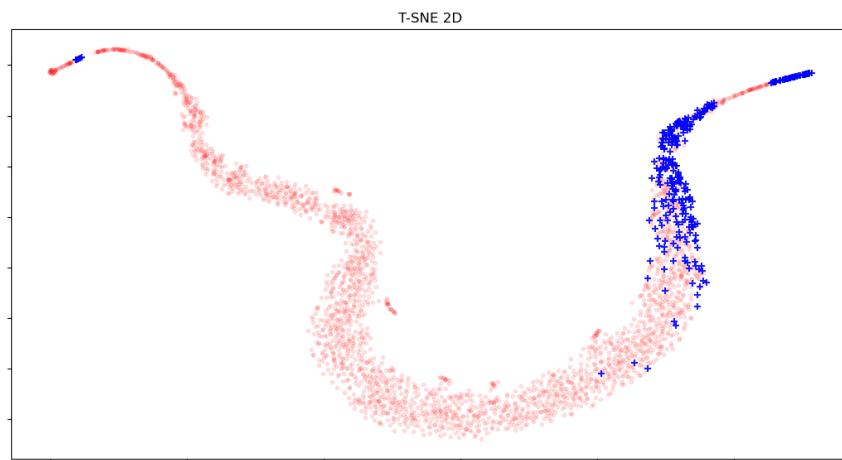


Figure 15: Two dimensional t-SNE visualisation of the dataset with outliers marked as blue

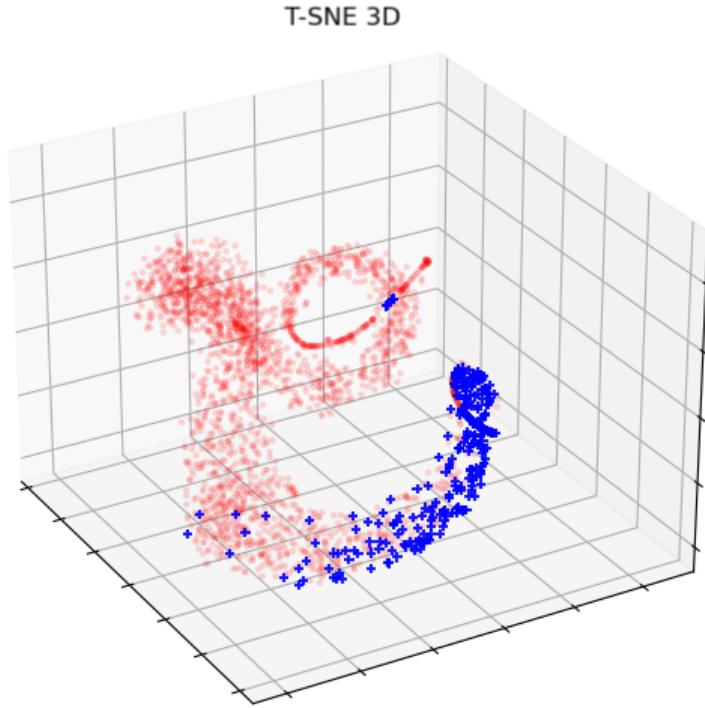


Figure 16: Three dimensional t-SNE visualisation of the dataset with outliers marked as blue

Now, it is time to comment on the results, here we have found a different shape cluster, which we think it is the normal of our data. Furthermore, an accumulation of outlier marked as blue is seen at the right of Figure 15. It may be possible that outliers can represent the same activity, which may be digging. These are, of course, wild guesses. Although we have a nice clustering, it may not represent what is expected.

Here, what is done is to compare it with something, which is our contractors' findings. The reason why we are given the project is that our contractor has quite a few false positives, which they do not want. Thus, this comparison does not give much information, however, it is worth of trying to see what happens, which I did.

Again, to visualize I have used t-SNE and marked the findings with different colors and stated the legend. Since the name of the contractor is confidential, I blacked out the name. Findings can be found in Figures 17 and 18, which are presented to the contractor in external meetings.

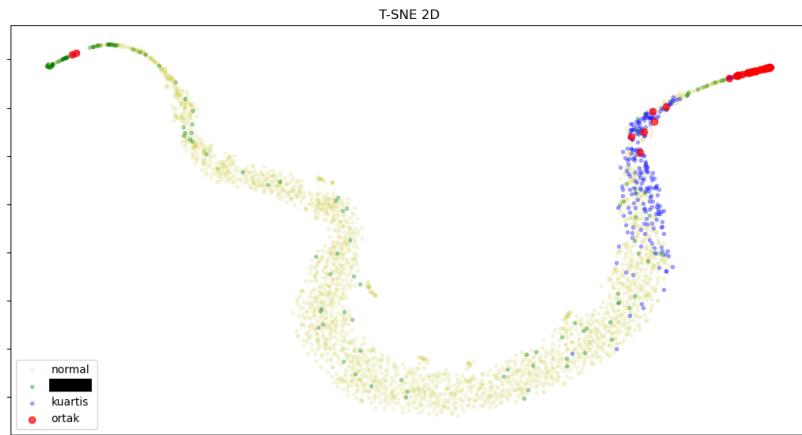


Figure 17: Two dimensional t-SNE visualisation of the dataset including contractors' decisions for comparison

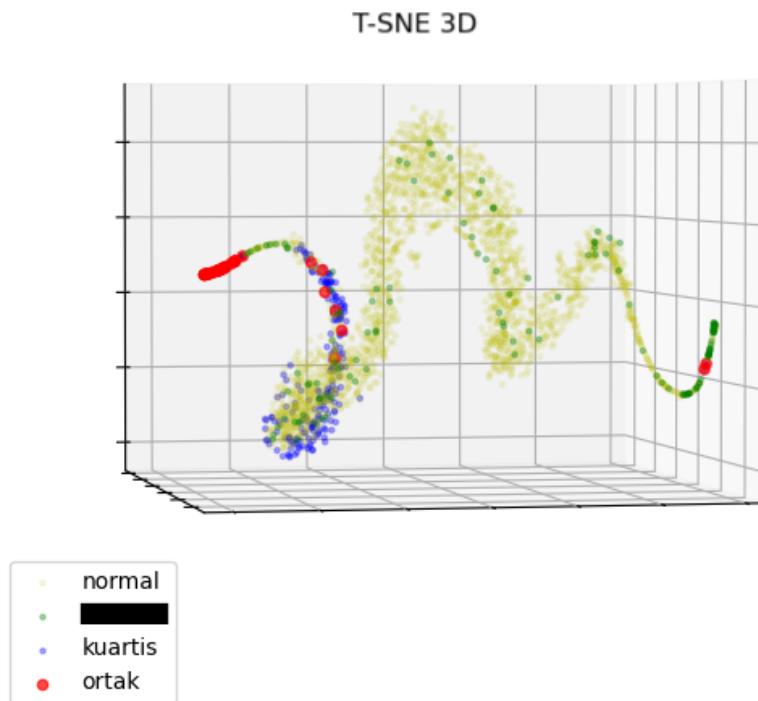


Figure 18: Three dimensional t-SNE visualisation of the dataset including contractors' decisions for comparison

Note that there is a substantial difference as seen in Figure 17, which indicated that features extracted from VAE does not match the contractors' features. Seeing blue points, representing the data considered anomaly by only us, was unexpected, and was a sign of existence of a mistake. To further investigate the findings, I was asked to create a waterfall image that includes the findings of both parties, as requested from the contractor.

