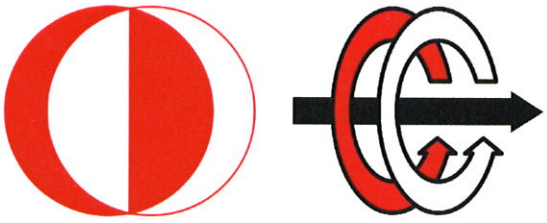


MIDDLE EAST TECHNICAL UNIVERSITY

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

05/04/2021

Contents

1 INTRODUCTION

I have performed my second internship at Kuartis Technology and Consulting, which is a company founded in 2012 by Ahmet Saraçoğlu 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

KUARTIS

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 automative 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.

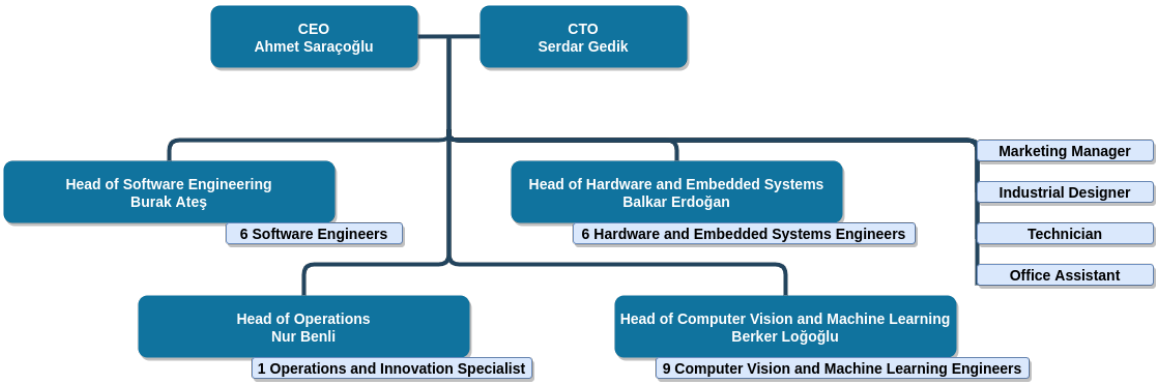


Figure 1: Administrative structure of Kuartis

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 [1–4].

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.

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 [**?**].

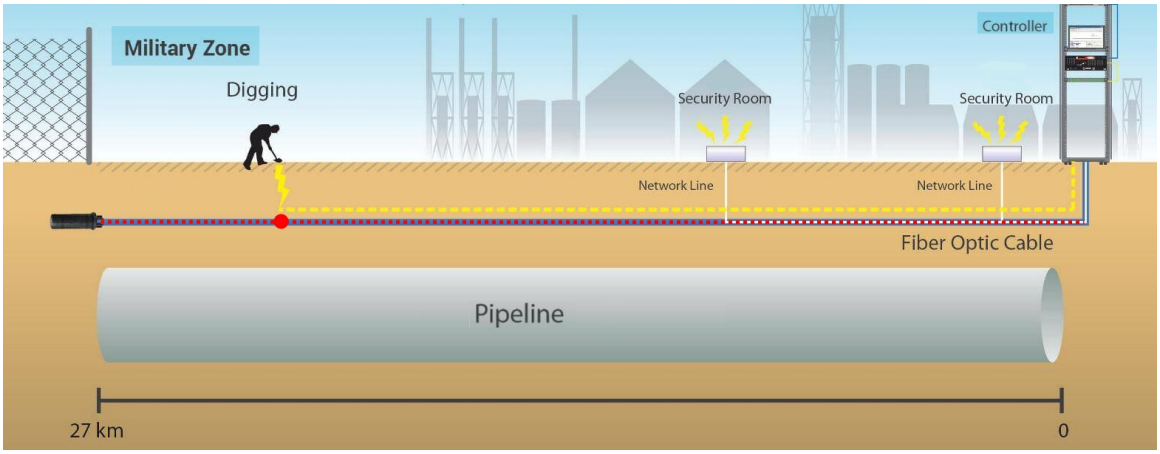


Figure 2: Illustration of pipeline security problem

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 [5].

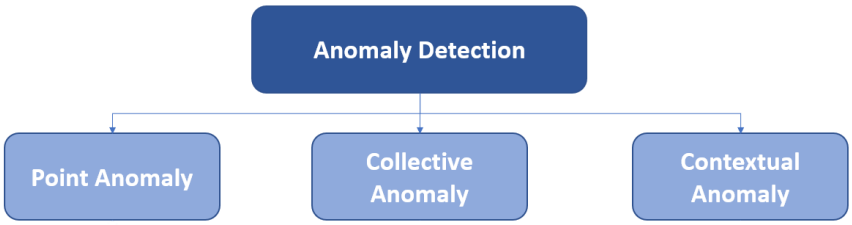


Figure 3: Types of anomalies

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 [6].

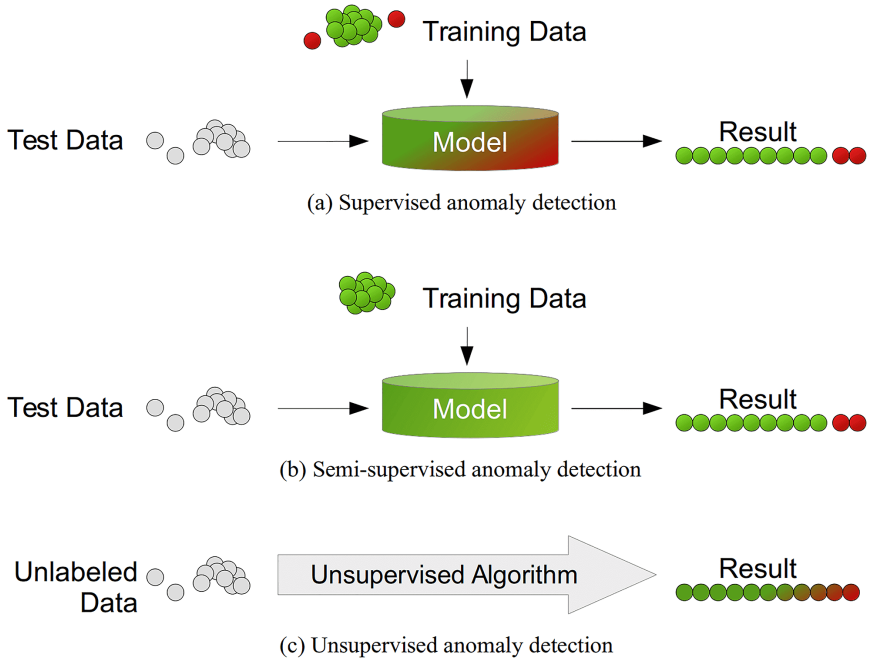


Figure 4: Different anomaly detection methods

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 [7]. 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.