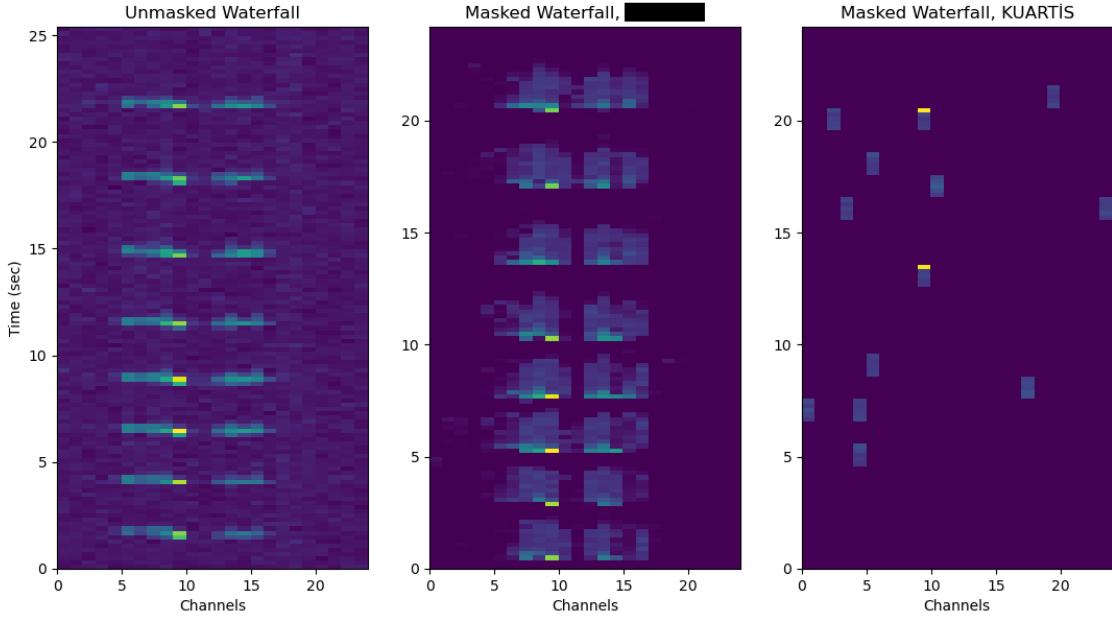


Figure 19: Comparison of the predicted anomaly events between of contractor and Kuartis

In Figure 19, the waterfall images are shown. In unmasked waterfall image, it is clearly seen at channel 9, there is an activity repeatedly, and its effect around other channels are observed. The contractors' probabilities are used to mask this image and the second image is created. For the third image, I used outlier scores obtained from HDBSCAN clustering to mask the image. It has shown that despite identifying two anomalies in the channel, it is not enough. The conclusion was that there is a problem in the features extracted from the variational autoencoder. To test the model, and its functioning, I had to train the model with a toy dataset, which can provide ground truth. I have train the model with MIMII dataset, which is a sound dataset for malfunctioning industrial machine investigation and inspection [16]. It was suited for the task, since it is a 1D data and has anomalies in it. The training results are given in Figure 20.

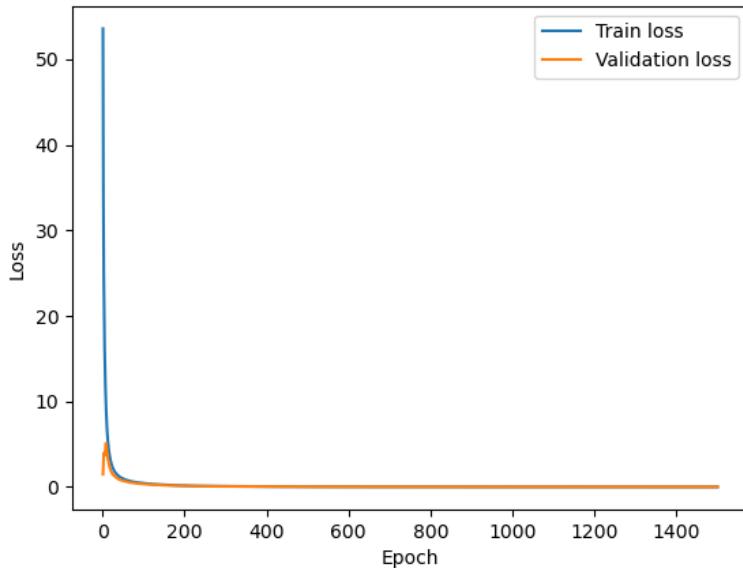


Figure 20: Training results of VAE with a 1D toy dataset called MIMII

After repeating the same steps for MIMII dataset, I have concluded that there is definitely something wrong with the model. To decide on that, I have used accuracy, precision, recall, and F1 score, since I do know the ground truth. In this stage, to train I have learned a lot about not only variational autoencoders but also optimizers and losses used in machine learning tasks. Moreover, I have learned about the pre-processing steps applied to our 1D dataset, which includes taking spectrograms of 1D data and feeding as input. I will explain the pre-processing steps more clearly, since, at the end, I have found out that the source of the errors is pre-processing.

By using the real dataset, I extracted a sample image from the dataset after pre-processing steps. I used the sample image as an input to the model, and tried to see the decoded image, which is shown in Figure 21. As seen from the figure, the decoded input does not represent the input at all, which means that the model is not trained well, and encoded features are not representing the data. Overall, it was an experience to figure out what is not right. This meant that what I was trying to cluster does not represent the data, clustering results are not meaningful. However, this realization helped to move forward, a step to a right direction.

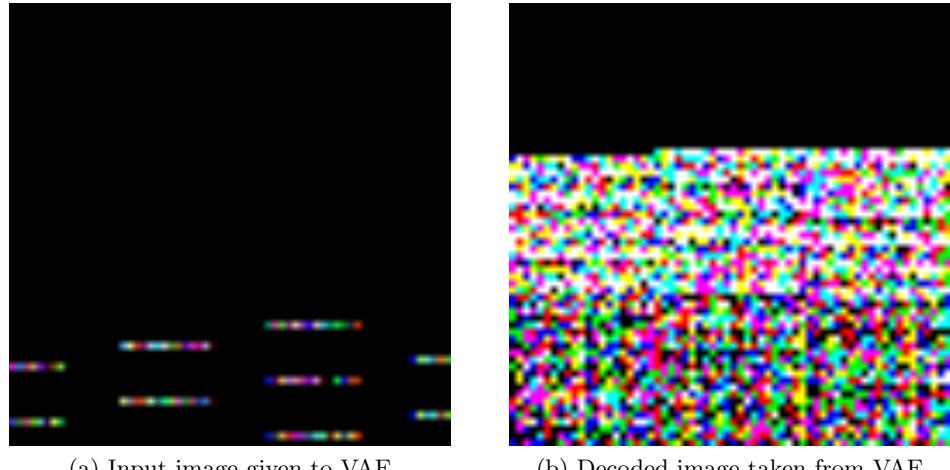
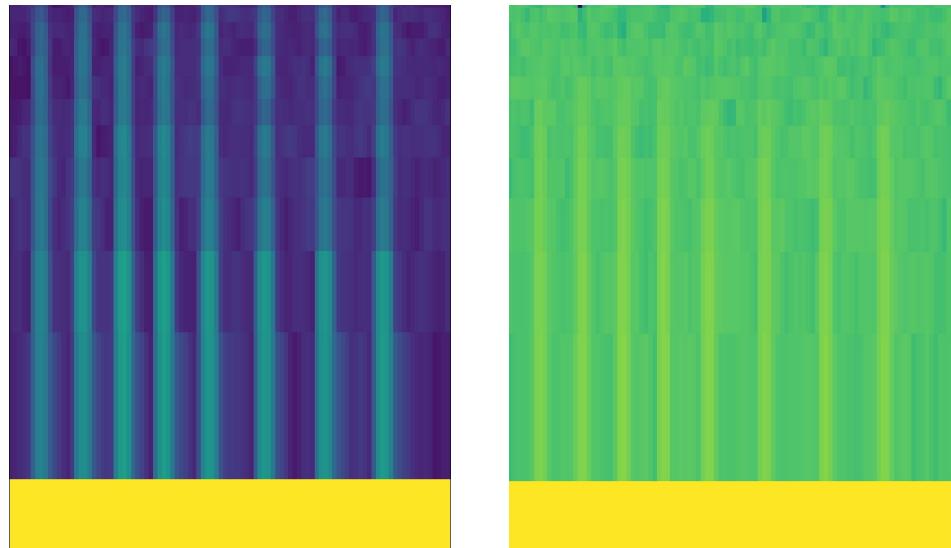


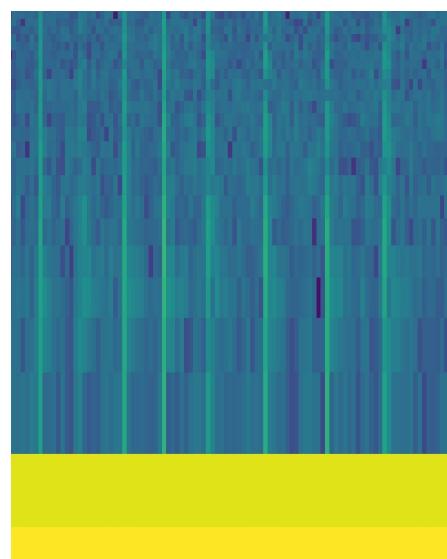
Figure 21: Input of the encoder and output of the decoder

The input is an audio-like data as mentioned before. The signal has an amplitude and changes over time. To extracting useful information out of this data can be done in time domain or frequency domain. For example, temporal dissimilarity model focuses on time domain signal. Frequency domain representation is called the spectrum and found by Fourier Transform. However, frequency changes over time since the signal is not periodic. So, a way is needed to show spectra over time. By computing spectrum over overlapping parts of the signal, we obtain what is called a spectrogram, which shows frequency on one axis, time on the other axis and amplitude by color coding. Thus, we have created an image representing 1D data. There are many ways of employing 1D data to an image. Which frequency range will be taken? How many channels will be shown? What will be the time period? How much overlapping will be selected? Will it be gray scale or color coded? There are also several spectrograms to be selected. There are too many choices for model using spectrograms as inputs. The contractor requested that the input that will be fed to the model should be fixed, which included several types of spectrograms. For feature extraction, librosa library for Python is used, which is a library for audio analysis. "melspectrogram", "mfcc", and "stft" were the main candidates. "stft" is short-time fourier transform, "mel-spectrogram" is a mel-scaled spectrogram, and "mfcc" is mel-frequency cepstral coefficients. I have researched all three of them, which I will not explain. For further information, documentation of "librosa" can be examined [17].



(a) Mel-scaled spectrogram

(b) MFCC



(c) STFT

Figure 22: Features extracted from 1 channel with a frequency range of 100Hz and a time duration of 25 seconds

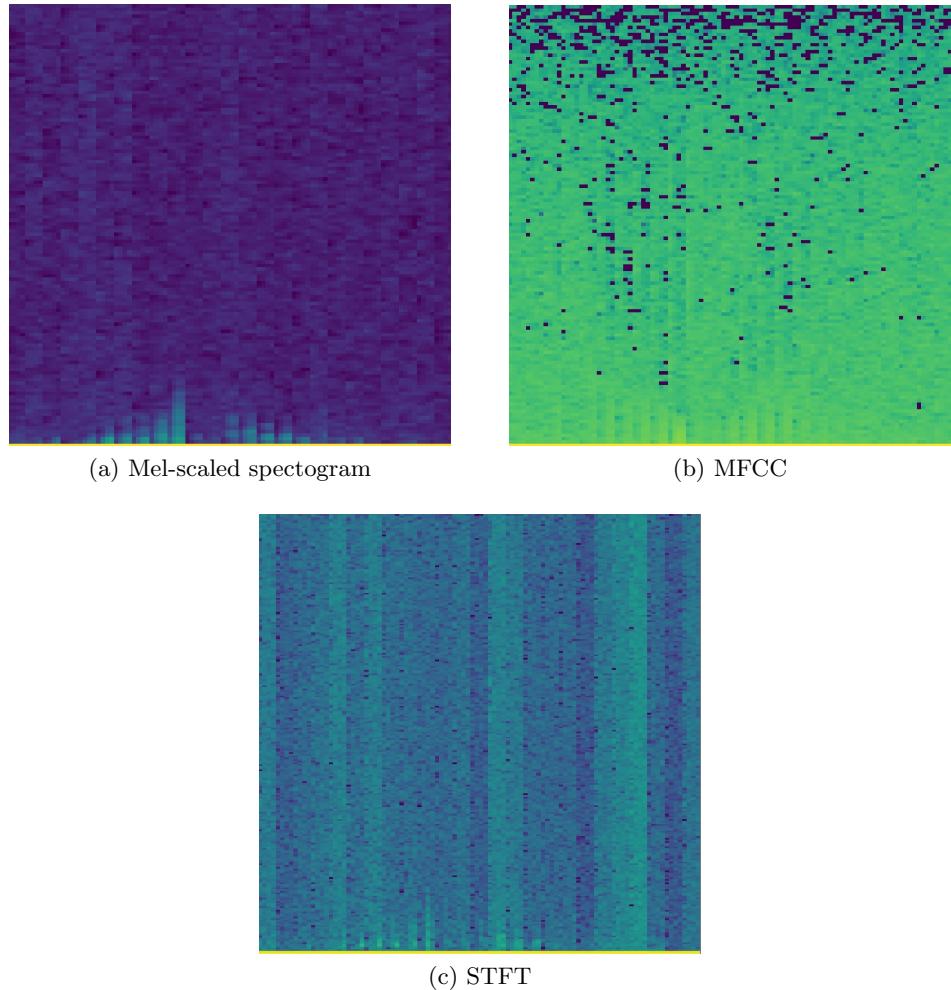
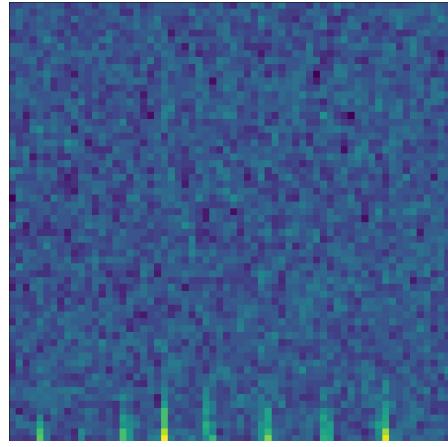
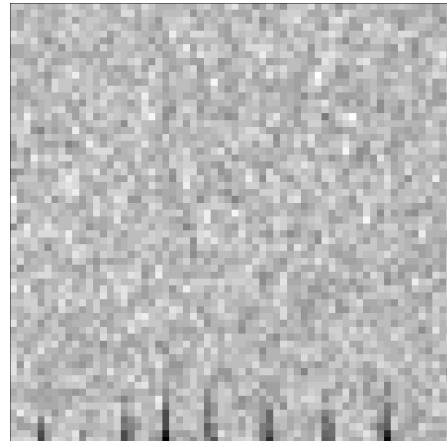


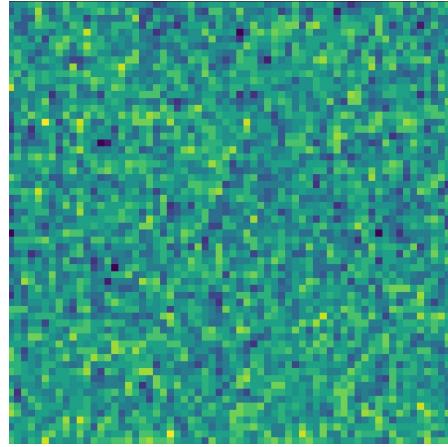
Figure 23: Features extracted from 25 channels with a frequency range of 1kHz and a time duration of 1 seconds



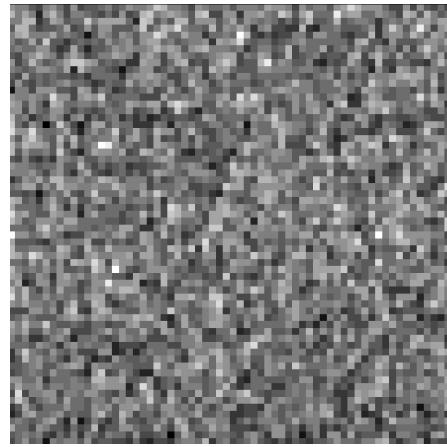
(a) Color coded mel-scaled spectrogram  
for anomaly



(b) Gray scaled mel-scaled spectrogram  
for anomaly



(c) Color coded mel-scaled spectrogram  
for normal



(d) Gray scaled mel-scaled spectrogram  
for normal

Figure 24: Features extracted from 1 channel with a frequency range of 1kHz and a time duration of 25 seconds

As seen from the Figures 22, 23, and 24, different employments of 1D data is possible and it is a design parameter. For this task, I have written a modular code to generate a dataset with desirable parameters. From Figures 22 and 23, for different selection of spectrograms, mel-scaled spectrogram gives better visual results, and expected to give better results when applied. Thus, a further visualization of abnormal and normal data is shown in Figure 24 for both in gray and color.