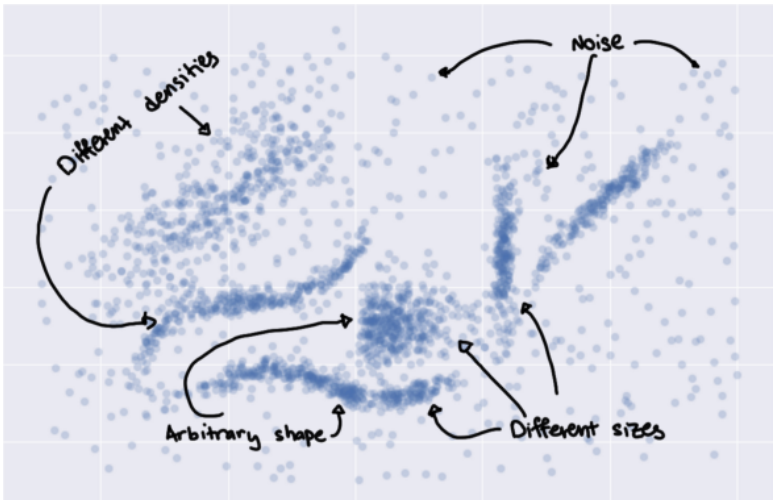


Figure 6: Challenges while clustering

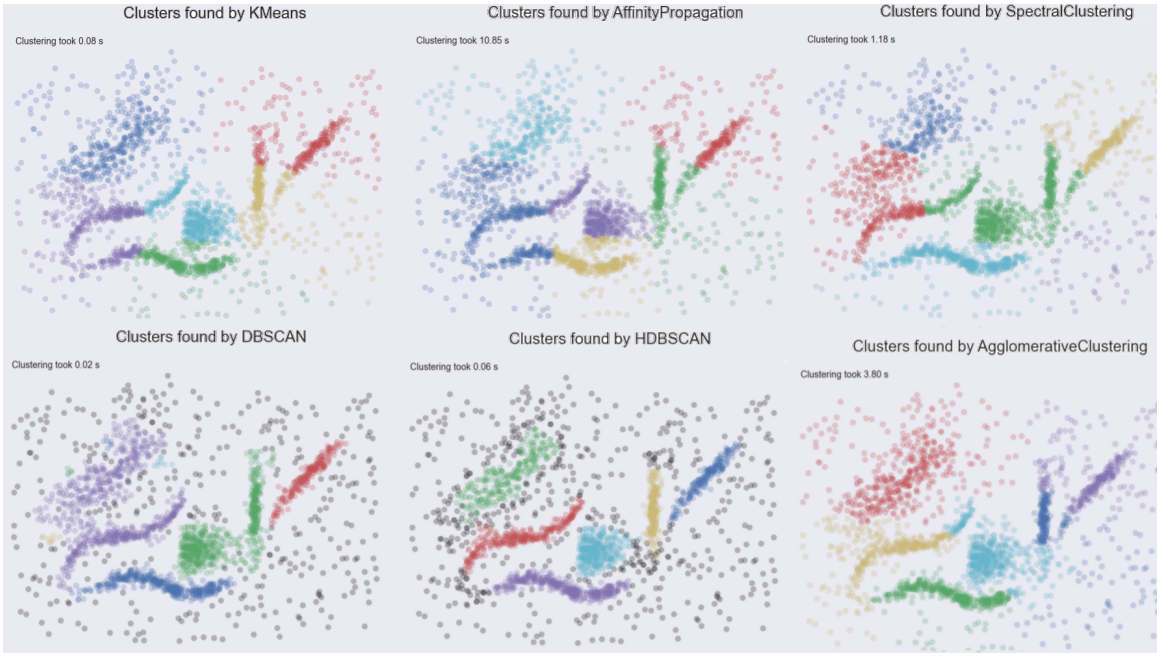


Figure 7: Comparison of different clustering algorithms

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 [8].

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 [9]. 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.

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

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 3.2.7 with other loss functions. The overall loss is given in Equation 2. Illustration of variational autoencoders are shown in Figure 8.

$$loss = \mathrm{C}\left\|\mathrm{x} - \hat{\mathrm{x}}'\right\|^2 + \mathrm{KL}\left[\mathrm{N}\left(\mu_{\mathrm{x}}, \sigma_{\mathrm{x}}\right), \mathrm{N}(0, \mathrm{l})\right] = \mathrm{C}\|\mathrm{x} - \mathrm{f}(\mathrm{z})\|^2 + \mathrm{KL}[\mathrm{N}(\mathrm{g}(\mathrm{x}), \mathrm{h}(\mathrm{x})), \mathrm{N}(0, \mathrm{I})] \tag{2}$$

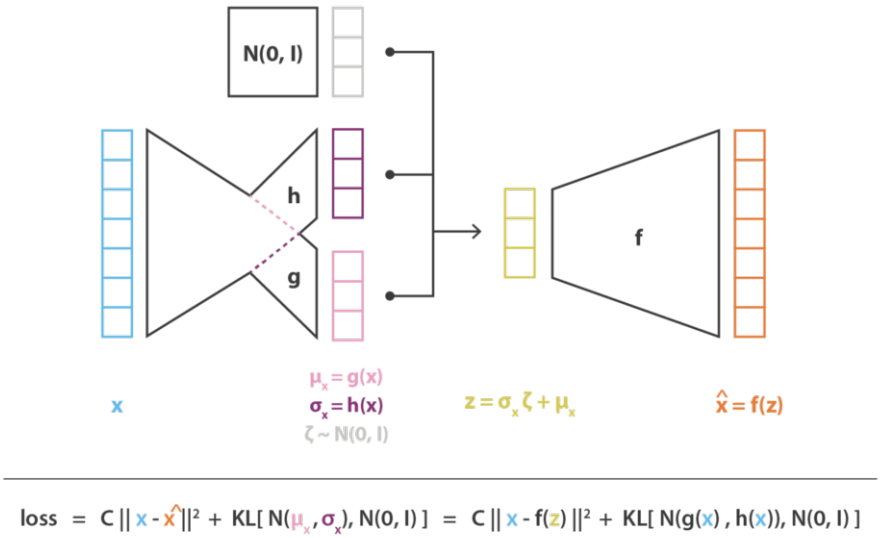


Figure 8: Illustration of Model for Variational Autoencoder

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 [10]. 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.

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

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.

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

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.

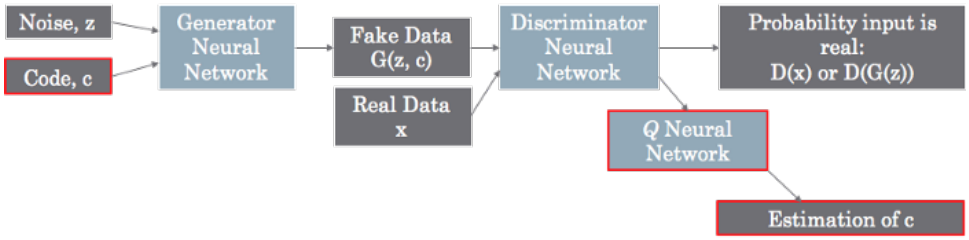


Figure 9: Illustration of Model for infoGAN

3.2.7 Optimizer and Losses

Optimizer

- **Adam Optimization**
  Adam is a method for stochastic optimization proposed by Diederik P. Kingma, and Jimmy Ba in 2014 [11]. 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 [12]. Its equation is given in Equation 5.

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

- **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 [12]. Its equation is given in Equation 6.

$$\text{CE} = -\left(y_i \log\left(\hat{y}i\right) + (1 - yi) \log\left(1 - \hat{y}_i\right)\right) \tag{6}$$

- **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 [12]. Its equation is given in Equation 7.

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

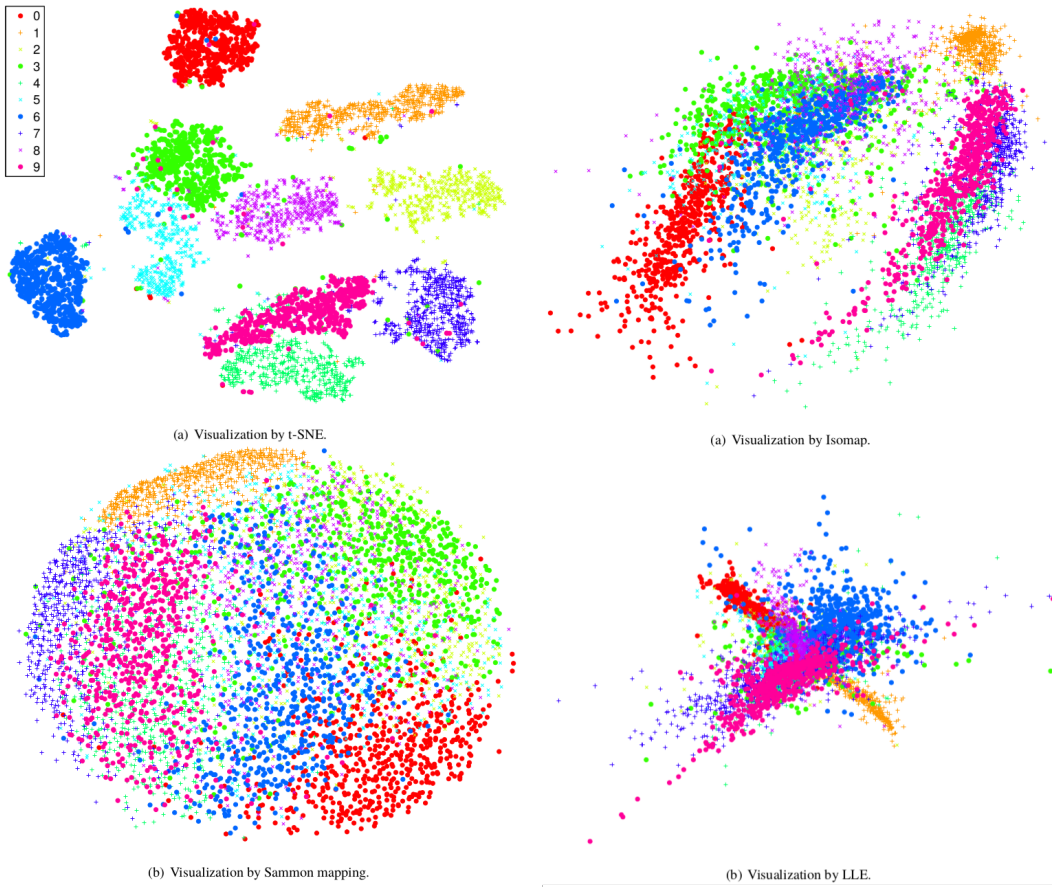
- **Kullback-Leibler Divergence Loss**
  Kullback-Leibler Divergence Loss, in a way, measures the difference between two probability distributions, hence it is also called relative entropy [12]. Its equation is given in Equation 8.

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

  While infoGAN uses MSE and Cross-Entropy, VAE uses Binary Cross-Entropy Loss and KL Divergence. All optimizations are done with Adam Optimization.

3.2.8 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 10, which shows the superiority of t-SNE clearly [13].



(a) Visualization by t-SNE.

(a) Visualization by Isomap.

(b) Visualization by Sammon mapping.

(b) Visualization by LLE.

Figure 10: 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 sch 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 to 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. Implementation of HDBSCAN can be examined from Appendix A.

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 for 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 in Appendix B and shown in Figures 12 and 13. 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 likelikehood Gaussian distribution is fitted and a threshold is decided accordingly, as seen in Figure 11.

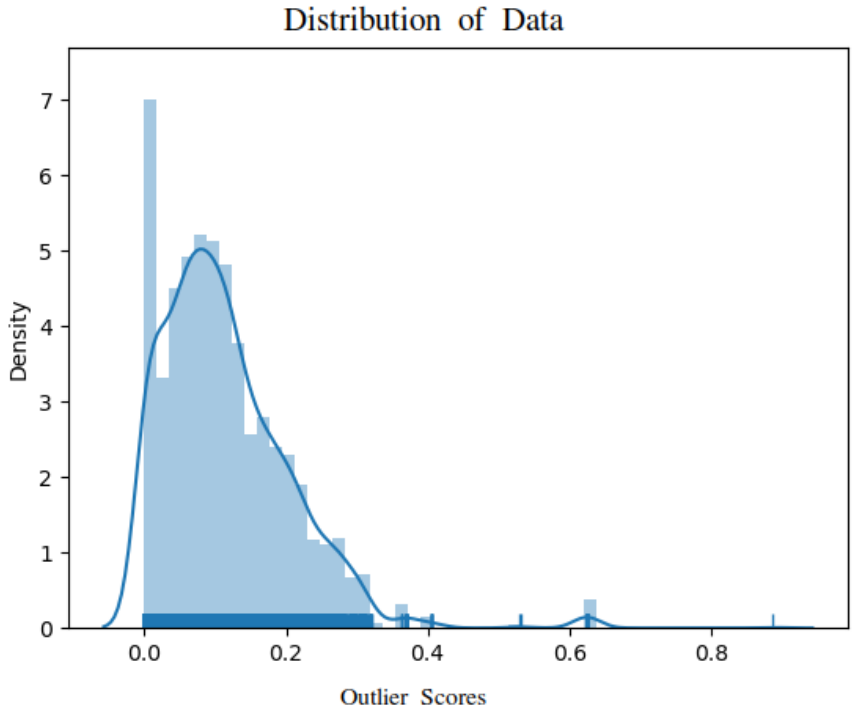


Figure 11: Distribution with a histogram and maximum likelihood Gaussian fit of the data

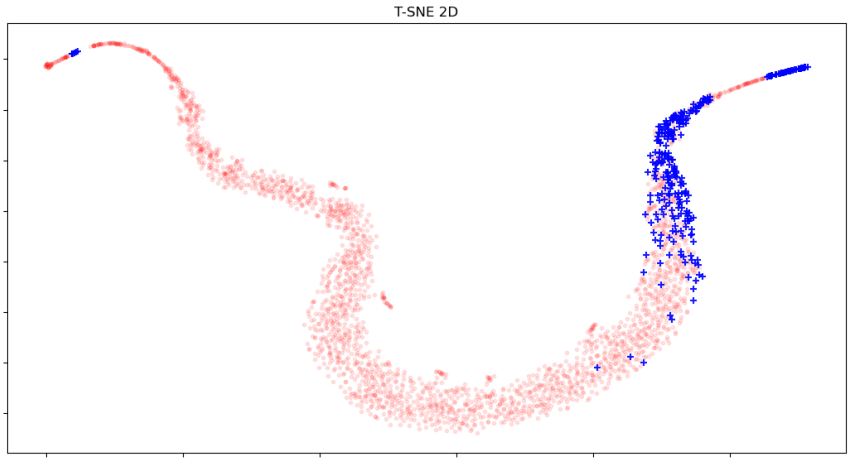


Figure 12: Two dimensional t-SNE visualisation of the dataset with outliers marked as blue

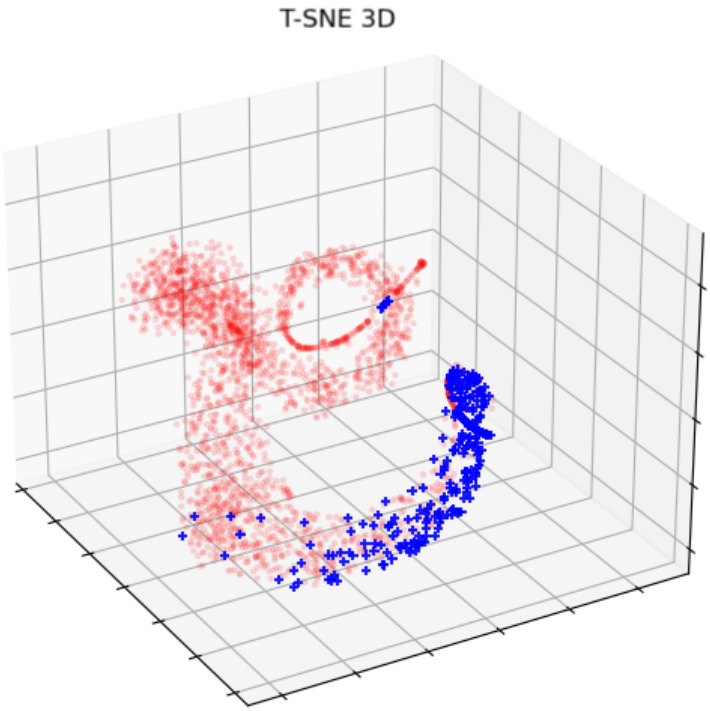


Figure 13: Three dimensional t-SNE visualisation of the dataset with outliers marked as blue

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 12. 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 14 and 15, which are presented to the contractor in external meetings.

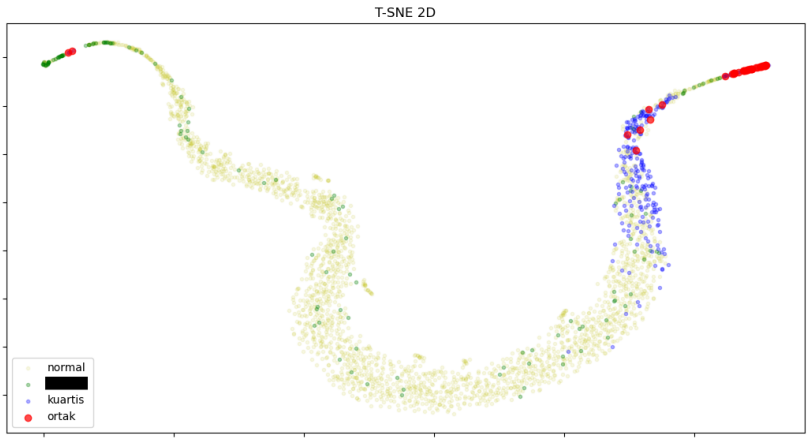


Figure 14: Two dimensional t-SNE visualisation of the dataset including contractors' decisions for comparison

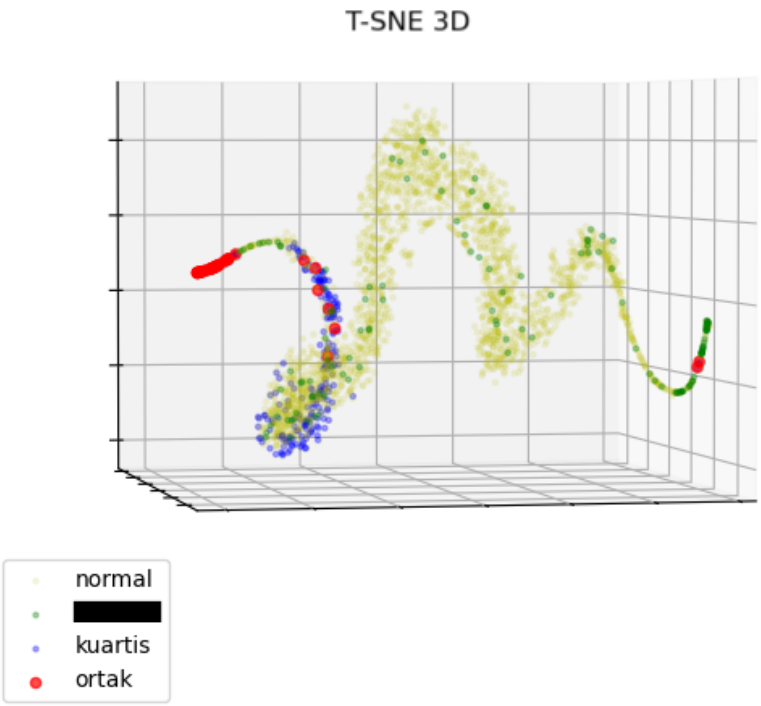


Figure 15: Three dimensional t-SNE visualisation of the dataset including contractors' decisions for comparison

Note that there is a substantial difference as seen in Figure 14, 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.