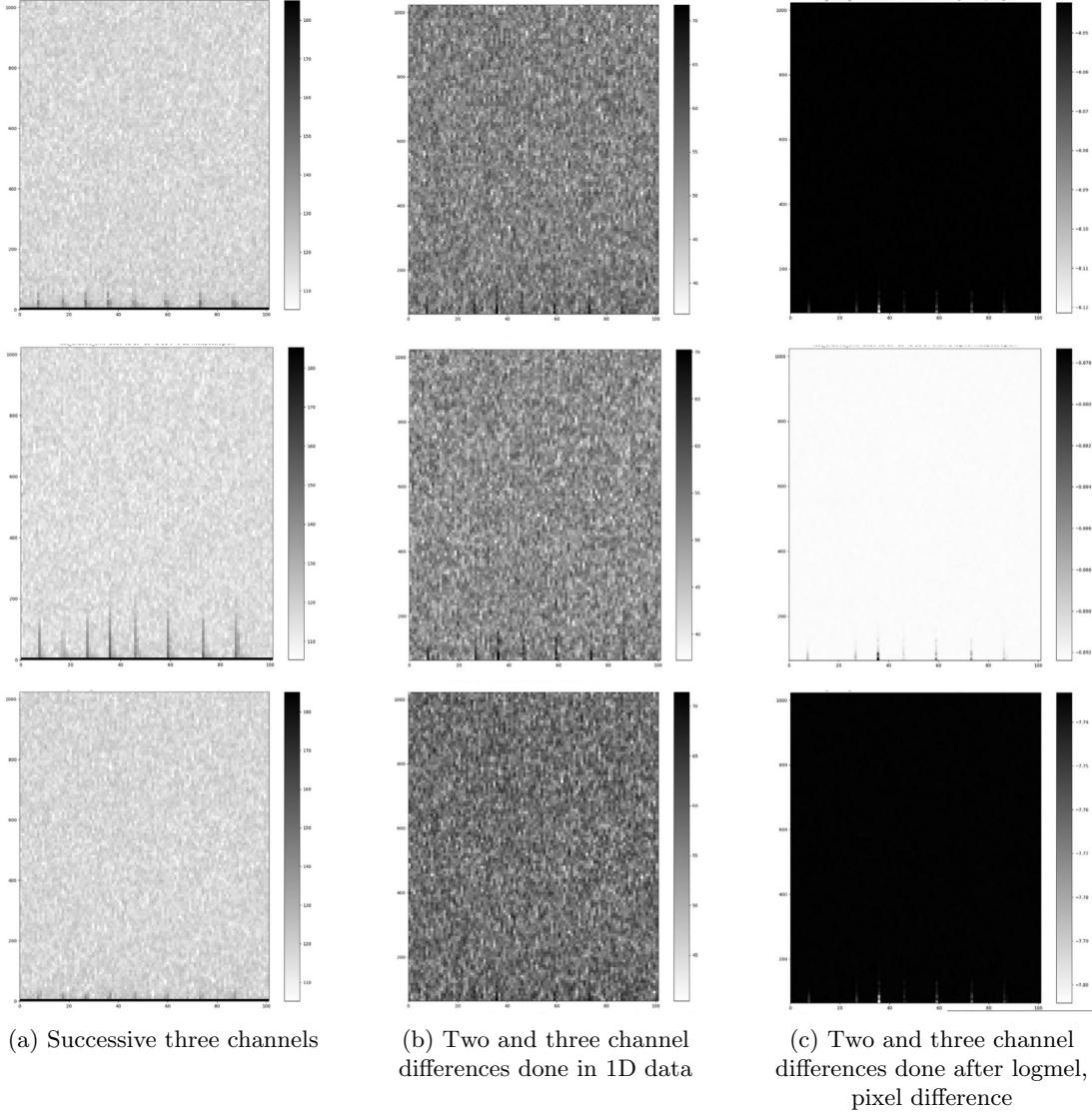


Figure 25: Enhancement trials by adding channel difference

To enhance the effect of the channel, to exaggerate the anomaly, difference methods are used. First, a difference in 1D taken between channels and then it is converted to a mel-spectrogram. In another version, after converting the mel-spectrogram, the pixel difference is taken to exaggerate. The results of this experiment can be seen in Figure 25. Comment on that may be 1D difference adds noise to spectrogram and mel-spectrogram difference is not successful. Enhancement can not be done in this way. Anomaly already can be detected easily without any enhancement, it may be unnecessary to add computational cost.

At this step, I also tried to work on another model called infoGAN, a GAN based model, which I mentioned about in Section 3.2.6. At that time, my supervisor implemented a model based on temporal dissimilarity mentioned in Section 3.2.7. I mentioned about the temporal dissimilarity model since I loved the idea and I was given to read and understand at the beginning of my internship. However, the implementation is not done by me, thus, there will be no results shown about the model.

While trying to implement the model, the training pushed the limits of my computer. Thus, I was not able to implement the model in my computer. So, I migrated the model to Google Colab, however, due to the confidentiality of the data, I was only able to run with MNIST dataset, which gave the expected results. I was not able to complete it with the real dataset, since at the time we have found the errors in the pre-processing, and I was expected to correct them. I have pre-processed the data by deleting the noise component seen in Figure 22 as a yellow band in low frequencies, choosing overlapping instead of padding, and resizing the image to correct dimensions for the model. I have managed to obtain a dataset of images like in Figure 24. By comparing what is used before in Figure 21, I think I helped to correct an important part of the procedure. I did not see the results of the models trained with the dataset I created, but I hope I helped and I hope it works.

## 4 CONCLUSION

I spent my six weeks interning at Kuartis Technology and Consulting. Kuartis created an environment in which I felt as soon as I walked in, that everyone at Kuartis has a vision, and is trying to do something impressive in the field. They reminded me that it is not easy to achieve the goals, it is necessary to make an effort to achieve them, and that doing all this with pleasure is the most important thing. Additionally, I felt like I was able to contribute to the company by assisting on the project throughout my internship. I have attended both internal, which were daily and weekly, and external meetings, where I observed the operation of the company. Furthermore, I also attended weekly paper reading sessions to keep the research environment alive.

In this journey, I have learned what is to be a machine learning engineer and a data scientist. Furthermore, I have used my skills in signal processing to improve the project. While implementing, I have always kept in mind that theories matter and I should look at them and try to understand them. I have worked on several machine learning models such as autoencoders and generative adversarial networks. I have implemented clustering algorithms and used several visualization methods. I have learned how to measure the performance and comment on the results. I have gained a huge amount of experience in exploratory data analysis. To be able to do these, I benefited from various research papers and vast knowledge of my supervisor, which contributed to me a lot.