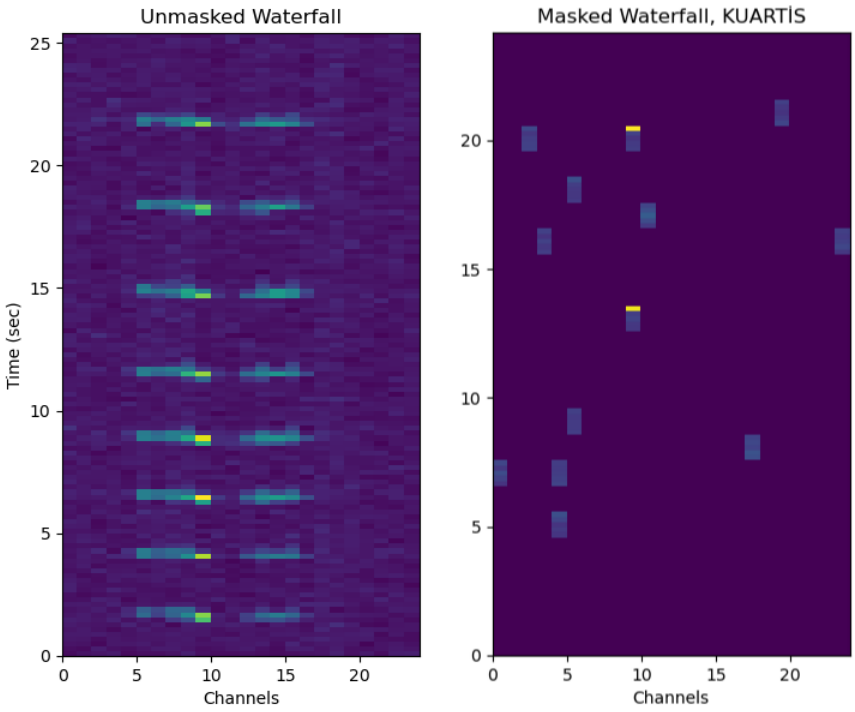


Figure 16: Comparison between waterfall image and our detections

In Figure 16, the waterfall images are shown, whose implementation is stated in Appendix C. 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 an image is created, which I can not add due to confidentiality. For the masked 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 [14]. It was suited for the task, since it is a 1D data and has anomalies in it. The training results are given in Figure 17.

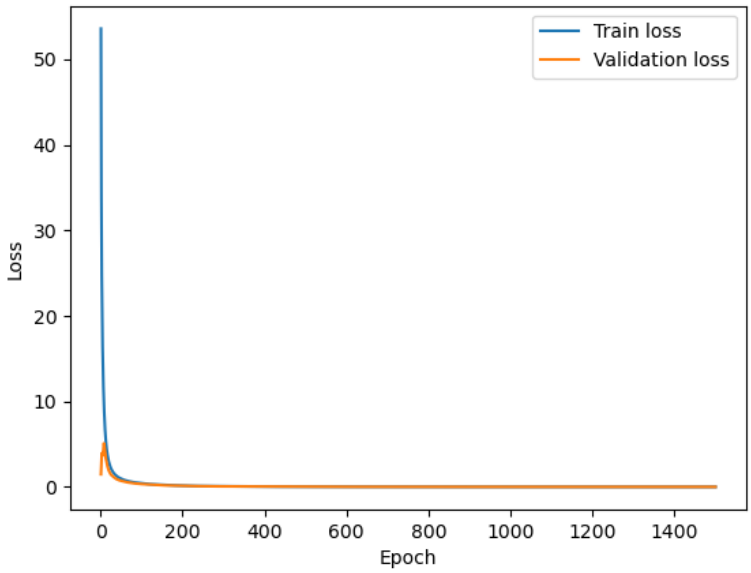
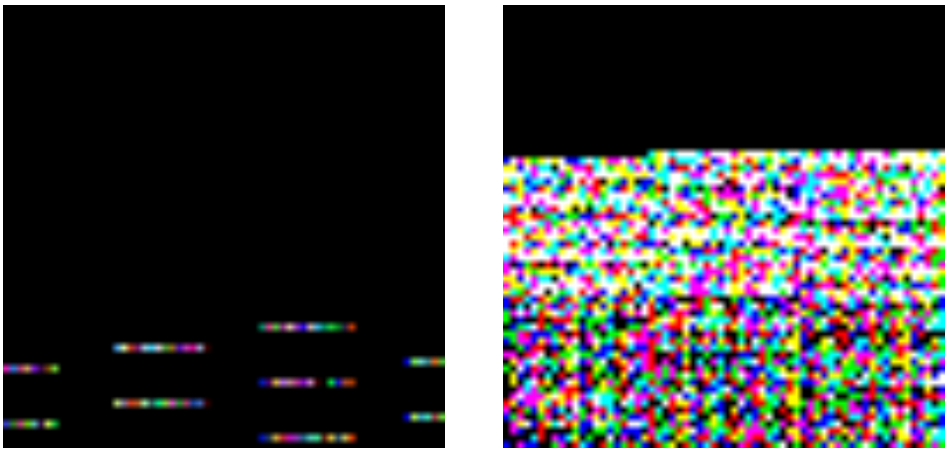


Figure 17: Traning results of VAE with a 1D toy dataset called MIMII

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 spectograms 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.

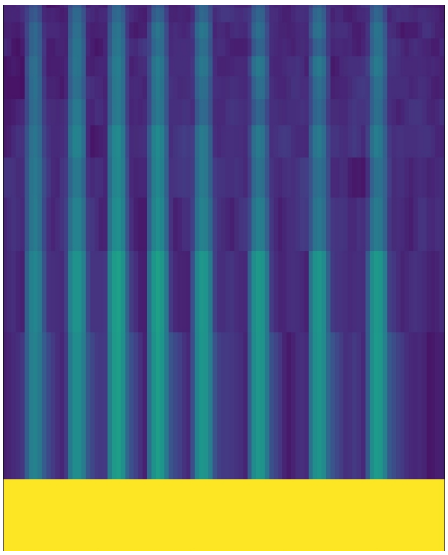
By using the real dataset, I extracted a sample image from the dataset after pre-processing steps, which can be seen from Appendix D. I used the sample image as an input to the model, and tried to see the decoded image, which is shown in Figure 18. As seen from the figure, the decoded input does not 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.



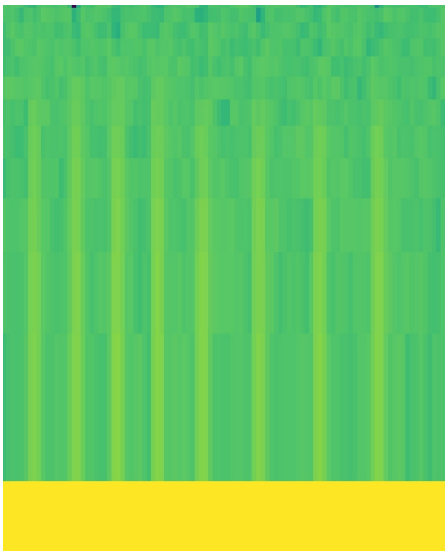
(a) Input image given to VAE (b) Decoded image taken from VAE

Figure 18: Input of the encoder and output of the decoder

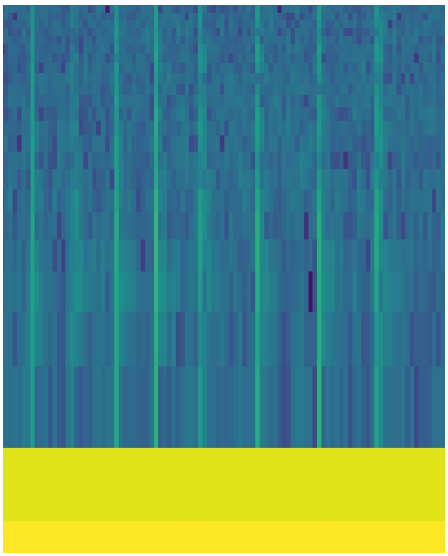
The input is an audio-like data as mentioned before. The signal has an amplitude and changes over time. To extracting useful information out of this data can be done in time domain or frequency domain. Frequency domain representation is called the spectrum and found by Fourier Transform. However, frequency 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 spectogram, 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 spectograms to be selected. There are too many choices for model using spectograms as inputs. The contractor requested that the input that will be fed to the model should be fixed, which included several types of spectograms. 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, "melspectrogram" is a mel-scaled spectogram, 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 [15].



(a) Mel-scaled spectogram

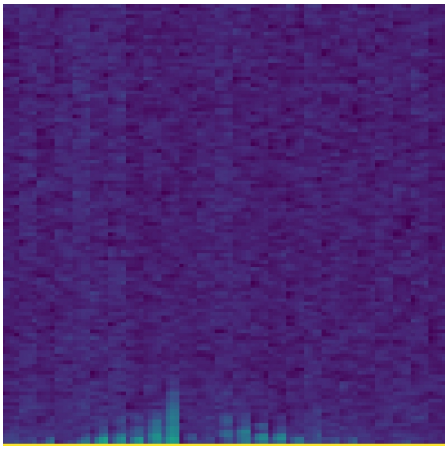


(b) MFCC

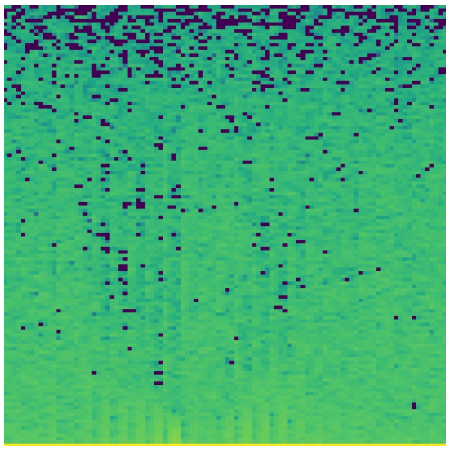


(c) STFT

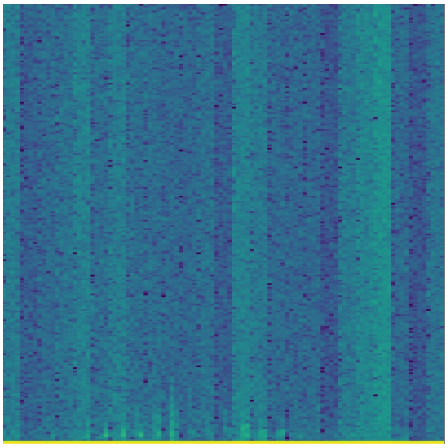
Figure 19: Features extracted from 1 channel with a frequncy range of 100Hz and a time duration of 25 seconds



(a) Mel-scaled spectogram

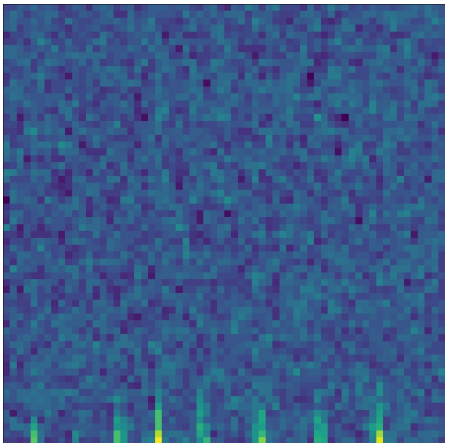


(b) MFCC

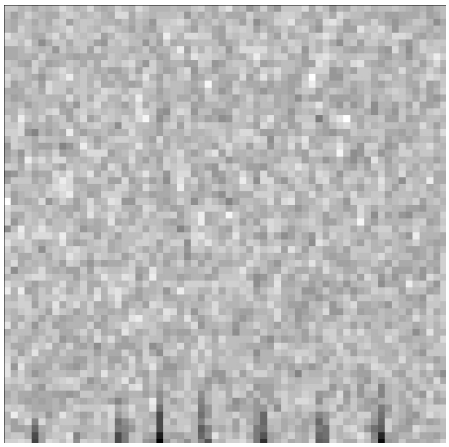


(c) STFT

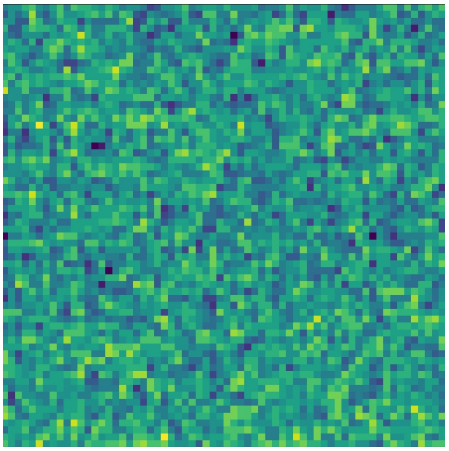
Figure 20: Features extracted from 25 channels with a frequncy range of 1kHz and a time duration of 1 seconds



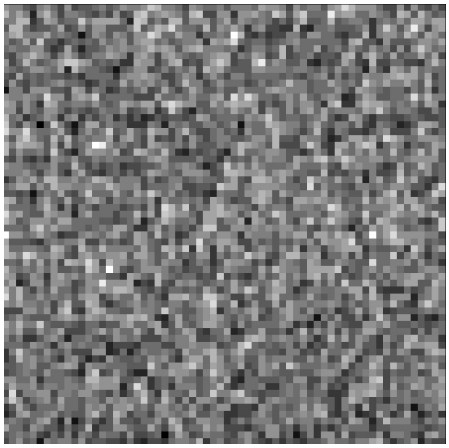
(a) Color coded mel-scaled spectogram for anomaly



(b) Gray scaled mel-scaled spectogram for anomaly



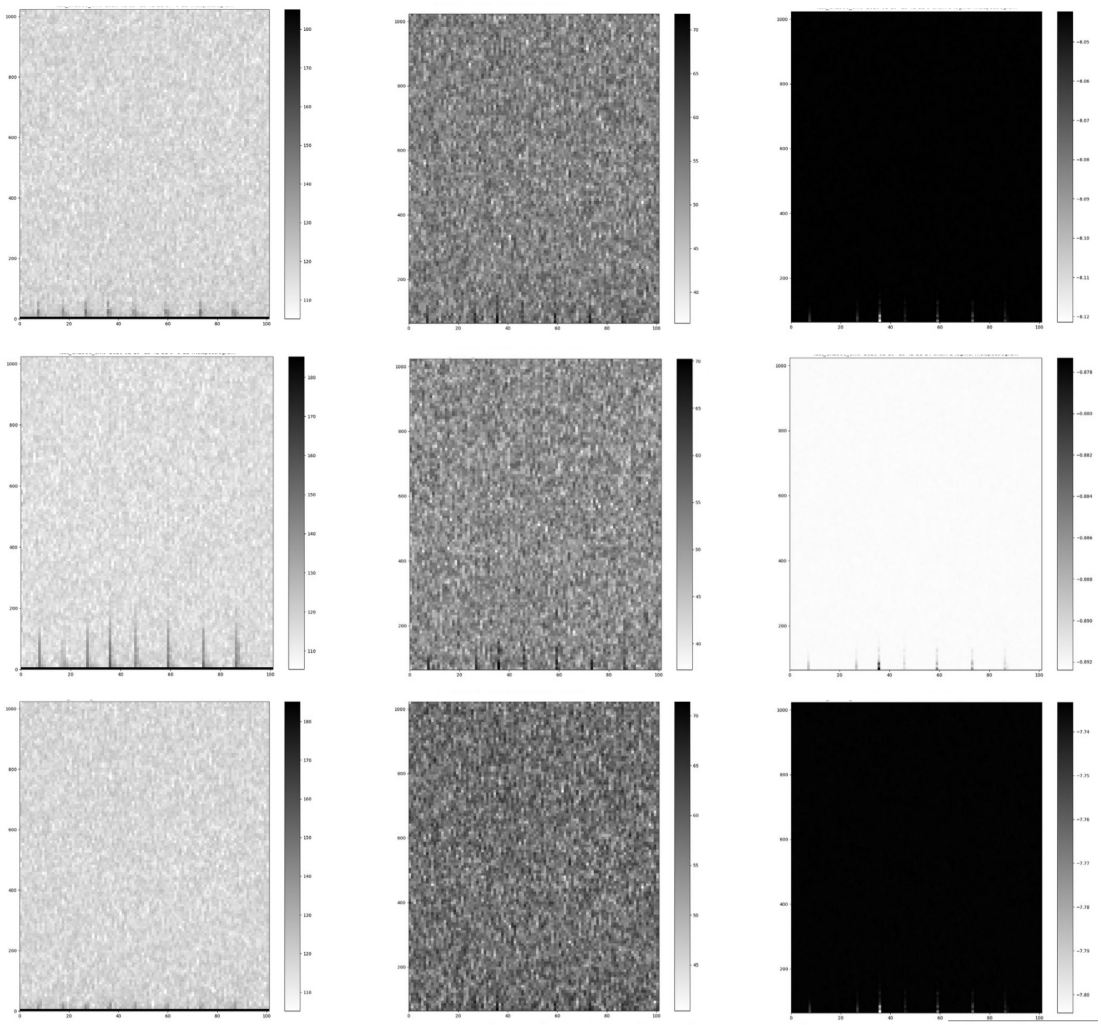
(c) Color coded mel-scaled spectogram for normal