Not only did I gain practical skills but I felt the great atmosphere at the office while paying attention to precautions for COVID-19, which was always welcoming. Everyone was allowed to go to the kitchen at any time and have their favourite snacks. People at the office had a chance to enjoy delicious baklavas and böreks, while some people were working from home. Participating such company events made interns feel like a part of the company.

All in all, I can say that my internship has been a success. I had an great opportunity to learn new things, feel the work environment and meet with incredible people. I could not be more thankful, and definitely I recommend Kuartis for a summer practice location.

## 5 REFERENCES

- [1] Hill, D., "Distributed acoustic sensing (das): Theory and applications," in [Frontiers in Optics 2015], Frontiers in Optics 2015 , FTh4E.1, Optical Society of America (2015).
- [2] Barnoski, M. K., Rourke, M. D., Jensen, S. M., and Melville, R. T., "Optical time domain reflectometer," Appl. Opt. 16, 2375–2379 (1977).
- [3] Barnoski, M. K. and Jensene, S. M., "Fiber waveguides - novel technique for investigating attenuation characteristics," Appl. Opt. 15, 2112–2115 (1976).
- [4] Metin Aktas, Toygar Akgun, Mehmet Umut Demircin, Duygu Buyukaydin, "Deep learning based multi-threat classification for phase-OTDR fiber optic distributed acoustic sensing applications," Proc. SPIE 10208, Fiber Optic Sensors and Applications XIV, 102080G (27 April 2017); <https://doi.org/10.1117/12.2262108>
- [5] Tejedor, J.; Macias-Guarasa, J.; Martins, H.F.; Pastor-Graells, J.; Corredora, P.; Martin-Lopez, S. Machine Learning Methods for Pipeline Surveillance Systems Based on Distributed Acoustic Sensing: A Review. Appl. Sci. 2017, 7, 841. <https://doi.org/10.3390/app7080841>
- [6] Friedman, Ellen and Ted Dunning. "Practical Machine Learning: A New Look at Anomaly Detection." (2014).
- [7] M. E. Celebi and K. Aydin, Eds., Unsupervised Learning Algorithms. Springer International Publishing, 2016.
- [8] Brian S. Everitt, Sabine Landau, and Morven Leese. 2009. Cluster Analysis (4th. ed.). Wiley Publishing.
- [9] L. McInnes, J. Healy, S. Astels, hdbscan: Hierarchical density based clustering In: Journal of Open Source Software, The Open Journal, volume 2, number 11. 2017
- [10] Kingma, Diederik Welling, Max. (2014). Auto-Encoding Variational Bayes.
- [11] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio. "Generative Adversarial Networks." ArXiv 2014.
- [12] T. Komatsu and R. Kondo, "Detection of anomaly acoustic scenes based on a temporal dissimilarity model," 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), New Orleans, LA, USA, 2017, pp. 376-380, doi: 10.1109/ICASSP.2017.7952181.
- [13] Diederik P. Kingma, et al. "Adam: A Method for Stochastic Optimization." (2017).
- [14] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep Learning. MIT Press, 2016.

- [15] L.J.P. van der Maaten and G.E. Hinton. Visualizing High-Dimensional Data Using t-SNE. *Journal of Machine Learning Research* 9(Nov):2579-2605, 2008.
- [16] Harsh Purohit, et al. "MIMII Dataset: Sound Dataset for Malfunctioning Industrial Machine Investigation and Inspection." (2019).
- [17] McFee, Brian, Colin Raffel, Dawen Liang, Daniel PW Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto. "librosa: Audio and music signal analysis in python." In *Proceedings of the 14th python in science conference*, pp. 18-25. 2015.

## 6 APPENDICES

### A HDBSCAN Implementation

```
1 data = test_set.data.type(torch.FloatTensor).to(device)
2 print(data.shape)
3 mu, var = embed_model.encode(data)
4 mu = np.nan_to_num(mu.cpu().detach().numpy())
5 var = np.nan_to_num(var.cpu().detach().numpy())
6 hdb_data_t = np.concatenate((mu, var), axis=1)
7 hdb_label_t = test_set.labels
8 name = test_set.names
9 print("Data embedded.")

10
11 cluster_model = hdbscan.HDBSCAN(min_cluster_size=2, min_samples=200,
12                                 allow_single_cluster=False, prediction_data=True).fit(hdb_data_t)
12 pred_y, _ = hdbscan.approximate_predict(cluster_model, hdb_data_t)
13 cluster_model.condensed_tree_.plot(select_clusters=True, selection_palette=
14         sns.color_palette('deep', 8))

14
15 print("silhouette_score:", silhouette_score(hdb_data_t, pred_y))
16 print("calinski_harabasz_score:", calinski_harabasz_score(hdb_data_t, pred_y
17     ))
17 print("davies_bouldin_score:", davies_bouldin_score(hdb_data_t, pred_y))
18
19 sns.displot(cluster_model.outlier_scores_[np.isfinite(cluster_model.
20     outlier_scores_)], rug=True)
20 threshold = pd.Series(cluster_model.outlier_scores_).quantile(0.9)
21 print("threshold:", threshold)
22 outliers = np.where(cluster_model.outlier_scores_ > threshold)[0]
23 outlier_scores = cluster_model.outlier_scores_
```

### B t-SNE Implementation

```
1 #2D TSNE
2 cluster_model = TSNE(2, init='pca', random_state=32,
3                         learning_rate=100.0, perplexity=40.0)
4
5 Y = cluster_model.fit_transform(hdb_data_t)
6 print("2D Model Fitted")
7
8 fig = plt.figure(figsize=(45, 38))
9 ax = fig.add_subplot(111)
10 lo = ax.scatter(Y[class1, 0], Y[class1, 1], c='y', marker='.', alpha=0.1)
11 ll = ax.scatter(Y[class2, 0], Y[class2, 1], c='g', marker='.', alpha=0.3)
12 l = ax.scatter(Y[class3, 0], Y[class3, 1], c='b', marker='.', alpha=0.3)
13 a = ax.scatter(Y[class4, 0], Y[class4, 1], c='r', marker='o', alpha=0.7)
14 #o = ax.scatter(Y[class5, 0], Y[class5, 1], c='k', marker='o', alpha=0.7)
15 plt.legend((lo,ll,l,a), ('normal','kuartis','ortak'), loc='lower left')
16
17 ax.xaxis.set_major_formatter(NullFormatter())
18 ax.yaxis.set_major_formatter(NullFormatter())
```

```

19 ax.axis('tight')
20 plt.title('T-SNE 2D')
21
22 #3D TSNE
23 cluster_model = TSNE(3, init='pca', random_state=32,
24                      learning_rate=100.0, perplexity=40.0)
25
26 Y = cluster_model.fit_transform(hdb_data_t)
27 print("3D Model Fitted")
28
29 fig = plt.figure(figsize=(45, 38))
30 ax = fig.add_subplot(111, projection='3d')
31 lo = ax.scatter(Y[:,0][class1], Y[:,1][class1], Y[:,2][class1], c= 'y',
32                  marker='.', alpha=0.1)
32 ll = ax.scatter(Y[:,0][class2], Y[:,1][class2], Y[:,2][class2], c= 'g',
33                  marker='.', alpha=0.3)
33 l = ax.scatter(Y[:,0][class3], Y[:,1][class3], Y[:,2][class3], c= 'b',
34                  marker='.', alpha=0.3)
34 a = ax.scatter(Y[:,0][class4], Y[:,1][class4], Y[:,2][class4], c= 'r',
35                  marker='o', alpha=0.7)
35 #o = ax.scatter(Y[:,0][class5], Y[:,1][class5], Y[:,2][class5], c= 'r',
36                  marker='o', alpha=0.7)
36 plt.legend((lo,ll,l,a), ('normal','kuartis','ortak'), loc='lower left')
37
38 ax.xaxis.set_major_formatter(NullFormatter())
39 ax.yaxis.set_major_formatter(NullFormatter())
40 ax.zaxis.set_major_formatter(NullFormatter())
41 ax.axis('tight')
42 plt.title('T-SNE 3D')