(d) Gray scaled mel-scaled spectogram for normal

Figure 21: Features extracted from 1 channel with a frequncy range of 1kHz and a time duration of 25 seconds

As seen from the Figures 19, 20, and 21, 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, which is stated in Appendix E. From Figures 19 and 20, for different selection of spectograms, mel-scaled spectogram 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 21 for both in gray and color.



(a) Successive three channels

(b) Two and three channel differences done in 1D data

(c) Two and three channel differences done after logmel, pixel difference

Figure 22: 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-spectogram. In another version, after converting the mel-spectogram, the pixel difference is taken to exaggerate. The results of this experiment can be seen in Figure 22. Comment on that may be 1D difference adds noise to spectogram and mel-spectogram 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. While trying to implement the model, as seen in Appendix F, the training pushed the limits of my computer. For the implementation of the model, I have benefitted from a git-repository called "PyTorch-GAN" by Erik Linder-Norén [16]. 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 19 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 21. By comparing what is used before in Figure 18, 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] Tejedor, J., Macias-Guarasa, J., Martins, H.F., Pastor-Graells, J., Corredera, P., Martin-Lopez, S. Machine Learning Methods for Pipeline Surveillance Systems Based on Distributed Acoustic Sensing: A Review. Appl. Sci. 2017, 7, 841.

[5] Friedman, Ellen and Ted Dunning. "Practical Machine Learning: A New Look at Anomaly Detection." 2014.

[6] M. E. Celebi and K. Aydin, Eds., Unsupervised Learning Algorithms. Springer International Publishing, 2016.

[7] Brian S. Everitt, Sabine Landau, and Morven Leese. 2009. Cluster Analysis. Wiley Publishing.

[8] 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.

[9] Kingma, Diederik Welling, Max. 2014. Auto-Encoding Variational Bayes.

[10] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio. "Generative Adversarial Networks." 2014.

[11] Diederik P. Kingma, et al. "Adam: A Method for Stochastic Optimization." (2017).

[12] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep Learning. MIT Press, 2016.

[13] 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.

[14] Harsh Purohit, et al. "MIMII Dataset: Sound Dataset for Malfunctioning Industrial Machine Investigation and Inspection." 2019.

[15] McFee, Brian, Colin Raffel, Dawen Liang, Daniel PW Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto. "librosa: Audio and music signal analysis in python." 2015.

[16] Erik Linder-Norén, Erik Linder-Norén, (2019), GitHub repository, https://github.com/eriklindernoren/PyTorch-GAN

6 APPENDICES

A HDBSCAN Implementation

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

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

print("silhouette_score:", silhouette_score(hdb_data_t, pred_y))
print("calinski_harabasz_score:", calinski_harabasz_score(hdb_data_t, pred_y
    ))
print("davies_bouldin_score:", davies_bouldin_score(hdb_data_t, pred_y))

sns.displot(cluster_model.outlier_scores_[np.isfinite(cluster_model.
    outlier_scores_)], rug=True)
threshold = pd.Series(cluster_model.outlier_scores_).quantile(0.9)
print("threshold:", threshold)
outliers = np.where(cluster_model.outlier_scores_ > threshold)[0]
outlier_scores = cluster_model.outlier_scores_
```

B t-SNE Implementation

```
#2D TSNE
cluster_model = TSNE(2, init='pca', random_state=32,
                     learning_rate=100.0, perplexity=40.0)

Y = cluster_model.fit_transform(hdb_data_t)
print("2D Model Fitted")

fig = plt.figure(figsize=(45, 38))
ax = fig.add_subplot(111)
lo = ax.scatter(Y[class1, 0], Y[class1, 1], c='y', marker='.', alpha=0.1)
ll = ax.scatter(Y[class2, 0], Y[class2, 1], c='g', marker='.', alpha=0.3)
l = ax.scatter(Y[class3, 0], Y[class3, 1], c='b', marker='.', alpha=0.3)
a = ax.scatter(Y[class4, 0], Y[class4, 1], c='r', marker='o', alpha=0.7)
#o = ax.scatter(Y[class5, 0], Y[class5, 1], c='k', marker='o', alpha=0.7)
plt.legend((lo,ll,l,a), ('normal','','kuartis','ortak'), loc='lower left')

ax.xaxis.set_major_formatter(NullFormatter())
ax.yaxis.set_major_formatter(NullFormatter())
```

```
ax.axis('tight')
plt.title('T-SNE 2D')

#3D TSNE
cluster_model = TSNE(3, init='pca', random_state=32,
                     learning_rate=100.0, perplexity=40.0)

Y = cluster_model.fit_transform(hdb_data_t)
print("3D Model Fitted")

fig = plt.figure(figsize=(45, 38))
ax = fig.add_subplot(111, projection='3d')
lo = ax.scatter(Y[:,0][class1], Y[:,1][class1], Y[:,2][class1], c= 'y',
    marker='.', alpha=0.1)
ll = ax.scatter(Y[:,0][class2], Y[:,1][class2], Y[:,2][class2], c= 'g',
    marker='.', alpha=0.3)
l = ax.scatter(Y[:,0][class3], Y[:,1][class3], Y[:,2][class3], c= 'b',
    marker='.', alpha=0.3)
a = ax.scatter(Y[:,0][class4], Y[:,1][class4], Y[:,2][class4], c= 'r',
    marker='o', alpha=0.7)
#o = ax.scatter(Y[:,0][class5], Y[:,1][class5], Y[:,2][class5], c= 'r',
    marker='o', alpha=0.7)
plt.legend((lo,ll,l,a), ('normal','','kuartis','ortak'), loc='lower left')

ax.xaxis.set_major_formatter(NullFormatter())
ax.yaxis.set_major_formatter(NullFormatter())
ax.zaxis.set_major_formatter(NullFormatter())
ax.axis('tight')
plt.title('T-SNE 3D')
```

C Waterfall Image Implementation

```
def visualize_anomaly(data_path='',
    fig = plt.figure(figsize=(15, 8))
    t = np.arange(0, realPower.shape[0]/5, step=1/5)
    if channel is None:
        # Plot time series data for all channels
        ax = fig.add_subplot(131)
        ax.pcolormesh(realPower, vmin=0, vmax=25)
    else:
        # plot time series data for a single channel
        x = np.arange(0, channel[-1]+1,step=1)
        ax = fig.add_subplot(131)
        ax.pcolormesh(x,t,realPower[:, channel])
        plt.ylabel('Time (sec)')
        plt.xlabel('Channels')
        plt.title('Unmasked Waterfall')

    realPower = realPower[6:realPower.shape[0], :]
    masked_rP = np.multiply(prob, realPower)

    t = np.arange(0, masked_rP.shape[0]/5, step=1/5)
```

```
    if channel is None:
        # Plot time series data for all channels
        ax = fig.add_subplot(132)
        ax.pcolormesh(masked_rP, vmin=0, vmax=25)
    else:
        # plot time series data for a single channel
        x = np.arange(0, channel[-1]+1,step=1)
        ax = fig.add_subplot(132)
        ax.pcolormesh(x, t, masked_rP[:, channel])
        plt.xlabel('Channels')
        plt.title('Masked Waterfall, x')

    data = np.repeat(data, 5, axis=0)
    data = data[1:data.shape[0]-2, :]
    masked_kuarr = np.multiply(data, realPower)
    #masked_kuarr = np.multiply(prob, masked_kuarr)
    t = np.arange(0, masked_kuarr.shape[0]/5, step=1/5)
    if channel is None:
        # Plot time series data for all channels
        ax = fig.add_subplot(133)
        ax.pcolormesh(masked_kuarr, vmin=0, vmax=25)
    else:
        # plot time series data for a single channel
        x = np.arange(0, channel[-1]+1,step=1)
        ax = fig.add_subplot(133)
        ax.pcolormesh(x, t, masked_kuarr[:, channel])
        plt.xlabel('Channels')
        plt.title('Masked Waterfall, KUART S')

    # Save the figure as jpeg
    if save:
        fig.savefig(os.path.join(data_path, f+'anomaly.jpeg'), optimize=True
    )
    if return_fig:
        return fig
    else:
        plt.close()
```

D Testing VAE

```
data = torch.FloatTensor(dataset[0]).to(device)
const_data = embed_model(data)[0]
const_data = const_data.numpy()*255
const_data = const_data.reshape((64, 64, 3))
data = data.numpy()
data = data.reshape((64, 64, 3))
image1 = Image.fromarray((data).astype(numpy.uint8))
image1.show()
image2 = Image.fromarray((const_data).astype(numpy.uint8))
image2.save('my2.png')
image2.show()
```

E Mel-Spectogram Generation

```
"""LOGMEL DATA GENERATION AND VISUALIZATION"""

import os, sys
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm as cmaps
import librosa
import librosa.display
from readData import readInfo, readRawData
from PIL import Image

def channel_diff(segment0, segment1, segment2):
    if segment2 is None:
        seg01 = segment0-segment1
        seg21 = np.ones_like(seg01)
    else:
        seg01 = segment0-segment1
        seg21 = segment2-segment1
    return seg01 * seg21

def melspectrogram(data_path='', f='', save_path='', time_window=1., time_shift=1., sr=1./2000., channel=None,
                           use_channels=None, save=False, return_fig=False, nfft=512, vmin=None, vmax=None,
                           chdiff=False, mode="1D", diff=5, n_diff=3):

    if time_window:
        time_steps = int(time_window/sr)
    else:
        time_steps = rawData.shape[0]
        time_index_shift = rawData.shape[0]
    if use_channels:
        j = max(0,use_channels[0])
        j_stop = min(use_channels[-1],rawData.shape[1])
    else:
        j = 0
        j_stop = rawData.shape[1]
    # Start iterating
    while j < j_stop: # for channels
        i = 0
        while i+time_steps <= rawData.shape[0]: # for time steps
            print("Channel:", j, "Time:", i*sr)
            seg1 = np.asfortranarray(rawData[i:i+time_steps, j].copy().astype(float))
            print(seg1.shape)
            if chdiff and i > diff:
                seg0 = np.asfortranarray(rawData[i:i+time_steps, j-diff].copy().astype(float))
                seg2 = np.asfortranarray(rawData[i:i+time_steps, j+diff].copy().astype(float))
                if mode == "1D":
```

```
                    if n_diff == 2:
                        rawdiff = channel_diff(seg0.reshape(-1, 1), seg1.reshape(-1, 1), None)
                    else:
                        rawdiff = channel_diff(seg0.reshape(-1, 1), seg1.reshape(-1, 1), seg2.reshape(-1, 1))
                    ps = librosa.feature.melspectrogram(y=rawdiff.flatten(), sr=sr, n_fft=nfft)
                if mode == "logmel":
                    ps0 = librosa.feature.melspectrogram(y=seg0, sr=sr, n_fft=nfft).astype(np.float64)
                    ps1 = librosa.feature.melspectrogram(y=seg1, sr=sr, n_fft=nfft).astype(np.float64)
                    if n_diff == 2:
                        ps = channel_diff(ps0, ps1, None)
                    else:
                        ps2 = librosa.feature.melspectrogram(y=seg2, sr=sr, n_fft=nfft).astype(np.float64)
                        ps = channel_diff(ps0, ps1, ps2)

            elif channel == 1:
                ps = librosa.feature.melspectrogram(y=seg1, sr=sr, n_fft=nfft)

            elif channel!=1:
                if j % channel == 0 and j!=0:
                    data = np.zeros_like(rawData[i:i+time_steps, j], dtype=float)
                    for k in range(0,channel-1):
                        data += rawData[i:i+time_steps, j-k]
                    ps = librosa.feature.melspectrogram(y=data, sr=sr, n_fft=nfft)

            if channel==1 or (j % channel == 0 and j!=0):
                ps_db = librosa.power_to_db(ps, ref=1.0)
                ps_db = ps_db[8:,:]
                #ps_db = np.repeat(ps_db, int(ps_db.shape[0]/ps_db.shape[1]), axis=1)
                #while ps_db.shape[0] != 128:
                #    ps_db = np.concatenate((ps_db, ps_db[ps_db.shape[0]-1].reshape(1,ps_db.shape[1])), axis=0)
                #while ps_db.shape[1] != 128:
                #    ps_db = np.concatenate((ps_db, ps_db[:,ps_db.shape[1]-1].reshape(ps_db.shape[0],1)), axis=1)

                ps_db = Image.fromarray(np.uint8(ps_db)).resize((64,64))
                ps_db = np.asarray(ps_db)
                ps_db = (ps_db - np.min(ps_db)) / np.ptp(ps_db)
                print(ps_db.shape)
                if vmin != np.min(ps_db) or vmax != np.max(ps_db):
                    print("Different limits!", np.min(ps_db), np.max(ps_db))

```

```
                if save:
                    if not os.path.exists(save_path):
                        os.mkdir(save_path)
                    if return_fig:
                        fig = plt.figure(figsize=(12.8, 12.8), dpi=100)
                        plt.pcolormesh(ps_db, vmin=vmin, vmax=vmax)
                        plt.set_cmap(cmaps.binary)
                        if vmin and vmax is not None:
                            plt.clim(vmin, vmax)
                        fig.savefig(os.path.join(save_path, f+'-CH_'+str(j)+'-T_'+str(i)+'-CHS_'+str(channel)+'-CHD_'+str(chdiff)+'-ND_'+str(n_diff)+'-M_'+mode+'-mel.jpeg'))
                        plt.close()
                    np.save(os.path.join(save_path, f+'-CH_'+str(j)+'-T_'+str(i)+'-CHS_'+str(channel)+'-CHD_'+str(chdiff)+'-ND_'+str(n_diff)+'-M_'+mode+'-mel.jpeg'), ps_db)

            i = i + time_index_shift # shift time
        j = j+1 # shift channels


if __name__ == "__main__":

    ROOT = ''
    SAVE = ''

    FILES = sorted([f[:-4] for f in os.listdir(ROOT) if f[-4:] == ".bin"])
    for f in FILES:
        melspectrogram(data_path=ROOT, f=f, save_path=SAVE, time_window=10., time_shift=1., sr=1./2000., channel=1,
                           use_channels=None, save=True, return_fig=True, nfft=512, vmin=0., vmax=1.,
                           chdiff=False, mode="1D", diff=5, n_diff=3)
    sys.exit("done")
```

F infoGAN Implementation

```
import os
import numpy as np
import math
import itertools

import torchvision.transforms as transforms
from torchvision.utils import save_image

from torch.utils.data import DataLoader
from torchvision import datasets
from torch.autograd import Variable

import torch.nn as nn
import torch.nn.functional as F
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,
       is_grayscale=True, balanced=False, fresh_start=True, data_type='mel')
42
43 dataloader = torch.utils.data.DataLoader(train_set, batch_size=batch_size,
       shuffle=True)
44
45 print("Data Loaded")
46 BEST_PATH = "/content/drive/My Drive/SAMPLE/"
47 if not os.path.exists(BEST_PATH):
48     os.mkdir(BEST_PATH)
49 BEST_MODEL1 = "best_infoGAN_generator.pth"
50 BEST_MODEL2 = "best_infoGAN_discriminator.pth"
51
52 def weights_init_normal(m):
53     classname = m.__class__.__name__
54     if classname.find("Conv") != -1:
55         torch.nn.init.normal_(m.weight.data, 0.0, 0.02)
56     elif classname.find("BatchNorm") != -1:
57         torch.nn.init.normal_(m.weight.data, 1.0, 0.02)
58         torch.nn.init.constant_(m.bias.data, 0.0)
59
60
61 def to_categorical(y, num_columns):
62     """Returns one-hot encoded Variable"""
63     y_cat = np.zeros((y.shape[0], num_columns))
64     y_cat[range(y.shape[0]), y] = 1.0
65
```