```

## C Waterfall Image Implementation

```

1 def visualize_anomaly(data_path='', f=' ', act_class_map={'kazma':0,'yurume':
2 :1,'kesici':2,'makine':3,'gurultu':4},
3                         activity='makine', data=[], sr=1.0/2000.0, channel=None,
4                         save=False, return_fig=False):
5     """
6     Visualize Anomalies in Channels
7     """
8     info = readInfo(os.path.join(data_path, f+'.bin.info'))
9     # Input specific exceptions!
10    try:
11        numChannel = info['selected_channel_zones'][0][1]-info['
12 selected_channel_zones'][0][0] +1
13    except IndexError:
14        #print(f, info)
15        numChannel = int(info['number_of_channels_in_one_sample'])
16    except TypeError:
17        #print(f, info)
18        numChannel = info['selected_channel_zones'][1]-info['
19 selected_channel_zones'][0]+1
20    # Read data

```

```

17     prob = 0
18     for k, value in act_class_map.items():
19         prob += readProb(os.path.join(data_path, 'Model8/' + f + '.bin_prob.prob'),
20                           numChannel, act_class_map, numClass=len(act_class_map.keys()),
21                           activity=value)
22     realPower, normPower = readPower(os.path.join(data_path, 'Model8/' + f + '.bin_power.pow'),
23                                      numChannel)
24
25     fig = plt.figure(figsize=(15, 8))
26     t = np.arange(0, realPower.shape[0]/5, step=1/5)
27     if channel is None:
28         # Plot time series data for all channels
29         ax = fig.add_subplot(131)
30         ax.pcolormesh(realPower, vmin=0, vmax=25)
31     else:
32         # plot time series data for a single channel
33         x = np.arange(0, channel[-1]+1, step=1)
34         ax = fig.add_subplot(131)
35         ax.pcolormesh(x, t, realPower[:, channel])
36         plt.ylabel('Time (sec)')
37         plt.xlabel('Channels')
38         plt.title('Unmasked Waterfall')
39
40     realPower = realPower[6:realPower.shape[0], :]
41     masked_rP = np.multiply(prob, realPower)
42
43     t = np.arange(0, masked_rP.shape[0]/5, step=1/5)
44     if channel is None:
45         # Plot time series data for all channels
46         ax = fig.add_subplot(132)
47         ax.pcolormesh(masked_rP, vmin=0, vmax=25)
48     else:
49         # plot time series data for a single channel
50         x = np.arange(0, channel[-1]+1, step=1)
51         ax = fig.add_subplot(132)
52         ax.pcolormesh(x, t, masked_rP[:, channel])
53         plt.xlabel('Channels')
54         plt.title('Masked Waterfall, x')
55
56     data = np.repeat(data, 5, axis=0)
57     data = data[1:data.shape[0]-2, :]
58     masked_kuarr = np.multiply(data, realPower)
59     #masked_kuarr = np.multiply(prob, masked_kuarr)
60     t = np.arange(0, masked_kuarr.shape[0]/5, step=1/5)
61     if channel is None:
62         # Plot time series data for all channels
63         ax = fig.add_subplot(133)
64         ax.pcolormesh(masked_kuarr, vmin=0, vmax=25)
65     else:
66         # plot time series data for a single channel
67         x = np.arange(0, channel[-1]+1, step=1)
68         ax = fig.add_subplot(133)

```

```

66         ax.pcolormesh(x, t, masked_kuarr[:, channel])
67         plt.xlabel('Channels')
68         plt.title('Masked Waterfall, KUART S')
69
70     # Save the figure as jpeg
71     if save:
72         fig.savefig(os.path.join(data_path, f+'anomaly.jpeg'), optimize=True)
73
74     if return_fig:
75         return fig
76     else:
77         plt.close()

```

## D Testing VAE

```

1 data = torch.FloatTensor(dataset[0]).to(device)
2 const_data = embed_model(data)[0]
3 const_data = const_data.numpy()*255
4 const_data = const_data.reshape((64, 64, 3))
5 data = data.numpy()
6 data = data.reshape((64, 64, 3))
7 image1 = Image.fromarray((data).astype(numpy.uint8))
8 image1.show()
9 image2 = Image.fromarray((const_data).astype(numpy.uint8))
10 image2.save('my2.png')
11 image2.show()

```

## E Mel-Spectrogram Generation

```

1 """LOGMEL DATA GENERATION AND VISUALIZATION"""
2
3 import os, sys
4 import numpy as np
5 import matplotlib.pyplot as plt
6 from matplotlib import cm as cmaps
7 import librosa
8 import librosa.display
9 from readData import readInfo, readRawData
10 from PIL import Image
11
12 def channel_diff(segment0, segment1, segment2):
13     if segment2 is None:
14         seg01 = segment0 - segment1
15         seg21 = np.ones_like(seg01)
16     else:
17         seg01 = segment0 - segment1
18         seg21 = segment2 - segment1
19     return seg01 * seg21
20
21 def melspectrogram(data_path=' ', f=' ', save_path=' ', time_window=1.,
22                     time_shift=1., sr=1./2000., channel=None,

```

```

22             use_channels=None, save=False, return_fig=False,
23             nfft=512, vmin=None, vmax=None,
24             chdiff=False, mode="1D", diff=5, n_diff=3):
25
26     # Input specific exceptions!
27     info = readInfo(os.path.join(data_path, f+'.bin.info'))
28     try:
29         numChannel = info['selected_channel_zones'][0][1]-info['
29 selected_channel_zones'][0][0] +1
30     except IndexError:
31         #print(f, info)
32         numChannel = int(info['number_of_channels_in_one_sample'])
33     except TypeError:
34         #print(f, info)
35         numChannel = info['selected_channel_zones'][1]-info['
35 selected_channel_zones'][0]+1
36
37     # Read data
38     rawData = readRawData(os.path.join(data_path, f+'.bin'), numChannel)
39     time_index_shift = int(time_shift/sr)
40
41     # Check whether a time step is specified, if not use all samples
42     # Overwrites time shift!
43     if time_window:
44         time_steps = int(time_window/sr)
45     else:
46         time_steps = rawData.shape[0]
47         time_index_shift = rawData.shape[0]
48
49     if use_channels:
50         j = max(0,use_channels[0])
51         j_stop = min(use_channels[-1],rawData.shape[1])
52     else:
53         j = 0
54         j_stop = rawData.shape[1]
55
56     # Start iterating
57     while j < j_stop: # for channels
58         i = 0
59         while i+time_steps <= rawData.shape[0]: # for time steps
60             print("Channel:", j, "Time:", i*sr)
61             seg1 = np.asfortranarray(rawData[i:i+time_steps, j].copy()
62             .astype(float))
63             print(seg1.shape)
64             if chdiff and i > diff:
65                 seg0 = np.asfortranarray(rawData[i:i+time_steps, j-diff].
66                 copy().astype(float))
67                 seg2 = np.asfortranarray(rawData[i:i+time_steps, j+diff].
68                 copy().astype(float))
69
70                 if mode == "1D":
71                     if n_diff == 2:
72                         rawdiff = channel_diff(seg0.reshape(-1, 1), seg1.
73                         reshape(-1, 1), None)
74                     else:
75                         rawdiff = channel_diff(seg0.reshape(-1, 1), seg1.
76                         reshape(-1, 1), seg2.reshape(-1, 1))

```

```

66         ps = librosa.feature.melspectrogram(y=rawdiff.flatten(),
67         sr=sr, n_fft=nfft)
68         if mode == "logmel":
69             ps0 = librosa.feature.melspectrogram(y=seg0, sr=sr,
70             n_fft=nfft).astype(np.float64)
71             ps1 = librosa.feature.melspectrogram(y=seg1, sr=sr,
72             n_fft=nfft).astype(np.float64)
73             if n_diff == 2:
74                 ps = channel_diff(ps0, ps1, None)
75             else:
76                 ps2 = librosa.feature.melspectrogram(y=seg2, sr=sr,
77                 n_fft=nfft).astype(np.float64)
78                 ps = channel_diff(ps0, ps1, ps2)
79
80             elif channel == 1:
81                 ps = librosa.feature.melspectrogram(y=seg1, sr=sr, n_fft=
82             nfft)
83
84             elif channel!=1:
85                 if j % channel == 0 and j!=0:
86                     data = np.zeros_like(rawData[i:i+time_steps, j], dtype=
87             float)
88                     for k in range(0,channel-1):
89                         data += rawData[i:i+time_steps, j-k]
90                     ps = librosa.feature.melspectrogram(y=data, sr=sr, n_fft=
91             =nfft)
92
93                 if channel==1 or (j % channel == 0 and j!=0):
94                     ps_db = librosa.power_to_db(ps, ref=1.0)
95                     ps_db = ps_db[8:,:]
96                     #ps_db = np.repeat(ps_db, int(ps_db.shape[0]/ps_db.shape[1]))
97                     #while ps_db.shape[0] != 128:
98                         #    ps_db = np.concatenate((ps_db, ps_db[ps_db.shape[0]-1].
99                     reshape(1,ps_db.shape[1])), axis=0)
100                     #while ps_db.shape[1] != 128:
101                         #    ps_db = np.concatenate((ps_db, ps_db[:,ps_db.shape
102                     [1]-1].reshape(ps_db.shape[0],1)), axis=1)
103
104                     ps_db = Image.fromarray(np.uint8(ps_db)).resize((64,64))
105                     ps_db = np.asarray(ps_db)
106                     ps_db = (ps_db - np.min(ps_db)) / np.ptp(ps_db)
107                     print(ps_db.shape)
108                     if vmin != np.min(ps_db) or vmax != np.max(ps_db):
109                         print("Different limits!", np.min(ps_db), np.max(ps_db))
110
111                     if save:
112                         if not os.path.exists(save_path):
113                             os.mkdir(save_path)
114                         if return_fig:
115                             fig = plt.figure(figsize=(12.8, 12.8), dpi=100)
116                             plt.pcolormesh(ps_db, vmin=vmin, vmax=vmax)

```

```

108         plt.set_cmap(cmmaps.binary)
109         if vmin and vmax is not None:
110             plt.clim(vmin, vmax)
111         fig.savefig(os.path.join(save_path, f+'-CH_{:d}'.format(j)+_
112             '-T_{:d}'.format(i)+'-CHS_{:d}'.format(channel)+'-CHD_{:d}'.format(chdiff)+_
113             '-ND_{:d}'.format(n_diff)+'-M_{:d}'.format(mode)+'.mel.jpeg'))
114         plt.close()
115         np.save(os.path.join(save_path, f+'-CH_{:d}'.format(j)+_
116             '-T_{:d}'.format(i)+'-CHS_{:d}'.format(channel)+'-CHD_{:d}'.format(chdiff)+_
117             '-ND_{:d}'.format(n_diff)+'-M_{:d}'.format(mode)+'.mel.jpeg'), ps_db)
118
119     i = i + time_index_shift # shift time
120     j = j+1 # shift channels
121
122
123
124 if __name__ == "__main__":
125     ROOT = ''
126     SAVE = ''
127
128     FILES = sorted([f[:-4] for f in os.listdir(ROOT) if f[-4:] == ".bin"])
129     for f in FILES:
130         melspectrogram(data_path=ROOT, f=f, save_path=SAVE, time_window=10.,
131                         time_shift=1., sr=1./2000., channel=1,
132                         use_channels=None, save=True, return_fig=True,
133                         nfft=512, vmin=0., vmax=1.,
134                         chdiff=False, mode="1D", diff=5, n_diff=3)
135     sys.exit("done")

```

## F infoGAN Implementation

```

1 import os
2 import numpy as np
3 import math
4 import itertools
5
6 import torchvision.transforms as transforms
7 from torchvision.utils import save_image
8
9 from torch.utils.data import DataLoader
10 from torchvision import datasets
11 from torch.autograd import Variable
12
13 import torch.nn as nn
14 import torch.nn.functional as F
15 import torch
16
17 os.makedirs("images/static/", exist_ok=True)
18 os.makedirs("images/varying_c1/", exist_ok=True)
19 os.makedirs("images/varying_c2/", exist_ok=True)
20
21 n_epochs = 1000

```

```

22 batch_size = 64
23 lr = 0.001
24 b1 = 0.5
25 b2 = 0.999
26 n_cpu = 8
27 latent_dim = 62
28 code_dim = 2
29 sample_interval = 400
30 n_classes = 2
31 img_size = 64
32 channels = 1
33
34 cuda = True if torch.cuda.is_available() else False
35
36 from google.colab import drive
37 drive.mount('/content/drive')
38
39 DATASET_PATH = "/content/drive/My Drive/SAMPLE/gri/"
40
41 _, _, train_set = get_dataset(DATASET_PATH, test_size=1, valid_size=0,
42                               is_grayscale=True, balanced=False, fresh_start=True, data_type='mel')
43
44 dataloader = torch.utils.data.DataLoader(train_set, batch_size=batch_size,
45                                         shuffle=True)
46
47 print("Data Loaded")
48 BEST_PATH = "/content/drive/My Drive/SAMPLE/"
49 if not os.path.exists(BEST_PATH):
50     os.mkdir(BEST_PATH)
51 BEST_MODEL1 = "best_infoGAN_generator.pth"
52 BEST_MODEL2 = "best_infoGAN_discriminator.pth"
53
54 def weights_init_normal(m):
55     classname = m.__class__.__name__
56     if classname.find("Conv") != -1:
57         torch.nn.init.normal_(m.weight.data, 0.0, 0.02)
58     elif classname.find("BatchNorm") != -1:
59         torch.nn.init.normal_(m.weight.data, 1.0, 0.02)
60         torch.nn.init.constant_(m.bias.data, 0.0)
61
62 def to_categorical(y, num_columns):
63     """Returns one-hot encoded Variable"""
64     y_cat = np.zeros((y.shape[0], num_columns))
65     y_cat[range(y.shape[0]), y] = 1.0
66
67     return Variable(FloatTensor(y_cat))
68
69 class Generator(nn.Module):
70     def __init__(self):
71         super(Generator, self).__init__()

```

```

72     input_dim = latent_dim + n_classes + code_dim
73
74     self.init_size = img_size // 4 # Initial size before upsampling
75     self.l1 = nn.Sequential(nn.Linear(input_dim, 128 * self.init_size ** 2))
76
77     self.conv_blocks = nn.Sequential(
78         nn.BatchNorm2d(128),
79         nn.Upsample(scale_factor=2),
80         nn.Conv2d(128, 128, 3, stride=1, padding=1),
81         nn.BatchNorm2d(128, 0.8),
82         nn.LeakyReLU(0.2, inplace=True),
83         nn.Upsample(scale_factor=2),
84         nn.Conv2d(128, 64, 3, stride=1, padding=1),
85         nn.BatchNorm2d(64, 0.8),
86         nn.LeakyReLU(0.2, inplace=True),
87         nn.Conv2d(64, channels, 3, stride=1, padding=1),
88         nn.Tanh(),
89     )
90
91     def forward(self, noise, labels, code):
92         gen_input = torch.cat((noise, labels, code), -1)
93         out = self.l1(gen_input)
94         out = out.view(out.shape[0], 128, self.init_size, self.init_size)
95         img = self.conv_blocks(out)
96         return img
97
98
99 class Discriminator(nn.Module):
100     def __init__(self):
101         super(Discriminator, self).__init__()
102
103         def discriminator_block(in_filters, out_filters, bn=True):
104             """Returns layers of each discriminator block"""
105             block = [nn.Conv2d(in_filters, out_filters, 3, 2, 1), nn.
106             LeakyReLU(0.2, inplace=True), nn.Dropout2d(0.25)]
107             if bn:
108                 block.append(nn.BatchNorm2d(out_filters, 0.8))
109             return block
110
111         self.conv_blocks = nn.Sequential(
112             *discriminator_block(channels, 16, bn=False),
113             *discriminator_block(16, 32),
114             *discriminator_block(32, 64),
115             *discriminator_block(64, 128),
116         )
117
118         # The height and width of downsampled image
119         ds_size = img_size // 2 ** 4
120
121         # Output layers
122         self.adv_layer = nn.Sequential(nn.Linear(128 * ds_size ** 2, 1))