```
    return Variable(FloatTensor(y_cat))


class Generator(nn.Module):
    def __init__(self):
        super(Generator, self).__init__()
        input_dim = latent_dim + n_classes + code_dim

        self.init_size = img_size // 4  # Initial size before upsampling
        self.l1 = nn.Sequential(nn.Linear(input_dim, 128 * self.init_size ** 2))

        self.conv_blocks = nn.Sequential(
            nn.BatchNorm2d(128),
            nn.Upsample(scale_factor=2),
            nn.Conv2d(128, 128, 3, stride=1, padding=1),
            nn.BatchNorm2d(128, 0.8),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Upsample(scale_factor=2),
            nn.Conv2d(128, 64, 3, stride=1, padding=1),
            nn.BatchNorm2d(64, 0.8),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Conv2d(64, channels, 3, stride=1, padding=1),
            nn.Tanh(),
        )

    def forward(self, noise, labels, code):
        gen_input = torch.cat((noise, labels, code), -1)
        out = self.l1(gen_input)
        out = out.view(out.shape[0], 128, self.init_size, self.init_size)
        img = self.conv_blocks(out)
        return img


class Discriminator(nn.Module):
    def __init__(self):
        super(Discriminator, self).__init__()

        def discriminator_block(in_filters, out_filters, bn=True):
            """Returns layers of each discriminator block"""
            block = [nn.Conv2d(in_filters, out_filters, 3, 2, 1), nn.LeakyReLU(0.2, inplace=True), nn.Dropout2d(0.25)]
            if bn:
                block.append(nn.BatchNorm2d(out_filters, 0.8))
            return block

        self.conv_blocks = nn.Sequential(
            *discriminator_block(channels, 16, bn=False),
            *discriminator_block(16, 32),
            *discriminator_block(32, 64),
            *discriminator_block(64, 128),
        )
```

```
116
117         # The height and width of downsampled image
118         ds_size = img_size // 2 ** 4
119
120         # Output layers
121         self.adv_layer = nn.Sequential(nn.Linear(128 * ds_size ** 2, 1))
122         self.aux_layer = nn.Sequential(nn.Linear(128 * ds_size ** 2,
    n_classes), nn.Softmax())
123         self.latent_layer = nn.Sequential(nn.Linear(128 * ds_size ** 2,
    code_dim))
124
125     def forward(self, img):
126         out = self.conv_blocks(img)
127         out = out.view(out.shape[0], -1)
128         validity = self.adv_layer(out)
129         label = self.aux_layer(out)
130         latent_code = self.latent_layer(out)
131
132         return validity, label, latent_code
133
134
135 # Loss functions
136 adversarial_loss = torch.nn.MSELoss()
137 categorical_loss = torch.nn.CrossEntropyLoss()
138 continuous_loss = torch.nn.MSELoss()
139
140 # Loss weights
141 lambda_cat = 1
142 lambda_con = 0.1
143
144 # Initialize generator and discriminator
145 generator = Generator()
146 discriminator = Discriminator()
147
148 if cuda:
149     generator.cuda()
150     discriminator.cuda()
151     adversarial_loss.cuda()
152     categorical_loss.cuda()
153     continuous_loss.cuda()
154
155 # Initialize weights
156 generator.apply(weights_init_normal)
157 discriminator.apply(weights_init_normal)
158
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(
```

```
    itertools.chain(generator.parameters(), discriminator.parameters()), lr=lr, betas=(b1, b2)
)

FloatTensor = torch.cuda.FloatTensor if cuda else torch.FloatTensor
LongTensor = torch.cuda.LongTensor if cuda else torch.LongTensor

# Static generator inputs for sampling
static_z = Variable(FloatTensor(np.zeros((n_classes ** 2, latent_dim))))
static_label = to_categorical(
    np.array([num for _ in range(n_classes) for num in range(n_classes)]), num_columns=n_classes
)
static_code = Variable(FloatTensor(np.zeros((n_classes ** 2, code_dim))))


def sample_image(n_row, batches_done):
    """Saves a grid of generated digits ranging from 0 to n_classes"""
    # Static sample
    z = Variable(FloatTensor(np.random.normal(0, 1, (n_row ** 2, latent_dim))))
    static_sample = generator(z, static_label, static_code)
    save_image(static_sample.data, "images/static/%d.png" % batches_done, nrow=n_row, normalize=True)

    # Get varied c1 and c2
    zeros = np.zeros((n_row ** 2, 1))
    c_varied = np.repeat(np.linspace(-1, 1, n_row)[:, np.newaxis], n_row, 0)
    c1 = Variable(FloatTensor(np.concatenate((c_varied, zeros), -1)))
    c2 = Variable(FloatTensor(np.concatenate((zeros, c_varied), -1)))
    sample1 = generator(static_z, static_label, c1)
    sample2 = generator(static_z, static_label, c2)
    save_image(sample1.data, "images/varying_c1/%d.png" % batches_done, nrow=n_row, normalize=True)
    save_image(sample2.data, "images/varying_c2/%d.png" % batches_done, nrow=n_row, normalize=True)

# ----------
#  Training
# ----------
for epoch in range(n_epochs):
    for i, (imgs, labels) in enumerate(dataloader):
        batch_size = imgs.shape[0]

        # Adversarial ground truths
        valid = Variable(FloatTensor(batch_size, 1).fill_(1.0), requires_grad=False)
        fake = Variable(FloatTensor(batch_size, 1).fill_(0.0), requires_grad=False)

        # Configure input
        real_imgs = Variable(imgs.type(FloatTensor))
```

```
        #labels = to_categorical(labels.numpy(), num_columns=n_classes)
        # -----------------
        #  Train Generator
        # -----------------

        optimizer_G.zero_grad()

        # Sample noise and labels as generator input
        z = Variable(FloatTensor(np.random.normal(0, 1, (batch_size, latent_dim))))
        label_input = to_categorical(np.random.randint(0, n_classes, batch_size), num_columns=n_classes)
        code_input = Variable(FloatTensor(np.random.uniform(-1, 1, (batch_size, code_dim))))

        # Generate a batch of images
        gen_imgs = generator(z, label_input, code_input)
        # Loss measures generator's ability to fool the discriminator
        validity, _, _ = discriminator(gen_imgs)
        g_loss = adversarial_loss(validity, valid)

        g_loss.backward()
        optimizer_G.step()

        # ---------------------
        #  Train Discriminator
        # ---------------------

        optimizer_D.zero_grad()

        # Loss for real images
        real_pred, _, _ = discriminator(real_imgs)
        d_real_loss = adversarial_loss(real_pred, valid)

        # Loss for fake images
        fake_pred, _, _ = discriminator(gen_imgs.detach())
        d_fake_loss = adversarial_loss(fake_pred, fake)

        # Total discriminator loss
        d_loss = (d_real_loss + d_fake_loss) / 2

        d_loss.backward()
        optimizer_D.step()

        # ------------------
        # Information Loss
        # ------------------

        optimizer_info.zero_grad()

        # Sample labels
        sampled_labels = np.random.randint(0, n_classes, batch_size)
```

```
257
258         # Ground truth labels
259         gt_labels = Variable(LongTensor(sampled_labels), requires_grad=False
    )
260
261         # Sample noise, labels and code as generator input
262         z = Variable(FloatTensor(np.random.normal(0, 1, (batch_size,
    latent_dim))))
263         label_input = to_categorical(sampled_labels, num_columns=n_classes)
264         code_input = Variable(FloatTensor(np.random.uniform(-1, 1, (
    batch_size, code_dim))))
265
266         gen_imgs = generator(z, label_input, code_input)
267         _, pred_label, pred_code = discriminator(gen_imgs)
268
269         info_loss = lambda_cat * categorical_loss(pred_label, gt_labels) +
    lambda_con * continuous_loss(
270             pred_code, code_input
271         )
272
273         info_loss.backward()
274         optimizer_info.step()
275
276         # --------------
277         # Log Progress
278         # --------------
279         if epoch%10 == 0:
280           torch.save(generator, BEST_PATH+BEST_MODEL1)
281           torch.save(discriminator, BEST_PATH+BEST_MODEL2)
282           print(
283             "[Epoch %d/%d] [Batch %d/%d] [D loss: %f] [G loss: %f] [info
    loss: %f]"
284             % (epoch, n_epochs, i, len(dataloader), d_loss.item(), g_loss.
    item(), info_loss.item())
285             )
286
287         batches_done = epoch * len(dataloader) + i
288         if batches_done % sample_interval == 0:
289             sample_image(n_row=n_classes, batches_done=batches_done)
```