```

```

122         self.aux_layer = nn.Sequential(nn.Linear(128 * ds_size ** 2,
123                                         n_classes), nn.Softmax())
124         self.latent_layer = nn.Sequential(nn.Linear(128 * ds_size ** 2,
125                                         code_dim))
126
127     def forward(self, img):
128         out = self.conv_blocks(img)
129         out = out.view(out.shape[0], -1)
130         validity = self.adv_layer(out)
131         label = self.aux_layer(out)
132         latent_code = self.latent_layer(out)
133
134     return validity, label, latent_code
135
136 # Loss functions
137 adversarial_loss = torch.nn.MSELoss()
138 categorical_loss = torch.nn.CrossEntropyLoss()
139 continuous_loss = torch.nn.MSELoss()
140
141 # Loss weights
142 lambda_cat = 1
143 lambda_con = 0.1
144
145 # Initialize generator and discriminator
146 generator = Generator()
147 discriminator = Discriminator()
148
149 if cuda:
150     generator.cuda()
151     discriminator.cuda()
152     adversarial_loss.cuda()
153     categorical_loss.cuda()
154     continuous_loss.cuda()
155
156 # Initialize weights
157 generator.apply(weights_init_normal)
158 discriminator.apply(weights_init_normal)
159
160 # Optimizers
161 optimizer_G = torch.optim.Adam(generator.parameters(), lr=lr, betas=(b1, b2))
162 optimizer_D = torch.optim.Adam(discriminator.parameters(), lr=lr, betas=(b1, b2))
163 optimizer_info = torch.optim.Adam(
164     itertools.chain(generator.parameters(), discriminator.parameters()), lr=
165     lr, betas=(b1, b2))
166
167 FloatTensor = torch.cuda.FloatTensor if cuda else torch.FloatTensor
168 LongTensor = torch.cuda.LongTensor if cuda else torch.LongTensor

```

```

169
170 # Static generator inputs for sampling
171 static_z = Variable(FloatTensor(np.zeros((n_classes ** 2, latent_dim))))
172 static_label = to_categorical(
173     np.array([num for _ in range(n_classes) for num in range(n_classes)]),
174     num_columns=n_classes
175 )
176 static_code = Variable(FloatTensor(np.zeros((n_classes ** 2, code_dim))))
177
178 def sample_image(n_row, batches_done):
179     """Saves a grid of generated digits ranging from 0 to n_classes"""
180     # Static sample
181     z = Variable(FloatTensor(np.random.normal(0, 1, (n_row ** 2, latent_dim))
182     ))
183     static_sample = generator(z, static_label, static_code)
184     save_image(static_sample.data, "images/static/%d.png" % batches_done,
185     nrow=n_row, normalize=True)
186
187     # Get varied c1 and c2
188     zeros = np.zeros((n_row ** 2, 1))
189     c_varied = np.repeat(np.linspace(-1, 1, n_row)[:, np.newaxis], n_row, 0)
190     c1 = Variable(FloatTensor(np.concatenate((c_varied, zeros), -1)))
191     c2 = Variable(FloatTensor(np.concatenate((zeros, c_varied), -1)))
192     sample1 = generator(static_z, static_label, c1)
193     sample2 = generator(static_z, static_label, c2)
194     save_image(sample1.data, "images/varying_c1/%d.png" % batches_done, nrow
195     =n_row, normalize=True)
196     save_image(sample2.data, "images/varying_c2/%d.png" % batches_done, nrow
197     =n_row, normalize=True)
198
199 # -----
200 # Training
201 # -----
202 for epoch in range(n_epochs):
203     for i, (imgs, labels) in enumerate(dataloader):
204         batch_size = imgs.shape[0]
205
206         # Adversarial ground truths
207         valid = Variable(FloatTensor(batch_size, 1).fill_(1.0),
208         requires_grad=False)
209         fake = Variable(FloatTensor(batch_size, 1).fill_(0.0), requires_grad
210         =False)
211
212         # Configure input
213         real_imgs = Variable(imgs.type(FloatTensor))
214         #labels = to_categorical(labels.numpy(), num_columns=n_classes)
215         # -----
216         # Train Generator
217         # -----
218
219         optimizer_G.zero_grad()

```

```

214
215     # Sample noise and labels as generator input
216     z = Variable(FloatTensor(np.random.normal(0, 1, (batch_size,
217         latent_dim))))
217     label_input = to_categorical(np.random.randint(0, n_classes,
218         batch_size), num_columns=n_classes)
219     code_input = Variable(FloatTensor(np.random.uniform(-1, 1, (
220         batch_size, code_dim))))
221
222     # Generate a batch of images
223     gen_imgs = generator(z, label_input, code_input)
224     # Loss measures generator's ability to fool the discriminator
225     validity, _, _ = discriminator(gen_imgs)
226     g_loss = adversarial_loss(validity, valid)
227
228     g_loss.backward()
229     optimizer_G.step()
230
231     # -----
232     # Train Discriminator
233     # -----
234
235     optimizer_D.zero_grad()
236
237     # Loss for real images
238     real_pred, _, _ = discriminator(real_imgs)
239     d_real_loss = adversarial_loss(real_pred, valid)
240
241     # Loss for fake images
242     fake_pred, _, _ = discriminator(gen_imgs.detach())
243     d_fake_loss = adversarial_loss(fake_pred, fake)
244
245     # Total discriminator loss
246     d_loss = (d_real_loss + d_fake_loss) / 2
247
248     d_loss.backward()
249     optimizer_D.step()
250
251     # -----
252     # Information Loss
253     # -----
254
255     optimizer_info.zero_grad()
256
257     # Sample labels
258     sampled_labels = np.random.randint(0, n_classes, batch_size)
259
260     # Ground truth labels
261     gt_labels = Variable(LongTensor(sampled_labels), requires_grad=False)
262
263     # Sample noise, labels and code as generator input

```

```

262     z = Variable(FloatTensor(np.random.normal(0, 1, (batch_size,
263         latent_dim))))
264     label_input = to_categorical(sampled_labels, num_columns=n_classes)
265     code_input = Variable(FloatTensor(np.random.uniform(-1, 1, (
266         batch_size, code_dim))))
267
268     gen_imgs = generator(z, label_input, code_input)
269     _, pred_label, pred_code = discriminator(gen_imgs)
270
271     info_loss = lambda_cat * categorical_loss(pred_label, gt_labels) +
272     lambda_con * continuous_loss(
273         pred_code, code_input
274     )
275
276     info_loss.backward()
277     optimizer_info.step()
278
279     # -----
280     # Log Progress
281     # -----
282     if epoch%10 == 0:
283         torch.save(generator, BEST_PATH+BEST_MODEL1)
284         torch.save(discriminator, BEST_PATH+BEST_MODEL2)
285         print(
286             "[Epoch %d/%d] [Batch %d/%d] [D loss: %f] [G loss: %f] [info
287             loss: %f]"
288             % (epoch, n_epochs, i, len(dataloader), d_loss.item(), g_loss.
289                 item(), info_loss.item())
290         )
291
292     batches_done = epoch * len(dataloader) + i
293     if batches_done % sample_interval == 0:
294         sample_image(n_row=n_classes, batches_done=batches_done)

```